Algorithms for Data-Driven Geometric Stylization & Acceleration

by

Hsueh-Ti Derek Liu

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Department of Computer Science
University of Toronto

Abstract

In this thesis, we investigate computer algorithms for creating stylized 3D digital content and numerical tools for processing high-resolution geometric data.

This thesis first addresses the problem of geometric stylization. Existing 3D content creation tools lack support for creating stylized 3D assets. They often require years of professional training and are tedious for creating complex geometries. One goal of this thesis is to address such a difficulty by presenting a novel suite of easy-to-use stylization algorithms. This involves a differentiable rendering technique to generalize image filters to filter 3D objects and a machine learning approach to renovate classic modeling operations. In addition, we address the problem by proposing an optimization framework for stylizing 3D shapes. We demonstrate how these new modeling tools can lower the difficulties of stylizing 3D geometric objects.

The second part of the thesis focuses on scalability. Most geometric algorithms suffer from expensive computation costs when scaling up to high-resolution meshes. The computation bottleneck of these algorithms often lies in fundamental numerical operations, such as solving systems of linear equations. In this thesis, we present two directions to overcome such challenges. We first show that it is possible to coarsen a geometry and enjoy the efficiency of working on coarsened representation without sacrificing the quality of solutions. This is achieved by simplifying a mesh while preserving its spectral properties, such as eigenvalues and eigenvectors of a differential operator. Instead of coarsening the domain, we also present a scalable geometric multigrid solver for curved surfaces. We show that this can serve as a drop-in replacement of existing linear solvers to accelerate several geometric applications, such as shape deformation and physics simulation.

The resulting algorithms in this thesis can be used to develop data-driven 3D stylization tools for inexperienced users and for scaling up existing geometry processing pipelines.

Acknowledgements

This thesis is dedicated to my mentors, colleagues, friends, and my family who support me over this journey.

Words cannot express how grateful I am to my PhD advisor Alec. This journey would not have been possible without him trusting in me. Alec taught me so much about research, geometry, programming, and (embarrassingly) English. He is my role-model of how to be a researcher, a presenter, a colleague, and a great teacher. Thank you for investing in me and giving me many great life lessons. Pursuing a PhD with you was the best decision I have ever made.

My journey to geometry all started with my MS advisors: Keenan and Burak. I still remember that a friend asked me whether I want to do a PhD in my first year at CMU. I said "100% no". But things changed dramatically since I met Keenan and Burak. My passion to geometry and research really blossomed because their research is simply so inspiring and doing research under their guidance is so rewarding. So here I am.

I am fortunate to have amazing mentors, including Maks, Sanja, and Vova who introduced me to new research topics and taught me a lot about life in general.

Thank you my co-authors who have helped me to fight against the devil in the details: Ben, Chun-Liang, Dave, Derek (Bob), Eris, Francis, Honglin, Jean-Marc, Mark, Michael, Miri, Nick, Noam, Oded, Or, Rana, Sid, Silvia, Tamy, Thibault, and Zoë. It was a wonderful experience to collaborate and to learn from you all.

My journey would not have been that enjoyable without the friends at DGP, including Aravind, Abhishek, Changjian, Darren, Etienne, Gavin, Honglin, Jiannan, John, Jonathan, Josh, Karran, Lawson, Leo, Michael, Nick, Nicole, Oded, Otman, Rahul, Rinat, Risa, Sarah, Selena, Seungbae, Silvia, Thomas, Tim, Towaki, Ty, Vismay, Wenzheng, Yixin, Yun-Chun, and many others. I am also lucky to be surrounded by brilliant faculties at DGP such as Dave, David (Lindell), Eitan, and Kyros who gave me invaluable research and career advice. In addition, I want to thank John and Xuan who have helped me to deal with countless IT and administrative issues.

I am always indebted to Maks, Miri, Olga, and Sanja who offered a ton of great advice during my job search.

Thank you Misha and David (Fleet) for being my thesis committee and participating the last mileage in this journey.

Outside UofT, I was fortunate to spend a semester at École Polytechnique and met many talented individuals: Adrien, Dorian, Jean-Michel, Jing, Marie-Julie, Maxime, and Ruqi from the group led by Maks. I am also grateful for being a part of the STREAM team led by Marie-Paule.

On the personal side, I want to thank my family for all the encouragement and support. Leaving my beloved hometown – Taiwan – and studying abroad would not be possible without you all. I appreciate how welcoming you are whenever I come back. Having a place that always feels like home means a lot to me.

I am fortunate to have my aunt DY and my uncle UB who are no different from parents to me. All the lessons you gave have shaped who I am now.

Whitney and Cider are an amazing partner and a cat debugger during this period of my life. Thank you both for sharing my up and downs and reminding me how fun a life could be.

On the financial side, this thesis would not have been completed without the generous support from Adobe, Mitacs, Mary H. Beatty Fellowship, and Robert E. Lansdale / Okino Computer Graphics Graduate Fellowship.

Thank you everyone once again who have helped me to complete this important milestone in my career. I hope that one day I can pass on all the lessons I learned to help the next generation.

Contents

Chapter 1

Introduction

Stylized digital content has played a major role in visual effects, story-telling, and the metaverse industries. Algorithms, such as image stylization and non-photorealistic rendering, have drastically lowered the difficulty in creating stylized 2D content. However, creating stylized 3D content remains a major challenge. Armies of trained modelers are still required to meticulously create stylized geometric assets. This is because characterizing the style of 3D shapes is more challenging due to arbitrary topologies and curved metrics.

Although people have been developing a plethora of 3D modeling techniques to lower the difficulty in 3D content creation, they lack easy-to-use tools to support the creation of stylized assets, and a majority of the tools only support low-level operations, such as manually dragging control points on the geometry. Therefore, stylizing 3D assets remains a time-consuming process even for a trained modeler with state-of-the-art modeling tools. For people without professional training, creating a stylized 3D asset of their desire is even more difficult. This roadblock will unfortunately hinder the development of shaping the future metaverse collaboratively.

In this document, we address the above-mentioned bottlenecks by proposing a novel suite of data-driven techniques for creating stylized 3D assets. We further tackle the computational challenges in these algorithms and generic geometry processing by proposing scalable numerical tools to enable interactive performance on highly-detailed 3D assets.

1.1 Data-Driven Geometry Processing

Recent advances in geometric datasets and 3D scanning technologies have greatly increased the accessibility of geometric data. This motivates the development of *data-driven* approaches for geometric data processing. Instead of operating on a single shape, data-driven methods utilize a collection of shapes jointly to support geometry processing. This could imply transferring knowledge that existed in a few exemplars to novel 3D shapes. This could also imply acquiring high-level reasoning learned from a large collection of data. Data-driven techniques have the potential to achieve high-level shape understandings and could unlock

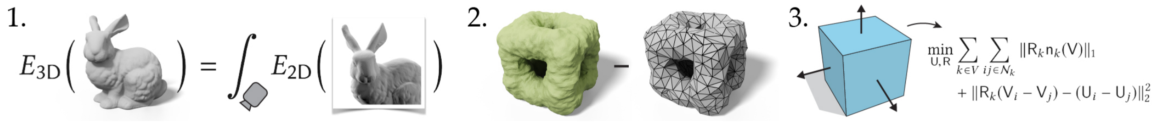

1.
$E_{3D}(\;) = \int E_{2D}(\;)$
2.
3.
$\min_{U,R} \sum_{k \in V} \sum_{ij \in \mathcal{N}_k} \|R_k n_k(V)\|_1 + \|R_k(V_i - V_j) - (U_i - U_j)\|_2^2$

Figure 1.1: We explore three different ways to define geometric styles: (1) style of its renderings, (2) difference from its coarsened counterpart, and (3) characterization on surface normals.

easy-to-use shape manipulation algorithms.

Although the accessibility of geometric datasets has raised significantly, many applications still suffer from having only a limited amount of geometric data. Existing 3D datasets have several orders of magnitude less data than image datasets. For domain-specific applications, collecting data is often the bottleneck of deploying powerful machine learning techniques. For instance, in dental applications, a dataset of oral scans usually consists of only a few hundred scans [Zanjani et al., 2019], which is smaller than the scale of multi-million image datasets. In the context of 3D stylization, collecting stylized 3D assets is even more challenging because they do not exist in reality, thus requiring many manual hours by professional artists. The fact that many real-world applications suffer from scarce geometric data motivates us to study more data-efficient methods.

In this thesis, we explore different possibilities to stylize 3D shapes without a large amount of geometric data. One possibility is to manually craft an energy and stylize shapes by minimizing the energy (e.g. Chapter. 5, 6). This approach is often the most effective one in terms of performance and supports incorporating artistic controls. However, coming up with an energy formulation is not trivial and is limited to relatively simple styles that can be characterized by mathematical equations. In response, we also explore alternatives in building generic data-driven stylization methods. For example, in Chapter. 2, we show that one can leverage image datasets to build an image generative model and then use a differentiable renderer to "translate" the image model to work on 3D geometries. This alleviates the need to have geometric data by learning from image data. Instead of leveraging image datasets, in Chapter. 4 we propose a self-supervised training framework and a mesh learning architecture that can effectively train a mesh generative model from only a few (or even one) 3D objects. These data-efficient methods show the possibilities of learning from a handful of 3D examples and generalizing to unseen geometries.

1.2 Geometric Stylization

Before delving into a core part of the thesis – creating stylized 3D shapes, we have to first understand "What is style?" and "How to characterize the style of a 3D object?".

Despite the fact that the notion of style is abstract and subjective, in art history, there is still a widely accepted definition that defines style as *a distinctive manner which permits* *the grouping of works into related categories* by Fernie [1995]. In other words, the definition of geometric styles is the method or the element that allows us to classify geometric objects into categories. This definition naturally leads us to study "What are the elements of geometric style?" Answering this question is key to quantifying style similarity. It could further enable us to develop mathematical expressions and algorithms to generate stylized geometries.

Discovering the element of geometric style is the central subject to classifying architectures from different time periods and cultures [Blumenson, 1995]. The three most common elements are the shape of architectural components, proportions, and the shape of characteristic curves in the architecture. The effectiveness of these ingredients in classifying architectures into categories has motivated the graphics community to build geometric style classifiers (e.g., [Lun et al., 2015]). However, these elements of architectural styles only cover a limited span of geometric styles. For instance, they are not applicable to identifying the style of detailed geometric textures. Furthermore, these elements are mainly used for classification. It is challenging to develop generative methods that can create novel shapes given a target style.

In the first part of the thesis, our contributions involve proposing three different characterizations of geometric styles. In addition, we present a novel suite of algorithms based on differentiable rendering, machine learning, discrete mesh correspondences, and variational methods for creating stylized geometry.

The first characterization explored in the thesis is to **characterize the style of geometry with the style of its rendered images**. Given a rich amount of image style definitions, our characterization of geometric styles enables one to "translate" those image style definitions to 3D geometry in order to quantify style similarity between a 3D shape and a 2D image. Built on top of this core idea, in Chapter. 2, we propose a fast differentiable renderer tailor-made for geometric optimization to enable generating novel 3D shapes given a desired 2D style. The overall procedure is to first render a 3D shape to a set of 2D images, apply the image stylization algorithm of choice on the rendered image, and then use our differentiable renderer to pull the pixel changes back to change the vertices of the 3D shape. We show that our method enables plug-and-play of different image stylizations or even image filters in 3D so that one can create novel 3D content as easily as we use image filters in our daily lives. In Chapter. 3, we further demonstrate the applicability of differentiable renderers for vision and machine learning tasks, beyond stylization. Specifically, we extend our differentiable render to also propagate gradients from pixels to the lighting. This enables us to construct *adversarial geometry* and *adversarial lighting conditions* to increase the robustness of deep learning image classifiers against the change of objects' appearance and natural lighting conditions. This is crucial when deploying deep learning systems to reality.

Another characterization of geometric styles presented in the thesis is to define **the style of geometric details as the difference between a shape and its simplified counterpart**.

This definition of style motivated us to formulate a learning-based upsampling problem, we call *neural subdivision* in Chapter. 4. The key idea is to train a neural network to add the geometric style by upsampling a simplified geometry back to the original input shape. To enable this training, we propose a self-supervised training framework for one to learn from a single training shape. We further propose a novel method for computing bijective correspondences between shapes with different triangulations and a novel architecture that enables machine learning on irregular triangle meshes. This neural subdivision method enables generating stylized details on a 3D object. Beyond stylization, neural subdivision achieves a better result by "smartly" figuring out how to subdivide a shape conditioned on local geometric features (e.g., curvatures) compared to traditional subdivision schemes.

Last but not least, we explore the third characterization by **defining geometric style similarity by the similarity of surface normals**. This characterization enables us to develop a mathematical expression to quantify certain geometric styles and efficiently generate novel stylized geometry with energy minimization. Specifically, in Chapter. 5, we introduce a mathematical equation that governs the style of *cubic shapes* (different from *cubism*). In Chapter. 6, we further generalize the cubic style energy to different styles prescribed by users. Tackling stylization with such a variational method leads to efficient and easy-to-control methods of stylizing 3D geometries.

1.3 Scalable Geometry Processing

Data-driven stylization is built around learning on geometric data and optimization. This inevitably involves computational problems such as solving systems of linear equations. However, existing numerical tools for geometric data are only applicable for shapes at moderate resolution. For example, the most widely used sparse Cholesky factorization in geometry processing fails to operate on meshes with millions of nodes (e.g., a fossil scan). This limits data-driven methods for modeling micro-scale geometric details (e.g., wrinkles on human faces) and large-scale 3D scenes.

We contributed to scalable geometry processing by studying *coarsening* algorithms. In Chapter. 7, we propose a *spectral coarsening* technique to compress a 3D shape while preserving the spectral properties. As these properties are crucial for many geometric operations, this technique enables one to enjoy the efficiency of working with a coarsened geometric representation without sacrificing solution quality.

Another direction to address the scalability issue is to replace existing solvers with a more scalable one. Multigrid is an ideal candidate due to its reputation of being one of the most scalable solvers. However, this ideal option cannot be deployed for geometry processing because most multigrid approaches require the domain to be regular (e.g., image grid). In Chapter. 8, we propose a novel multigrid method for curved triangle meshes that surpasses other alternatives, such as algebraic multigrid methods. This enables geometry processing algorithms to enjoy the efficiency of the multigrid method and unlock applications that require interactive frame rates on large-scale content.

This document is organized to introduce the contributions on data-driven geometric stylization from Chapter. 2-6 and scalable geometry processing from Chapter. 7-8. The text and figures have significant overlap with my publications over my PhD studies, including:

- **Hsueh-Ti Derek Liu**, Michael Tao, Alec Jacobson. "Paparazzi: surface editing by way of multi-view image processing". *ACM Transactions on Graphics* (2018)
- **Hsueh-Ti Derek Liu**, Michael Tao, Chun-Liang Li, Derek Nowrouzezahrai, Alec Jacobson. "Beyond Pixel Norm-Balls: Parametric Adversaries using an Analytically Differentiable Renderer". *International Conference on Learning Representations* (2019)
- **Hsueh-Ti Derek Liu**, Alec Jacobson, Maks Ovsjanikov. "Spectral Coarsening of Geometric Operators". *ACM Transactions on Graphics* (2019)
- **Hsueh-Ti Derek Liu**, Alec Jacobson. "Cubic Stylization". *ACM Transactions on Graphics* (2019)
- **Hsueh-Ti Derek Liu**, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, Alec Jacobson. "Neural Subdivision". *ACM Transactions on Graphics* (2020)
- **Hsueh-Ti Derek Liu**, Alec Jacobson. "Normal-Driven Spherical Shape Analogies". *Computer Graphics Forum* (2021)
- **Hsueh-Ti Derek Liu**, Jiayi Eris Zhang, Mirela Ben-Chen, Alec Jacobson. "Surface Multigrid via Intrinsic Prolongation". *ACM Transactions on Graphics* (2021)

Chapter 2

Paparazzi: Surface Editing by way of Multi-View Image Processing

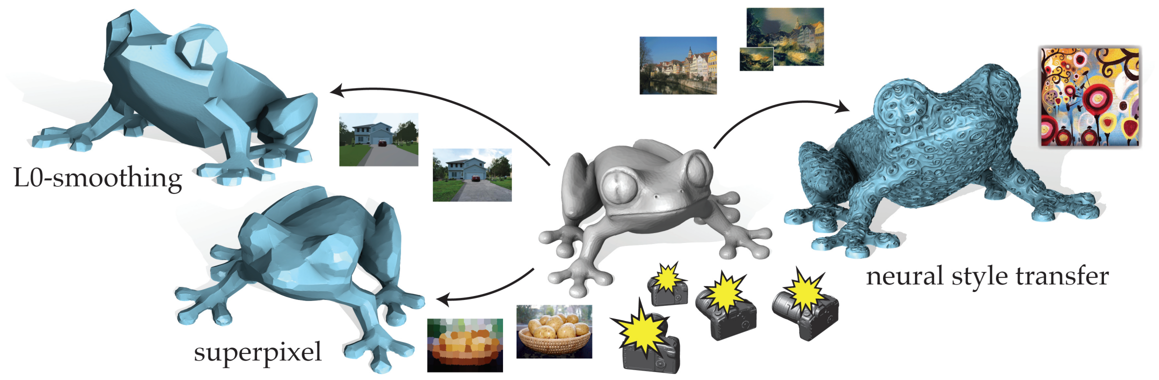


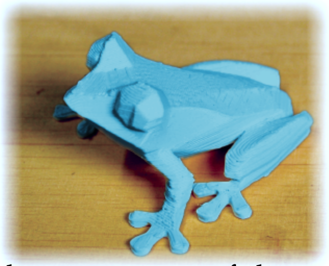
Figure 2.1: Paparazzi enables plug-and-play image processing algorithms on 3D shapes. For instance, superpixel produces a Mosaic style shape; L_0 norm makes the shape piecewise planar but preserves features such as the noses; style transfer synthesizes the artistic style from a painting to the geometry. Note that the images are just for showing the 2D effects, they are not in the Paparazzi optimization loop.

The image processing pipeline boasts a wide variety of complex filters and effects. Translating an individual effect to operate on 3D surface geometry inevitably results in a bespoke algorithm. Instead, we propose a general-purpose back-end optimization that allows users to edit an input 3D surface by simply selecting an off-the-shelf image processing filter. We achieve this by constructing a differentiable triangle mesh renderer, with which we can *back propagate* changes in the image domain to the 3D mesh vertex positions. The given image processing technique is applied to the entire shape via stochastic snapshots of the shape: hence, we call our method *Paparazzi*. We provide simple yet important design considerations to construct the *Paparazzi* renderer and optimization algorithms. The power of this rendering-based surface editing is demonstrated via the variety of image processing filters we apply. Each application uses an off-the-shelf implementation of an image processing method without requiring modification to the core *Paparazzi* algorithm.

2.1 Introduction

Decades of digital image processing research has culminated in a wealth of complex filters and effects. These filters are not only important as pre- and post-processes to other techniques in the image-processing pipeline, but also as useful tools for graphic designers, consumers, and social media users. Many of these filters depend heavily on the regular structure of pixel grids. For example, convolutional neural networks leverage this regularity to enable high-level, advanced filtering operations, such as neural style transfer.

While some simple *image* filters (e.g., Laplacian smoothing) have direct analogs for 3D *geometry* processing, building analogs of more complex filters often requires special case handling to accommodate arbitrary topologies, curved metrics, and irregular triangle mesh combinatorics found in 3D surface data. Moreover, many image processing methods are difficult to redefine for 3D geometry. For instance, artistic styles of paintings can be effectively captured and transferred across images, but it is not immediately clear how to transfer the *style* of a 2D painting to a 3D surface.



In this paper, we develop a novel tool, *Paparazzi*, to simultaneously generalize a large family of image editing techniques to 3D shapes. The key idea is to modify an input 3D surface mesh by applying a desired image processing technique on many rendered snapshots of the shape (hence, *Paparazzi*) to ensure multiview consistency. Essential to the core of *Paparazzi* is our differentiable rendering process that allows the propagation of changes in the image domain to changes in the mesh vertex positions. We first construct a stochastic multi-view optimization for generalizing energy-minimization-based image processing techniques. We then generalize this algorithm further to accommodate generic iterative image-processing filters. The renderer and its parameters are carefully constructed to consider view sampling, occlusion, and shading ambiguities. The intermediary and output triangle meshes of our optimization are filtered to ensure watertightness, facilitating downstream geometry processing applications, such as 3D printing (see inset for a 3D-printed L_0-smoothing Frog from Fig. 2.1). We demonstrate the versatility and plug-and-play nature of *Paparazzi* by generalizing a handful of image filtering techniques to 3D shapes, including guided filters, quantization, superpixel, L_0-smoothing, and neural style transfer. With *Paparazzi*, we generalize these image filters to geometry by simply plugging existing implementations in.

2.2 Related Work

Our work touches topics across visual computing, including rendering, computer vision, and geometry processing. We focus our discussion on methods similar in methodology or application.

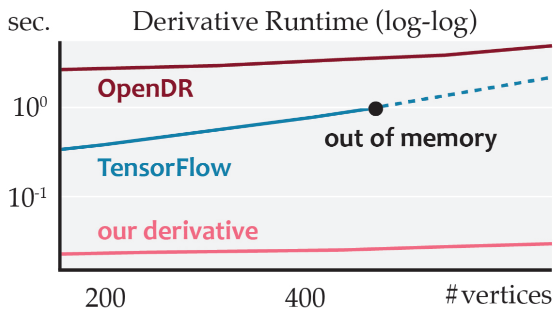


Figure 2.2: We compare runtime per iteration of our approach with two autodiff-based approaches (256×256 image). Our approach is faster and scales better.

Differential rendering Rendering is the *forward* process of synthesizing an image given information about the scene geometry, materials, lighting and viewing conditions. Solving the *inverse* problem is tantamount to solving computer vision. Loper and Black [2014] propose a fully differentiable rendering engine, OpenDR, using automatic differentiation. Their renderer is differentiable to any input parameter – not just geometry, thus more general than ours. Though they demonstrate considerable speedup over naive finite differences when differentiating with respect to many mesh vertices, our analytical derivative results in orders of magnitude speedups over their method for the case we consider in this paper (see Fig. 2.2).

Liu et al. [2017a] propose a neural network architecture to *approximate* the forward image formation process, and predicts intrinsic parameters, shape, illumination, and material, from a single image. This neural network approach is differentiable and utilizes existing data to achieve plausible material editing results. However, it is approximate and requires a large training effort for each task and for each rendering parameter. Many other differentiable or invertible renderers have been constructed for estimating materials/microgeometry [Gkioulekas et al., 2013; Zhao, 2014] or lighting conditions [Marschner and Greenberg, 1997; Ramamoorthi and Hanrahan, 2001b]. While our lighting conditions and materials are quite tame (flat shading with three directional lights), we differentiate the entire image with respect to all mesh vertex positions.

Our analytic derivatives are faster and scale better than existing automatic differentiation frameworks: OpenDR (forward mode) [Loper and Black, 2014] and the TensorFlow 3D Mesh Renderer (reverse mode, a.k.a., back propagation) [Genova et al., 2018]. On a single machine, *Paparazzi* can handle problems with more than 100,000 variables, but OpenDR and TensorFlow run out of memory on problems with few thousand and few hundreds variables respectively. In Fig. 2.2, our runtime (on a 256×256 image) is orders of magnitude faster.

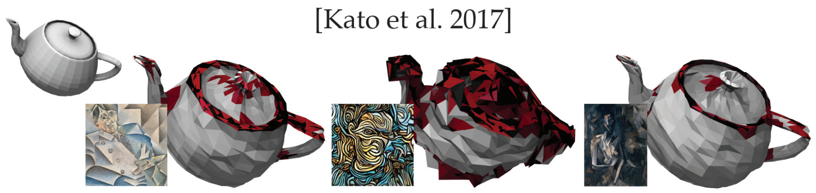


Figure 2.3: Without preventing self-intersections (red), image-driven methods may produce diverging or sub-optimal results.

Image-based Surface Editing In geometric modeling, a number of previous methods have proposed interactive or automatic methods to edit a shape by directly specifying its rendered appearance [Van Overveld, 1996; Kerautret et al., 2005; Tosun et al., 2007]. For example, Gingold and Zorin [2008] allow a user to paint darkening and brightening strokes on a surface rendered with a single light source. To overcome the shading ambiguity — a thorny issue shared by all shape-from-shading methods — they choose the deformation that would minimally change the existing surface. Instead, we overcome this ambiguity by increasing the lighting complexity. Schüller et al. [2014] take advantage of the *bas*-relief ambiguity for Lambertian surfaces to create bounded thickness surfaces that have the same appearance as a given surface from a particular view. A unique contribution of our work is that we optimize for desired appearance over all views of a surface using stochastic gradient descent.

Image-based methods are widely used for mesh simplification and accelerated rendering [Weier et al., 2017]. These methods reduce polygon meshes for rendering efficiency, but preserve their perceptual appearance [Hoppe, 1997; El-Sana et al., 1999; Luebke and Erikson, 1997; Xia and Varshney, 1996; Lindstrom and Turk, 2000; Luebke and Hallen, 2001; Williams et al., 2003]. The success of image-driven simplification demonstrates the power of image-driven methods, but only as a metric. Our method goes one step further, and utilize rendering similarity to generalize a large family of image processing techniques to surface editing.

Kato et al. [2018] generalize neural style transfer to 3D meshes by applying image style transfer on renderings. At a high level, their approach is similar to *Paparazzi* insofar as they propagate the image gradient to the geometry, but their derivatives are *approximate* while ours are *analytical*. In particular, they consider whether a pixel is covered by a certain triangle, which requires approximating a non-differentiable step function with respect to motions of mesh vertices in 3D. In contrast, we consider how a triangle's orientation (per-face normal) changes under an infinitesimal perturbation of a mesh vertex. This captures the continuous change of each pixel's color and enables analytical derivatives. Kato et al. [2018] do not prevent self-intersections (see Fig. 2.3) that inevitably arise during large deformations. Self-intersections may lead to diverging or sub-par optimization results. These differences make *Paparazzi* a more general image-driven method for creating high-quality, even fabricable 3D objects.

Shape from shading Recovering geometry from photographed (or rendered) images is known as the "shape from shading" problem. This subfield itself is quite broad, so we defer to existing surveys [Zhang et al., 1999; Prados and Faugeras, 2006] and focus on the most related methods. Given insufficient or unreliable data, shape from shading algorithms typically fall back on assumptions for regularization, such as surface smoothness [Barron and Malik, 2015], and consequently produce less detailed models. The single view shape from shading problem is made easier in the presence of per-pixel depth information, where inverse rendering can be used to *refine* the depth geometry to match the shading image [Wu et al., 2014; Or-El et al., 2016]. Shading-based depth refinement can be extended to full-shape *reconstruction* refinement if given multiple depth and shading images from different views [Wu et al., 2011; Choe et al., 2017; Robertini et al., 2017]. Gargallo et al. [2007] exactly differentiate the reprojection error function with respect to the unknown surface. Delaunoy and Prados [2011] extend this to minimize image-based regularization terms to aid in multi-view surface reconstruction. All such shape-from-shading methods are built around assumptions that the input data is captured – however unreliably – from an underlying fully realized physical shape. Our problem is similar to multi-view shading-based geometry refinement but there is a major difference – we have access to a general underlying geometric representation. We utilize this access to develop a more powerful framework that generalizes various image processing directly to 3D, rather than just geometry refinement.

Single-purpose Geometry Processing Filters In our results, we show examples of various filters being applied to geometry by simply attaching *Paparazzi* to existing image-processing code (e.g., SKIMAGE [Van der Walt et al., 2014]). Our results demonstrate that we successfully transfer these effects: for example, [Xu et al., 2011] creates piecewise-constant images, and via *Paparazzi* we use their method to create piecewise-constant appearance geometry (see Fig. 2.4 for a single-view example). Some of the image-processing filters we use for demonstration purposes have previously been *translated* to triangle meshes as single-purpose filters. For example, He and Schaefer [2013] introduce a novel edge-based Laplacian to apply L_0 regularization to meshes. Similarly, in order to create a 3D analogy of guided filters [He et al., 2010] for meshes, Zhang et al. [2015b] design a specific triangle-clustering method tailored to guided filtering. Extending texture synthesis to 3D geometry has been an active research area [Gu et al., 2002; Lai et al., 2005; Turk, 1991; Wei and Levoy, 2001; Landreneau and Schaefer, 2010; Knöppel et al., 2015; Dumas et al., 2015], the typical challenges lie in accounting for curvature and irregular mesh discretizations.

Rather than increment the state of the art for any specific mesh filter or application (e.g., denoising), our technical contribution is a suite of algorithms to provide a general-

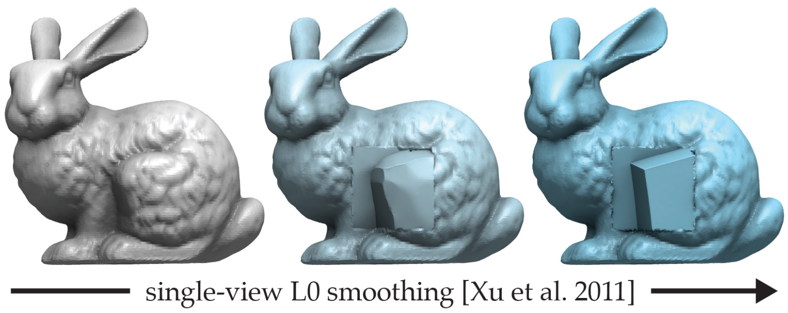


Figure 2.4: The Bunny is optimized so the gradient of its rendered image for a single view is minimal in an L_0 sense.

purpose, plug-and-play machinery for directly applying a large family of image processing filters to 3D. We evaluate our results in terms of how well *Paparazzi* correctly applies the image-processing effect to the input geometry.

2.3 Overview

Paparazzi is a general-purpose framework that allows a user to apply an image-processing filter to 3D geometry, without needing to redefine that filter for surfaces or even implement new code for triangle meshes. The input to our method is a non-self-intersecting, manifold triangle mesh and a specified image-processing technique. The output is a non-self-intersecting deformation of this mesh, whose appearance has undergone the given processing. The core insight is that if we can pull gradients of an image energy with respect to pixels back from rendered images to vertices then we can apply gradient-descent-based optimization with respect to vertex positions. We first describe the well-posed scenario where the specified image-processing technique is described as an energy optimization in the image domain. Subsequently, we show that a slight modification to our energy-based method enables us to generalize to the class of iterative image-processing techniques.

2.3.1 Energy-Based Image Filters

Many image editing algorithms can be formulated as the minimization of a differentiable, image-domain energy E. In the ideal setting, we extend any such energy minimization to surfaces by considering the integral of this energy applied to *all possible* rendered images for a space of camera "views":

$$\min_V \int_{i \in \text{views}} E(R_i(V)),$$

where R_i is a function mapping a mesh with vertices V to an image.

Minimization is straightforward to write as a gradient descent with respect to vertices

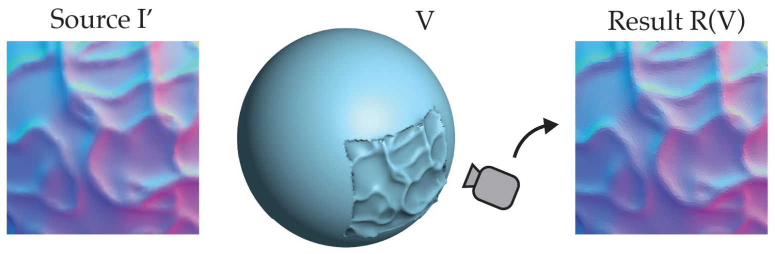


Figure 2.5: A sphere is deformed to match a source image I' from a camera.

using the chain rule:

$$V \leftarrow V - \int_{i \in \text{views}} \frac{\partial E}{\partial R_i} \frac{\partial R_i}{\partial V}. \tag{2.1}$$

The space of views can be tuned to satisfy problem-specific needs. For instance, it could be as small as a single front-on camera or as large as the space of all cameras where some amount of geometry is visible. We defer discussion of a good default choice until Sec. 2.6.

Consider the toy example of an energy-based image editing algorithm visible in Fig. 2.5, where the energy E is simply the L_2 distance to a rendering of another shape. In the optimization, we consider only a single view of a sphere. After gradient descent, the sphere's geometry is deformed so that this one view is imperceptibly similar to the source image. For a single view, our method only changes vertices that affect that view's rendering, which makes this result appear like a decal.

The presence of the Jacobian $\partial R_i / \partial V$ exposes the main requirement of the renderer R: differentiability with respect to vertex positions. In this paper, we present an as-simple-as-possible renderer that is *analytically* differentiable with mild assumptions and is effective for generating high-quality geometries (see Sec. 2.5).

2.3.2 Stochastic Multi-view Optimization

When we look at a single view, the analytical derivative $\partial R / \partial V$ enables the direct generalization of image processing algorithms to geometries via Eq. (2.1), but evaluating this integral over a continuous space or distribution of views is challenging.

We handle this issue by borrowing tools from the machine learning community, which has extensively applied gradient descent to energies involving integrals or large summations when training deep networks [Ruder, 2016; Bottou et al., 2016]. Rather than attempt to compute the integrated gradient exactly, we leverage stochastic gradient descent (SGD) and update the geometry with the gradient of a small subset of the views, as few as one. As is common in the machine learning literature, we apply momentum both to regularize the noise introduced by using a stochastic gradient and to improve general performance with the *Nesterov-Adam* method [Dozat, 2016], a variant of gradient descent that combines momentum and Nesterov's accelerated gradient. Our stochastic multi-view optimization

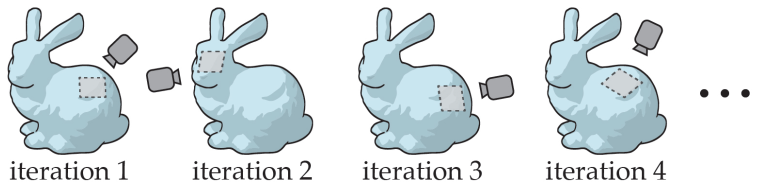


Figure 2.6: We sample a view per iteration in the multi-view optimization.

is summarized in Alg. 9.

As the mesh deforms according to the optimization, the quality of triangles may degrade and self-intersections may (and inevitably will) occur. We discuss the importance of interleaving a mesh quality improvement stage in our optimization loop in Sec. 2.4.3.

2.3.3 Iterative Image Filters

With only minor modifications we may generalize our method from energy-based deformations to the realm of iterative filters, generically defined as an iterative process acting in the image domain:

$$I_{j+1} \leftarrow f(I_j).$$

We replace the energy gradient with the update induced by a single iteration of the filter by replacing the derivative $\partial E / \partial R$ with the difference $\Delta R := f(R) - R$. The update to the mesh vertex positions becomes:

$$V \leftarrow V - \int_{i \in \text{views}} \left(f(R_i(V)) - R_i(V) \right) \frac{\partial R_i}{\partial V}.$$

This generalization works when no individual application of the iterative filter modifies the image by too much. In our experiments this is close enough to smoothness to allow for our method to converge to a geometry that matches the result of the filter from different views[1]. If a single application of the filter creates a dramatic effect, then our optimization can accommodate by using a smaller step size γ. The resulting algorithm is therefore Alg. 9.

Before demonstrating results (Sec. 2.7), we describe the considerations we made when designing our renderer and parameters.

2.4 Design Considerations

By working with renderings of geometry, image processing techniques may be applied in their native form on the rendered images of pixels. This allows *Paparazzi* to immediately

[1]When f is an first order explicit Euler over an energy's gradient this method is precisely our original method.

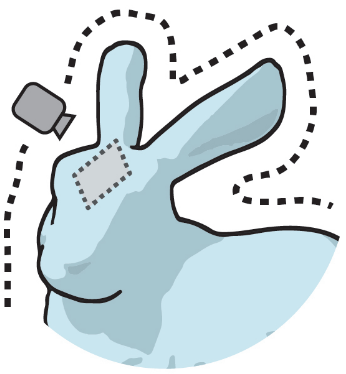
Algorithm 1: Energy-Based Method

```
Input:
$V_0, F_0$   triangle mesh
$R$          renderer
$E$          image energy
Output:
$V, F$       optimized triangle mesh
1. begin
2.     $V, F \leftarrow V_0, F_0$;
3.     $O \leftarrow$ offset surface from $V_0, F_0$;
4.     while $E$ not converge do
5.         $i \leftarrow$ sample camera from $O$;
6.         $\partial E/\partial R_i \leftarrow$ compute image derivative for camera $i$ ;
7.         $\partial R_i/\partial V \leftarrow$ compute shape derivative for camera $i$;
8.         $V \leftarrow V - \gamma \frac{\partial E}{\partial R_i} \frac{\partial R_i}{\partial V}$;
9.         $V, F \leftarrow CleanMesh(V, F)$;
```

generalize to a large family of image processing techniques, but shifts the burden to designing a rendering setup that faithfully captures the geometry and presents it in a meaningful way to the image processing technique. Where and how we render the geometry will have a large impact on the quality of the results.

2.4.1 Camera Sampling

A good camera placement strategy should "see" every part of the surface with equal probability. A surface patch that is never seen by any camera will not be changed. On the other hand, a surface patch that is visible to too many cameras will be updated faster than other surface regions and result in discontinuity between surface patches.



According to these two criteria: full coverage and uniform sampling, *Paparazzi* uniformly samples cameras on an *offset surface* at distance σ, whose points are at a fixed distance from the given shape, facing along inward vertex normals. This placement ensures that we are less biased to certain views, have smooth camera views around sharp edges, and have full coverage for most of the shapes. Increasing and decreasing σ only affects the near plane because we use an orthographic camera. We set σ to 5% of the shape's bounding box diameter for small-scale deformation and 25% for large-scale deformation (see Sec. 2.6 for values of each experiment).

Algorithm 2: Iterative Filter Method

```
Input:
$V_0, F_0$   triangle mesh
$R$          renderer
$f$          image filter
Output:
$V, F$       optimized triangle mesh
1. begin
2.     $V, F \leftarrow V_0, F_0$;
3.     $O \leftarrow$ offset surface from $V_0, F_0$;
4.     for iteration ¡ max. iterations do
5.         $i \leftarrow$ sample camera from $O$;
6.         $\Delta R_i \leftarrow f(R_i(V)) - R_i(V)$;
7.         $\partial R_i/\partial V \leftarrow$ compute shape derivative for camera $i$;
8.         $V \leftarrow V - \gamma \Delta R_i \frac{\partial R_i}{\partial V}$;
9.         $V, F \leftarrow CleanMesh(V, F)$;
```

2.4.2 Lighting & Shading

Our image-driven surface editor is designed so that inputs and outputs are both 3D shapes, thus the quality of intermediate renderings are only important insofar as we achieve the desired output geometry. We propose an as-simple-as-possible default renderer for *Paparazzi*, but take special care to avoid lighting and shading ambiguities that would cause artifacts during optimization.

Shading Ambiguity It is well-known that a single directional light is not sufficient to disambiguate convex and concave shapes and slope directions (see, e.g., [Liu and Todd, 2004]). Just as this *shading ambiguity* can confuse human observes, it will confuse *Paparazzi*'s optimization. One reason is that a single directional light is not sufficient to disambiguate convex/concave shapes and slope directions (see Fig. 2.7).

Our simple solution, inspired by *photometric stereo* [Woodham, 1980], is to increase the complexity of the lighting. By specifying three axis-aligned direction lights with R, G, B colors respectively, we effectively render an image of the surface normal vectors. This avoids the shading ambiguity.

Gouraud Ambiguity A more subtle, but nonetheless critical ambiguity appears if we follow the common computer graphics practice of smoothing and interpolating per-vertex lighting/normals within a triangle. When rendering a triangle mesh (especially if low-resolution), Gouraud shading [Gouraud, 1971] or Phong shading [Phong, 1975] will make the shape appear smoother than the actual piecewise-linear geometry. While this illusion is convenient for efficient rendering, its inherent averaging causes an ambiguity. A surface

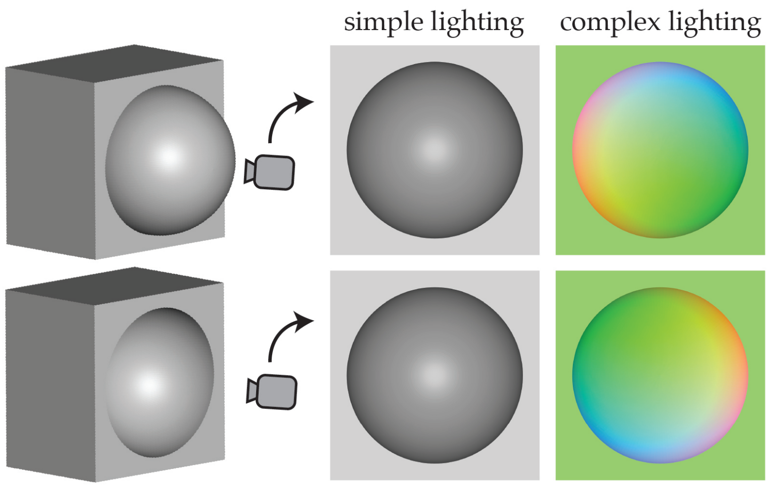


Figure 2.7: Shading ambiguity. Convex/concave shapes may result in the same image under a single directional light (middle). Adding complexity to the lighting could resolve the shading ambiguity.

with rough geometry can still produce a smooth rendering under Gouraud shading. We refer to this the *Gouraud ambiguity*. Using Gouraud shading during optimization in *Paparazzi* would immediately introduce a null space, leading to numerical issues and undesirable bumpy geometries[2] (see Fig. 2.8). Instead, we propose using flat shading. This is, in a sense, the most *honest* rendering of the triangle mesh's piecewise-linear geometry.

2.4.3 Mesh Quality

So far, our rendering choices are sufficient for *Paparazzi* to make small changes to the surface geometry but as the mesh continues to deform, the quality of individual triangles will degrade and even degenerate. Furthermore, local and global self-intersections may occur. For many of the image processing filters we consider, we would like sharp creases and corners to emerge during surface deformation which may not be possible without re-meshing.

These challenges and specifications bear resemblance to re-meshing needed during surface-tracking fluid simulation. We borrow a state-of-the-art tool from that community, El Topo [Brochu and Bridson, 2009], a package for robust explicit surface tracking with triangle meshes. It enables *Paparazzi* to generate manifold watertight meshes without self-intersections and refines the mesh in areas of high curvature which allows us to introduce sharp features without worrying about lacking enough degrees of freedom. In Fig. 2.9, we can see that shape optimization requires El Topo to mitigate issues with bad mesh quality and self-intersections, even though the rendered results are comparable.

El Topo handles two types of operations essential to the success of *Paparazzi*: those that are critical for maintaining manifold, non-intersecting meshes and those associated with

[2]We use a fast finite difference approach for the optimization under Gouraud shading.

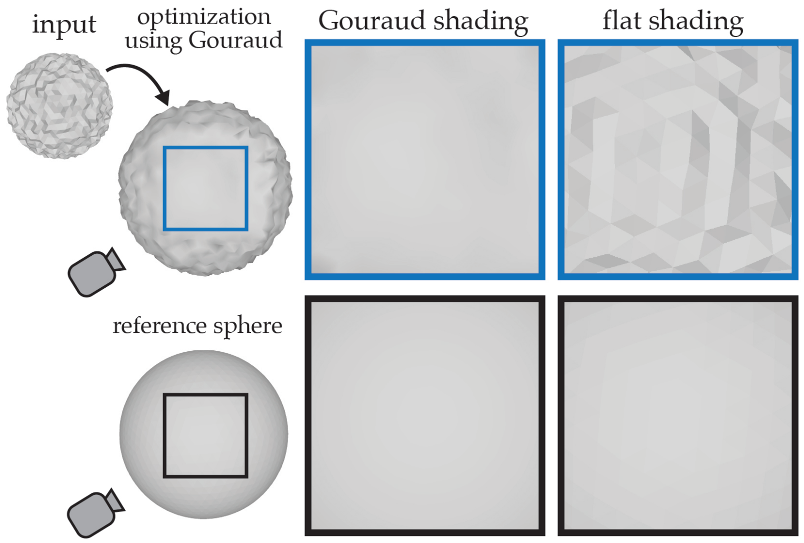


Figure 2.8: Gouraud ambiguity. Given a bumpy sphere (left), we minimize image Dirichlet energy under Gouraud shading to get the smoothed sphere (middle). Comparing the renderings of the smoothed region, we observe Gouraud ambiguity which the rendering of a non-smoothed sphere is very similar to the rendered smooth sphere (left column), but flat shading reveals the difference (right column).

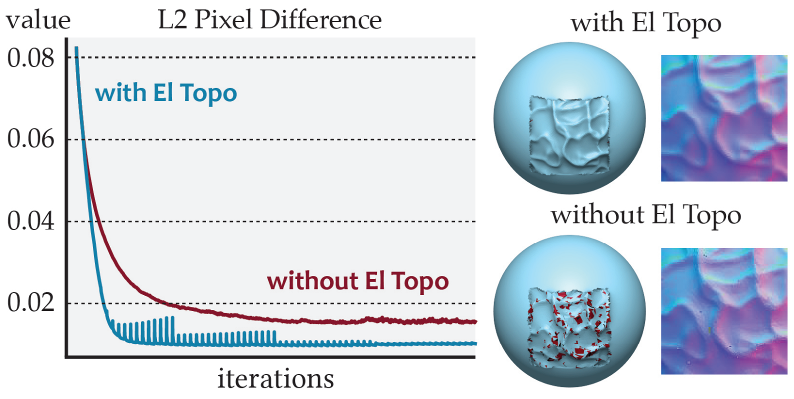


Figure 2.9: Shape optimization without El Topo (bottom) may cause self-intersections (red), despite the fact that the rendering is similar to the one with El Topo (top). Additionally, optimization with El Topo results in lower error (left). Note that the peaks in the plot are where we perform El Topo.

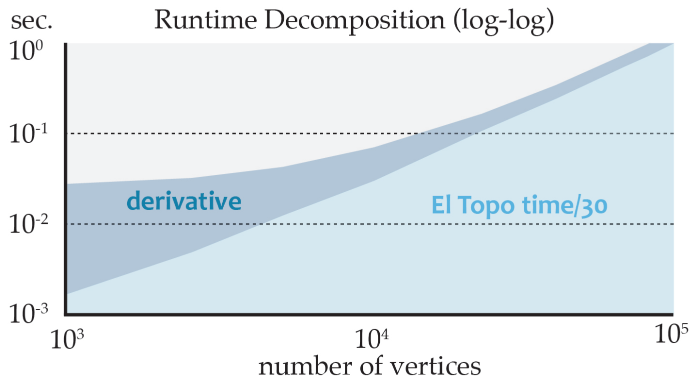


Figure 2.10: We show the decomposition of our total runtime, excluding the image processing part. The top half is the time for the derivative computation; The bottom half shows 1/30 of the El Topo runtime.

triangle quality. We feed El Topo the current non-self-intersecting mesh vertices and faces as well as the new desired vertex locations. El Topo checks whether triangles will collide or come too close together throughout the continuous motion from the current positions to the desired positions. This can either result in repulsive forces or in topological changes depending on user-defined thresholds. In order to improve the quality of the mesh, which improves the robustness of collision detection, El Topo does standard mesh improvement operations such as edge splitting and edge flipping, which improve the aspect ratios of triangles without affecting the overall topology of the mesh. El Topo also subdivides and decimates meshes to improve the quality of the mesh near high and low curvature regions respectively by keeping edge angles between a user-defined interval.

Remeshing and collision handling are essential for maintaining high-quality meshes during and after optimization, but it is also time-consuming — especially compared to our derivative computation. This is visible in Fig. 2.10, where we can see that El Topo dominates the total runtime. Because deformations between any individual iteration are generally small, in practice we invoke El Topo every 30 iterations, providing an empirically determined balance between computation time and optimization performance.

2.5 Differentiable Renderer

So far, we have discussed the design considerations of *Paparazzi*. As intermediate renderings are not the outputs, we have the flexibility in designing a suitable renderer which addresses the aforementioned challenges and, more importantly, is differentiable. In particular we present a differentiable renderer that allows us to analytically compute $\partial R/\partial V$, and to generalize image processing to 3D geometry.

2.5.1 Visibility

The rendered image of a triangle mesh with flat shading is continuous away from silhouettes, occluding contours, and triangle edges. It is differentiable *almost everywhere*: at all points on the image plane lying inside a triangle, but not across triangle edges or vertices (a set with measure zero). Therefore, we assume infinitesimal changes of surface positions will not change the visibility because in practice we only have finite image resolution on computers.

Visibility may change under large vertex perturbations eventually incurred in our optimization loop. Fortunately, due to the efficiency of z-buffering in the real-time rendering engine OpenGL, updating visibility can be handled efficiently by re-rendering the shape once every iteration.

2.5.2 Analytic derivative

Given our design choices of local illumination and Lambertian surface, we render m directional lights with directions $\hat{\boldsymbol{\ell}}_i$ (a unit vector in $\mathbb{R}^3$) having corresponding RGB colors $\{c_i^R, c_i^G, c_i^B\} \in [0,1]$. The output color $\{r_p^R, r_p^G, r_p^B\} \in [0,1]$ at a pixel p is computed by

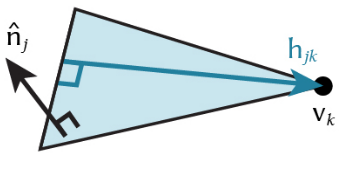
$$r_p^R = \mathsf{n}_j \cdot \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^R, \quad r_p^G = \mathsf{n}_j \cdot \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^G, \quad r_p^B = \mathsf{n}_j \cdot \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^B,$$

where n_j is a unit vector in $\mathbb{R}^3$ representing the normal of the jth face of the triangle mesh V, and the jth face is the nearest face under pixel p.

Without loss of generality, we only write the red component r_p^R in our derivation as r_p^G, r_p^B share the same formulation. We can analytically differentiate this formula with respect to the vertex positions to form each row $\partial r_p^R / \partial V \in \mathbb{R}^{3|V|}$ of the *sparse* Jacobian matrix. Only the position of each vertex $\mathsf{v}_k \in \mathbb{R}^3$ at a corner of the jth triangle contributes:

$$\frac{\partial r_p^R}{\partial \mathsf{v}_k} = \begin{cases} \frac{\partial \mathsf{n}_j}{\partial \mathsf{v}_k} \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^R & \text{if vertex } k \text{ is corner of triangle } j, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the 3×3 Jacobian of face normals n_j over triangle vertices v_k, $\partial \mathsf{n}_j / \partial \mathsf{v}_k$, can be computed *analytically*. Note that moving v_k in the triangle's plane will not change n_j. Also, in the limit, moving along n_j only changes n_j in the h_{jk} direction where $\mathsf{h}_{jk} \in \mathbb{R}^3$ is the "height" vector: the shortest vector to the corner v_k from the line of the opposite edge. This implies that the Jacobian must be some scalar multiple of $\mathsf{h}_{jk}\mathsf{n}_j^\top$. This change is inversely proportional to $\|\mathsf{h}_{jk}\|$, the distance of v_k to



the opposite edge, which means:

$$\frac{\partial \mathsf{n}_j}{\partial \mathsf{v}_k} = \frac{\mathsf{h}_{jk}\mathsf{n}_j^\top}{\|\mathsf{h}_{jk}\|^2}.$$

2.6 Implementation

In our experiments, we normalize shapes to fit in a unit-radius cube centered at the origin and upsample shapes to have 10^5-10^6 vertices in order to capture geometric detail. By default, we use a square, orthographic camera with a 0.5-wide field of view placed at an offset of $\sigma = 0.1$, where the units are those of the OpenGL canonical view volume. The offset surface meshes have 10^3-10^4 vertices. By default we use three directional lights in red, green, and blue colors respectively along each local camera axis, which is equivalent to rendering the surface normals in the camera frame.

We implement our derivative computation in Python using vectorized Numpy operations and calls to OpenGL for rendering and rasterization. We use libigl [Jacobson et al., 2018] and the Meshmixer [Schmidt and Singh, 2010] for mesh upsampling and offset surface computation. We test our implementation on a Linux workstation with an Intel Xeon 3.5GHz CPU, 64GB of RAM, and an NVIDIA GeForce GTX 1080 GPU.

2.6.1 Off-the-Shelf Image Processing Filters

We have designed *Paparazzi* to plug-and-play with existing image processing filters. We are able to use open, readily available, implementations of image-space filters with minimal effort. To evaluate our method we used a handful of off-the-shelf image filters. We use a Python implementation of the fast guided filter [He and Sun, 2015], found at `http://github.com/swehrwein/python-guided-filter`. For SLIC superpixels [Achanta et al., 2012] we use the implementation in the popular Python image processing library, Skimage. We translated a Matlab implementation of image smoothing with L_0-smoothing [Xu et al., 2011] from `http://github.com/soundsilence/ImageSmoothing` into Python. For neural style transfer [Gatys et al., 2016], we followed the corresponding PyTorch [Paszke et al., 2019] tutorial with minor modification to extract the $\partial E/\partial R_i$ gradient. We implemented simple image quantization with a fixed palette ourselves (see review in [Ozturk et al., 2014]).

Applying these filters to 3D geometry requires no modification of the *Paparazzi* algorithms. The caller either provides the $\partial E/\partial R_i$ gradient to use Alg. 9 for energy-based methods or provides the filter f as a function handle to use Alg. 9 for iterative methods. From a user's prospective, trying various filters is quite effortless. In Fig. 2.11 we demonstrate how *Paparazzi* produces different results for the various smoothing-type filters we tested. Each result respects the intent of the particular image processing filter, but now applied to a 3D surface.

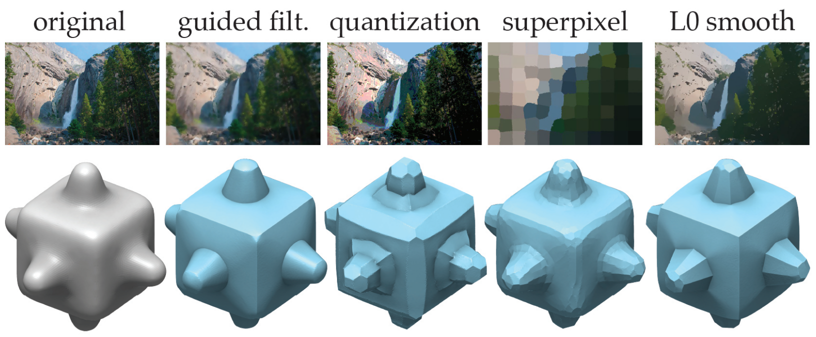


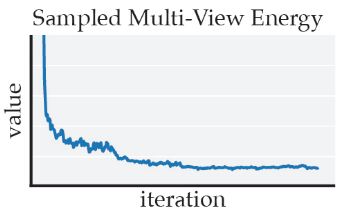
Figure 2.11: *Paparazzi* allows direct generalization of image processing to 3D, thus different image editing effects can be directly transferred to 3D shapes.

Table 2.1: For each *Example* image-processing filter on a mesh with $|V|$ vertices, rendering $|R|^2$ pixels for *Iters.* iterations, we list average seconds per iteration to call the image processing filter or gather its gradient ΔR, invoking El Topo (slowest), and computing $\partial R/\partial V$ (bold; fastest). Finally, we report *Total* time to make a result in minutes.

Example	$\|V\|$	$\|R\|$	*Iters.*	ΔR	El Topo	$\partial R/\partial \mathbf{V}$	*Tot.*
Guided	26K	128	3K	1.03*s*	0.13*s*	**0.08s**	64*m*
L_0	40K	256	3K	0.45*s*	0.17*s*	**0.12s**	40*m*
SLIC	43K	256	3K	0.10*s*	0.28*s*	**0.14s**	28*m*
Quant.	48K	256	1K	0.12*s*	0.27*s*	**0.16s**	11*m*
Neural	143K	128	10K	0.03*s*	1.73*s*	**0.48s**	390*m*

2.7 Results

In Table 2.1, we decompose our runtime in terms of subroutines: derivative computation, image processing, and mesh cleaning using El Topo. Our differentiation is orders of magnitude faster than previous methods (see Table 2.1). Mesh cleaning is the bottleneck for high-resolution meshes (see Fig. 2.10). Because our multi-view optimization processes the rendering of local patches multiple times, the runtime performance of the particular input image processing method is amplified by our approach (e.g., simple quantization is much faster than neural style transfer).



For energy-based filters, evaluating the *integrated* multi-view energy would require rendering and evaluating from all possible camera views. Even approximating this with a finite number of views every iteration would be too expensive. Instead, to evaluate convergence behavior we can set up a camera at a fixed view and evaluate its visible energy as the multi-view optimization stochastically decreases the (unmeasured) integrated energy. The energy of the particular rendering does not represent the value of multi-view energy, but the convergence behavior implies the convergence of multi-view energy. In the inset, we show the convergence of the

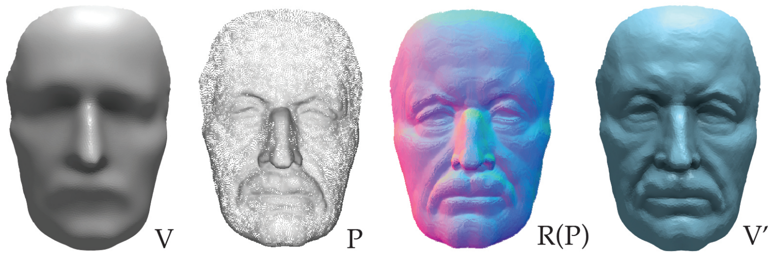


Figure 2.12: We transfer geometric details from the input point cloud P to the input shape V through rendering $R(P)$ the point cloud.

neural style transfer energy.

2.7.1 Evaluation on Image Filters

We evaluate *Paparazzi* according to its ability to reproduce the effect of 2D filters on 3D shapes, instead of its domain-specific success for any specific application (e.g., denoising). In Fig. 2.11 we see that changing the image processing filter indeed changes the resulting edited shape. Guided filter correctly achieves an edge-preserving smoothing effect; quantization makes surface patches align with predefined normals[3]; superpixel creates superfaces; and L_0-smoothing results in piecewise planar geometry. We can see that these filters are correctly transferred to 3D geometry in a plug-and-play fashion. All the while, our remeshing ensures the output mesh is watertight.

We start by considering a simple yet powerful differentiable energy – the L_2 pixel difference. Because its derivatives $\partial E/\partial R$ are known, we apply Alg. 9 to generalize this energy to 3D shapes. By caching the rendering of one geometry we can use this energy minimization to transfer its appearance to another geometry. Compared to dedicated mesh transfer tools (e.g., [Takayama et al., 2011]) we do not require the source geometry to be another triangle mesh: simply anything we can render. In Fig. 2.12 we can transfer details from point cloud P to a triangle mesh V by minimizing the L_2 image difference $\|R(P) - R(V)\|^2$. We use a simple splat rendering but this example would immediately fit from more advanced point cloud rendering (see, e.g., [Kobbelt and Botsch, 2004]).

The source geometry could be a mesh with defects such as self-intersections and holes. In Fig. 2.13, we transfer a triangle soup's appearance on top of a smooth surface reconstruction created using robust signed distance offsetting [Barill et al., 2018]. The result is a new watertight mesh with the appearance of the messy input, which previous mesh repairing methods have difficulty preserving [Attene, 2010, 2016]. This mesh is now fit for 3D printing.

In the following Figures 2.14–2.18, the images on the left are given as a reference to show the corresponding image processing and are not used to make the surface editing re-

[3]A *palette* containing the edge and face normals of a cube.

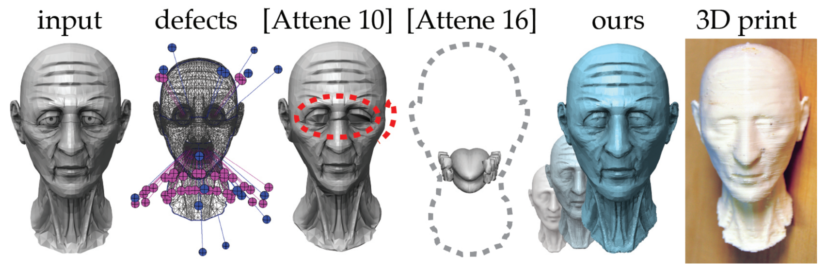


Figure 2.13: We can repair a mesh with disconnected parts and self-intersections by creating a coarse proxy and then applying detail transfer. These defects are visualized by MeshmMixer [Schmidt and Singh, 2010] and proved challenging for dedicated mesh cleaning methods.

sults. By construction our 3D inputs and outputs mirror the "analogies" of Hertzmann et al. [2001], but unlike that method we have direct access to the underlying image processing algorithm.

We now explore a more complicated energy — the Neural Style energy. Recently, inspired by the power of Convolutional Neural Network (CNN) [Krizhevsky et al., 2012], *Neural Style Transfer* has been a popular tool for transferring artistic styles from painting to other images [Gatys et al., 2016]. The goal is to generate a stylized image given a content image and a reference style image. Gatys et al. [2016] define the total energy to be the summation of content and style energies, where the content energy encourages the stylized output image to have similar image structure with the content image and the style energy encourages the output to have similar features with the reference style image. Note that the features are defined using the filter responses of a CNN across different layers.

Transferring artistic styles to 3D geometries is challenging because the redefinition of 2D painting styles on 3D is unclear. With *Paparazzi*, we can generalize it by applying the image neural style transfer on the renderings. Because the image gradient can be computed by differentiating the CNN, we can use Alg. 9 to generate stylized shapes. In Fig. 2.14, *Paparazzi* transfers the style of 2D paintings to 3D via growing geometric textures (we provide implementation detail about image neural style in App. 2.9.1).

A large portion of image processing algorithms are not based on energy minimization but applying iterative procedures. These algorithms may not have a well-defined energy or, even if they do, may not have an easily computable gradient. Fortunately, *Paparazzi* provides an effortless way to generalize a variety of iterative image filters using Alg. 9. The high-level idea is to perform image update on the rendering once, and update the shape once based on how the image change due to the image update.

Guided filters [He et al., 2010] compute the filtered image output by considering the content of a guidance image, the guidance image can be another image or the input itself. He et al. [2010] shows that the guided filter is effective in a variety of image processing applications including edge-aware smoothing, detail enhancement, image feathering, and

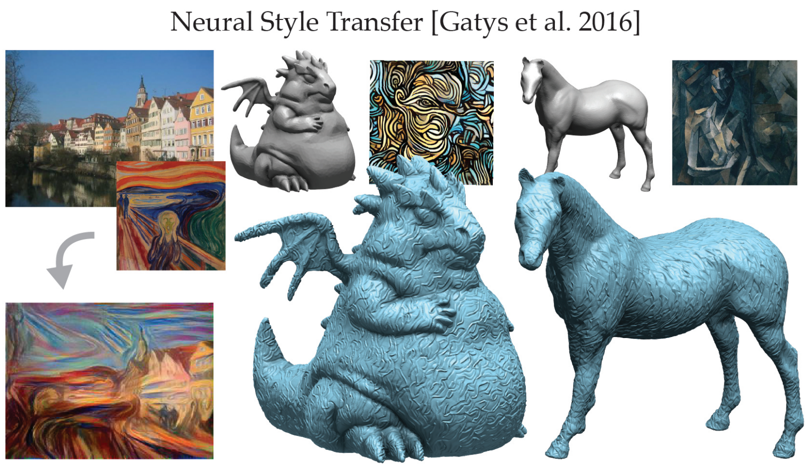


Figure 2.14: We generalize the neural style transfer to 3D by minimizing the style energy of local renderings through manipulating vertex positions.

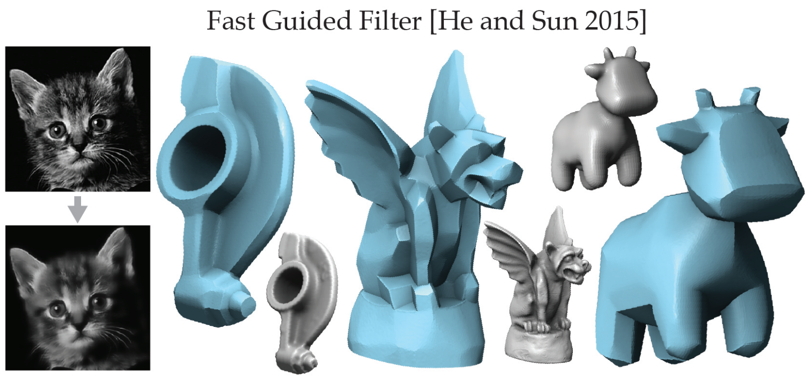


Figure 2.15: We generalize the fast guided filter to 3D and achieve edge-preserving smoothing effect.

so on. In Fig. 2.15 we apply the edge-aware smoothing guided filter with the acceleration proposed in [He and Sun, 2015]. We set the guidance image to be the input and the filter parameters to be $r = 4, \epsilon = 0.02$. By plugging this filter to the filter function f in Alg. 9, we can see that the guided filter smooths 3D shapes and preserves sharp features.

In addition to edge-preserving smoothing, we are interested in utilizing image filters to create different visual effects on geometry. A simple but stylistic choice is image *quantization*, an image compression technique compressing a range of color values to a single value and representing an image with only a small set of colors [Ozturk et al., 2014]. Again, by changing the filter f in Alg. 9, we can quantize 3D shapes with pre-define color set[4] (see Fig. 2.16). Note that these colors are encoded in the world coordinate, thus the shape quantization is orientation dependent and requires rendering normals in world coordinates, which is different from other filters that render normals in local coordinates.

[4]The pre-defined color set are the color of 20 face normals of a icosahedron

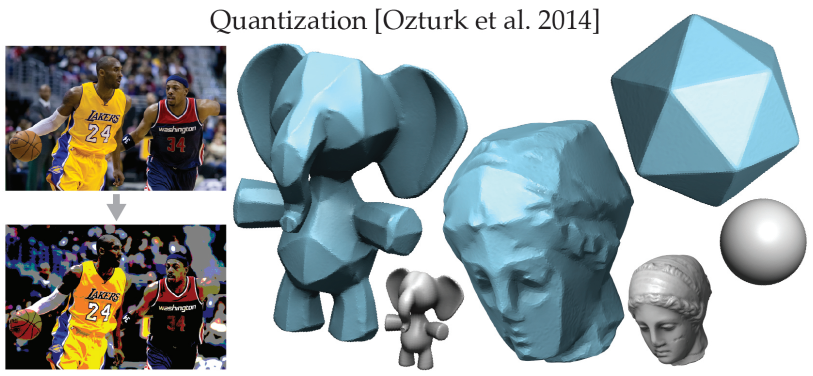


Figure 2.16: Image quantization is applied to geometries and make surface patches facing toward pre-defined color palettes.

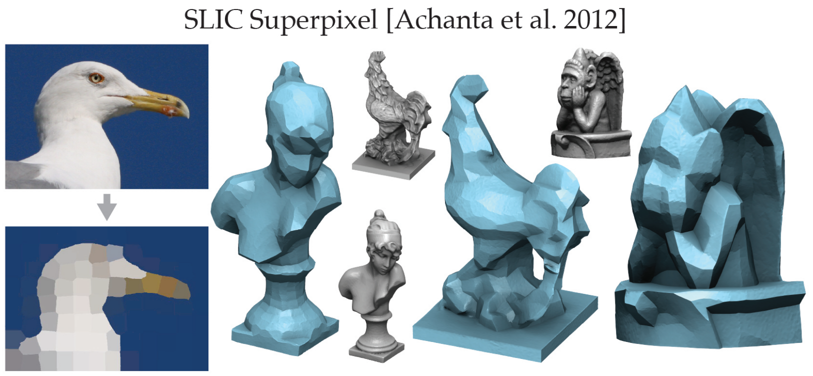


Figure 2.17: The SLIC superpixel method is applied to 3D objects, results in small surface patches appearing on the shape

Another pixel segmentation approach, but based on both color and spatial information, is the *superpixel*. In Fig. 2.17, we use Simple Linear Iterative Clustering (SLIC) [Achanta et al., 2012] which adapts k-means to segment pixels to create "super-faces" on shapes.

Last but not least, we consider a filter that minimizes the L_0 norm of image gradient [Xu et al., 2011]. L_0 norm has been a popular tool for image and signal processing for decades because it is a direct measure of signal sparsity. However, the L_0 norm can be difficult to optimize due to its discrete, combinatorial nature. Xu et al. [2011] present an iterative image optimization method to minimize L_0 gradient and generate edge-preserving, piecewise constant filtering effects. With Alg. 9 we can simply apply such iterative procedures to generalize the effect of L_0 norm to a 3D shape and make it piecewise planar which is the 3D analogue of piecewise constant in images (see Fig. 2.18).

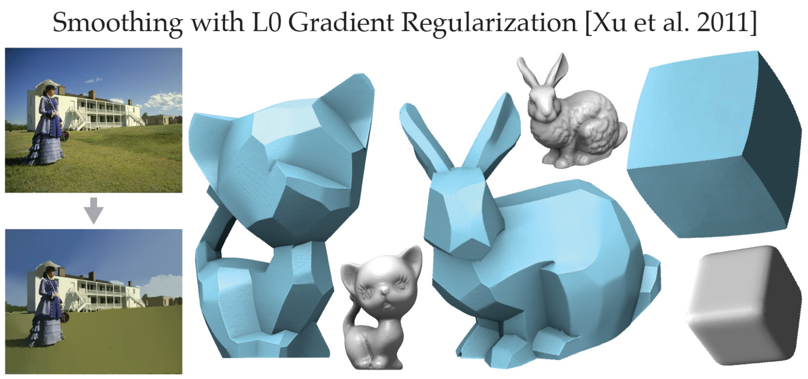
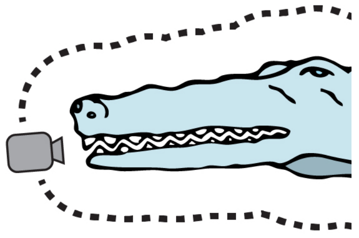


Figure 2.18: We minimize the L_0 norm of image gradients and encourage the output shapes (blue) to be piece-wise planar.

2.8 Conclusion

Paparazzi samples a precomputed offset surface for camera locations. This means heavily occluded or tight inner cavities of a surface will not receive edits (e.g., inside an alligator's mouth). It also means the shape is implicitly trapped inside its original offset surface *cage*. Removing this cage constraint and predicting the change of visibility would aid creating large shape deformations. For a stricter and more deterministic image energy, it would be important to align the cameras' orientation to encourage consistency across views in the overlapped regions. Meanwhile, we only present analytic derivatives for flat-shaded triangle meshes; similar derivatives could be derived for other representations such as subdivision surfaces or NURBS models. *Paparazzi*'s differentiation is orders of magnitude faster than previous work. In future work, we would like to further improve the performance of *Paparazzi* by exploiting the parallelism of the stochastic multi-view optimization and improving the collision-detection needed for dynamic meshing (currently, El Topo — used as a black-box — dominates our runtime, see Fig. 2.10).



At its core, *Paparazzi* is a differentiable renderer with a stochastic multiview gradient-descent procedure that can *back propagate* image changes to a 3D surface. *Paparazzi* imposes a 3D interpretation of a 2D filter, but could be an useful tool for studying other filters that have no straightforward 3D interpretation. Extending *Paparazzi* to operate with global illumination and textures as well as more sophisticated lighting models could be beneficial for applications that require realistic renderings, such as image classification. In our neural style transfer examples, we show only a small indication of the larger possibility for *Paparazzi*'s usefulness to transfer the success of image-based deep learning to 3D surface geometry. *Paparazzi* demonstrates the utility of rendering not just for visualization, but also as a method for *editing* of 3D shapes. It is exciting to consider other ways *Paparazzi* can influence and interact with the geometry processing pipeline.

2.9 Appendix

2.9.1 Neural Style Implementation Details

In this section, for the purpose of describing the parameters we used for the image neural style transfer [Gatys et al., 2016], we briefly summarize its key ingredients. Because we use this application to introduce surface details rather than change the large-scale geometric appearance of an input we have to change some of the lighting and camera parameters used in *Paparazzi*.

The goal of image style transfer is to, given a content image I_c and a reference style image I_s, generate an I_t that both looks similar to I_c in the style of I_s. This is performed by finding the image that minimizes an energy E_{NSE}, which is defined as a weighted sum of a content energy and style energy, which are in turn defined on the feature maps (i.e activations) in each layer of a convolutional neural network (CNN):

$$E_{\text{NSE}} = w \sum_{\ell} \alpha_{\ell} E^{\ell}_{\text{content}} + (1-w) \sum_{\ell} \beta_{\ell} E^{\ell}_{\text{style}},$$

where w controls the weighting between content and style energies, $\alpha_{\ell}, \beta_{\ell}$ correspond to content and style weights for layer ℓ. The feature maps of I_t, I_c, I_s at layer ℓ of the CNN are denoted by $\mathbf{T}^{\ell}, \mathbf{C}^{\ell}, \mathbf{S}^{\ell} \in \mathbb{R}^{N_{\ell} \times M_{\ell}}$ respectively, where N_{ℓ} is the number of features in the layer ℓ and M_{ℓ} is the size of a feature image on the ℓ^{th} layer.

$E^{\ell}_{\text{content}}$ is the squared difference between the feature maps of the target and content images at layer ℓ:

$$E^{\ell}_{\text{content}} = \frac{1}{2} \sum_{i=1}^{N_{\ell}} \sum_{j=1}^{M_{\ell}} (T^{\ell}_{ij} - C^{\ell}_{ij})^2.$$

E^{ℓ}_{style} is squared error between the features correlations expressed by the *Gram* matrices over features G of content and style images at layer ℓ:

$$E^{\ell}_{\text{style}} = \frac{1}{4 N_{\ell}^2 M_{\ell}^2} \sum_{i=1}^{N_{\ell}} \sum_{j=1}^{N_{\ell}} \left((G_t)^{\ell}_{ij} - (G_s)^{\ell}_{ij} \right)^2,$$

$G^{\ell} \in \mathbb{R}^{N_{\ell} \times N_{\ell}}$ for a feature map F^{ℓ} is explicitly written as

$$G^{\ell}_{ij} = \sum_{k=1}^{M_{\ell}} F^{\ell}_{ik} F^{\ell}_{jk}.$$

Both the style of strokes and colors contribute to the image style energy E_{NSE} but we omit the color style transfer because our focus is on the 3D geometry, instead of texture. In particular, we capture the style of strokes from a painting by perturbing vertex positions in order to create the shading changes required to decrease the style energy.

To eliminate the influence of the color component, we make both the style image and the rendered image gray-scale, we use one white directional light along the y-axis of the camera space to render the geometry. We also use offset $\sigma = 0.02$ and field of view of 0.1, OpenGL canonical view volume, to *zoom in* on the mesh in order to generate surface details rather than affect the large-scale geometric features. We set α_ℓ, β_1 to be 1 for layer *conv1_1, conv2_1, conv3_1, conv4_1, conv5_1* and 0 for the other layers in the VGG network [Simonyan and Zisserman, 2014]. We omit the content energy by setting $w = 0$. In our experiments, we can still transfer the style without losing the original appearance.

Chapter 3

Parametric Adversaries using an Analytically Differentiable Renderer

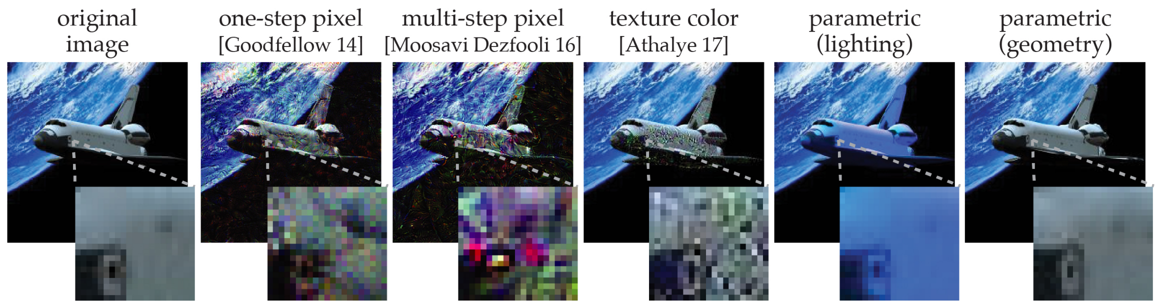

original image
one-step pixel [Goodfellow 14]
multi-step pixel [Moosavi Dezfooli 16]
texture color [Athalye 17]
parametric (lighting)
parametric (geometry)

Figure 3.1: Traditional pixel-based adversarial attacks yield unrealistic images under a larger perturbation (L^∞-norm ≈ 0.82), however our parametric lighting and geometry perturbations output more realistic images under the same norm (more results in App. 3.7.1).

Many machine learning image classifiers are vulnerable to adversarial attacks, inputs with perturbations designed to intentionally trigger misclassification. Current adversarial methods directly alter pixel colors and evaluate against *pixel norm-balls*: pixel perturbations smaller than a specified magnitude, according to a measurement norm. This evaluation, however, has limited practical utility since perturbations in the pixel space do not correspond to underlying real-world phenomena of image formation that lead to them and has no security motivation attached. Pixels in natural images are measurements of light that has interacted with the geometry of a physical scene. As such, we propose a novel evaluation measure, *parametric norm-balls*, by directly perturbing physical parameters that underly image formation. One enabling contribution we present is a physically-based differentiable renderer that allows us to propagate pixel gradients to the parametric space of lighting and geometry. Our approach enables physically-based adversarial attacks, and our differentiable renderer leverages models from the interactive rendering literature to balance the performance and accuracy trade-offs necessary for a memory-efficient and scalable adversarial data augmentation workflow.

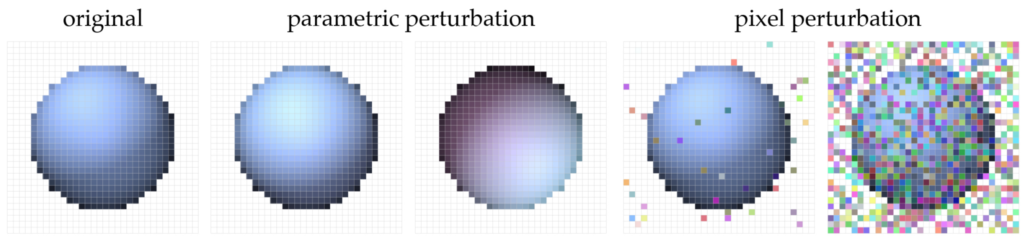


Figure 3.2: Parametrically-perturbed images remain natural, whereas pixel-perturbed ones do not.

3.1 Introduction

Research in adversarial examples continues to contribute to the development of robust (semi$\bar{/}$)supervised learning [Miyato et al., 2018], data augmentation [Goodfellow et al., 2015; Sun et al., 2018], and machine learning understanding [Kanbak et al., 2018]. One important caveat of the approach pursued by much of the literature in adversarial machine learning, as discussed recently [Goodfellow, 2018; Gilmer et al., 2018], is the reliance on overly simplified attack metrics: namely, the use of pixel value differences between an adversary and an input image, also referred to as the pixel *norm-balls*.

The pixel norm-balls game considers pixel perturbations of norm-constrained magnitude [Goodfellow et al., 2015], and is used to develop adversarial attackers, defenders and training strategies. The pixel norm-ball game is attractive from a research perspective due to its simplicity and well-posedness: no knowledge of image formation is required and any arbitrary pixel perturbation remains eligible (so long as it is "small", in the perceptual sense). Although the pixel norm-ball is useful for research purposes, it only captures limited real-world security scenarios.

Despite the ability to devise effective adversarial methods through the direct employment of optimizations using the pixel norm-balls measure, the pixel manipulations they promote are divorced from the types of variations present in the real world, limiting their usefulness "in the wild". Moreover, this methodology leads to defenders that are only effective when defending against unrealistic images/attacks, not generalizing outside of the space constrained by pixel norm-balls. In order to consider conditions that enable adversarial attacks in the real world, we advocate for a new measurement norm that is rooted in the physical processes that underly realistic image synthesis, moving away from overly simplified metrics, e.g., pixel norm-balls.

Our proposed solution – *parametric norm-balls* – rely on perturbations of physical parameters of a synthetic image formation model, instead of pixel color perturbations (Fig. 3.2). To achieve this, we use a physically-based differentiable renderer which allows us to perturb the underlying parameters of the image formation process. Since these parameters *indirectly* control pixel colors, perturbations in this parametric space *implicitly* span the space of natural images. We will demonstrate two advantages that fall from considering perturbations in this parametric space: (1) they enable adversarial approaches that more readily apply to real-world applications, and (2) they permit the use of much more significant perturbations (compared to pixel norms), without invalidating the realism of the resulting image (Fig. 3.1). We validate that parametric norm-balls game playing is critical for a variety of important adversarial tasks, such as building defenders robust to perturbations that can occur naturally in the real world.

We perform perturbations in the underlying image formation parameter space using a novel physically-based differentiable renderer. Our renderer analytically computes the derivatives of pixel color with respect to these physical parameters, allowing us to extend traditional pixel norm-balls to physically-valid parametric norm-balls. Notably, we demonstrate perturbations on an environment's *lighting* and on the shape of the 3D *geometry* it shades. Our differentiable renderer achieves state-of-the-art performance in speed and scalability (Sec. 3.3) and is fast enough for *rendered adversarial data augmentation* (Sec. 3.5): training augmented with adversarial images generated with a renderer.

Existing differentiable renders are slow and do not scale to the volume of high-quality, high-resolutions images needed to make adversarial data augmentation tractable (Sec. 3.2). Given our analytically-differentiable renderer (Sec. 3.3), we are able to demonstrate the efficacy of parametric space perturbations for generating adversarial examples. These adversaries are based on a substantially different phenomenology than their pixel norm-balls counterparts (Sec. 3.4). Ours is among the first steps towards the deployment of rendered adversarial data augmentation in real-world applications: we train a classifier with computer-generated adversarial images, evaluating the performance of the training against real photographs (i.e., captured using cameras; Sec. 3.5). We test on real photos to show the parametric adversarial data augmentation increases the classifier's robustness to "deformations" in the real world. Our evaluation differs from the majority of existing literature which evaluates against computer-generated adversarial images, since our parametric space perturbation is no-longer a wholly idealized representation of the image formation model but, instead, modeled against physically-based image generation.

3.2 Related Work

Our work is built upon the fact that *simulated* or *rendered* images can participate in computer vision and machine learning on real-world tasks. Many previous works use rendered (simulated) data to train deep networks, and those networks can be deployed to real-world or even outperform the state-of-the-art networks trained on real photos [Movshovitz-Attias et al., 2016; Chen et al., 2016; Varol et al., 2017; Su et al., 2015; Johnson-Roberson et al., 2017; Veeravasarapu et al., 2017b; Sadeghi and Levine, 2016; James and Johns, 2016]. For instance, Veeravasarapu et al. [2017a] show that training with 10% real-world data and 90% simulation data can reach the level of training with full real data. Tremblay et al. [2018]

even demonstrate that the network trained on synthetic data yields a better performance than using real data alone. As rendering can cheaply provide a theoretically infinite supply of annotated input data, it can generate data which is orders of magnitude larger than existing datasets. This emerging trend of training on synthetic data provides an exciting direction for future machine learning development. Our work complements these works. We demonstrate the utility of rendering can be used to study the potential danger lurking in misclassification due to subtle changes in geometry and lighting. This provides a future direction of leveraging synthetic data generation pipelines to perform *physically based adversarial training on synthetic data*.

Adversarial Examples Szegedy et al. [2014] expose the vulnerability of modern deep neural nets using purposefully-manipulated images with human-imperceptible noise that can trigger misclassification. Goodfellow et al. [2015] introduce a fast method to harness adversarial examples, leading to the idea of pixel norm-balls for evaluating adversarial attackers/defenders. Since then, many significant developments in adversarial techniques have been proposed [Akhtar and Mian, 2018; Szegedy et al., 2014; Rozsa et al., 2016; Kurakin et al., 2017; Moosavi Dezfooli et al., 2016; Dong et al., 2018; Papernot et al., 2017; Moosavi-Dezfooli et al., 2017; Chen et al., 2017b; Su et al., 2017]. Our work extends this progression in constructing adversarial examples, a problem that lies at the foundation of adversarial machine learning. Kurakin et al. [2016] study the transferability of attacks to the physical world by printing then photographing adversarial *images*. Athalye et al. [2017] and Eykholt et al. [2018] propose extensions to non-planar (yet, still fixed) geometry and multiple viewing angles. These works still rely fundamentally on the direct pixel or texture manipulation on physical objects. Since these methods assume independence between pixels in the image or texture space they remain variants of pixel norm-balls. This leads to unrealistic attack images that cannot model real-world scenarios [Goodfellow, 2018; Hendrycks and Dietterich, 2018; Gilmer et al., 2018]. Zeng et al. [2017] generate adversarial examples by altering physical parameters using a rendering network [Liu et al., 2017a] trained to approximate the physics of realistic image formation. This data-driven approach leads to an image formation model biased towards the rendering style present in the training data. This method also relies on differentiation through the rendering network in order to compute adversaries, which requires high-quality training on a large amount of data. Even with perfect training, in their reported performance, it still requires 12 minutes on average to find new adversaries, we only take a few seconds Sec. 3.4.1. Our approach is based on a differentiable physically-based renderer that directly (and, so, more convincingly) models the image formation process, allowing us to alter physical parameters – like geometry and lighting – and compute derivatives (and adversarial examples) much more rapidly compared to the [Zeng et al., 2017]. We summarize the difference between our approach and the previous non-image adversarial attacks in Table 3.1.

Table 3.1: Previous non-pixel attacks fall short in either the parameter range they can take derivatives or the performance.

Methods	Perf.	Color	Normal	Material	**Light**	**Geo.**
Athalye 17	✓	✓				
Zeng 17			✓	✓	✓	
Ours	✓	✓	✓		✓	✓

Table 3.2: Previous differentiable renderers fall short in one way or another among Performance, Bias, or Accuracy.

Methods		Performance.	Unbias	Accuracy
NN proxy	(Liu 17)	✓		
Approx.	(Kato 18)	✓	✓	
Autodiff	(Loper 14)		✓	✓
Analytical	(Ours)	✓	✓	✓

Differentiable Renderer Applying parametric norm-balls requires that we differentiate the image formation model with respect to the physical parameters of the image formation model. Modern realistic computer graphics models do not expose facilities to directly accommodate the computation of derivatives or automatic differentiation of pixel colors with respect to geometry and lighting variables. A physically-based differentiable renderer is fundamental to computing derivative of pixel colors with respect to scene parameters and can benefit machine learning in several ways, including promoting the development of novel network architectures [Liu et al., 2017a], in computing adversarial examples [Athalye et al., 2017; Zeng et al., 2017], and in generalizing neural style transfer to a 3D context [Kato et al., 2018; Liu et al., 2018]. Recently, various techniques have been proposed to obtain these derivatives: Wu et al. [2017]; Liu et al. [2017a]; Eslami et al. [2016] use neural networks to *learn* the image formation process provided a large amount of input/output pairs. This introduces unnecessary bias in favor of the training data distribution, leading to inaccurate derivatives due to imperfect learning. Kato et al. [2018] propose a differentiable renderer based on a simplified image formation model and an underlying linear approximation. Their approach requires no training and is unbiased to the training data, but their approximation of the image formation and the derivatives introduce more errors. Loper and Black [2014]; Genova et al. [2018] use automatic differentiation to build fully differentiable renderers. These renderers, however, are expensive to evaluate, requiring orders of magnitude more computation and much larger memory footprints compared to our method.

Our novel differentiable renderer overcomes these limitations by efficiently computing *analytical* derivatives of a physically-based image formation model. The key idea is that the non-differentiable visibility change can be ignored when considering infinitesimal perturbations. We model image variations by changing geometry and realistic lighting conditions in an analytically differentiable manner, relying on an accurate model of diffuse image formation that extends spherical harmonics-based shading methods (App. 3.7.3). Our analytic derivatives are efficient to evaluate, have scalable memory utilization, are unbiased, and are accurate by construction (Table 3.2). Our renderer explicitly models the physics of the image formation processes, and so the images it generates are realistic enough to illicit correct classifications from networks trained on real-world photographs.

3.3 Adversarial Attacks in Parametric Spaces

Adversarial attacks based on pixel norm-balls typically generate adversarial examples by defining a cost function over the space of images $\mathcal{C} : I \rightarrow \mathbb{R}$ that enforces some intuition of what failure should look like, typically using variants of gradient descent where the gradient $\partial\mathcal{C}/\partial I$ is accessible by differentiating through networks [Szegedy et al., 2014; Goodfellow et al., 2015; Rozsa et al., 2016; Kurakin et al., 2017; Moosavi Dezfooli et al., 2016; Dong et al., 2018].

The choices for $\mathcal{C}$ include increasing the cross-entropy loss of the correct class [Goodfellow et al., 2015], decreasing the cross-entropy loss of the least-likely class [Kurakin et al., 2017], using a combination of cross-entropies [Moosavi Dezfooli et al., 2016], and more [Szegedy et al., 2014; Rozsa et al., 2016; Dong et al., 2018; Tramèr et al., 2017]. We use a combination of cross-entropies to provide flexibility for choosing untargeted and targeted attacks by specifying a different set of labels:

$$\mathcal{C}\big(I(U,V)\big) = -\text{CrossEntropy}\big(f(I(U,V)), L_d\big) + \text{CrossEntropy}\big(f(I(U,V)), L_i\big), \tag{3.1}$$

where I is the image, $f(I)$ is the output of the classifier, L_d, L_i are labels for which a user wants to *decrease* and *increase* the predicted confidences respectively. In our experiments, L_d is the correct class and L_i is either ignored or chosen according to user preference. Our adversarial attacks in the parametric space consider an image $I(U,V)$ which is a function of physical parameters of the image formation model, including the lighting U and the geometry V. Adversarial examples constructed by perturbing physical parameters can then be computed via the chain rule

$$\frac{\partial \mathcal{C}}{\partial U} = \frac{\partial \mathcal{C}}{\partial I}\frac{\partial I}{\partial U} \qquad\qquad \frac{\partial \mathcal{C}}{\partial V} = \frac{\partial \mathcal{C}}{\partial I}\frac{\partial U}{\partial V}, \tag{3.2}$$

where $\partial I/\partial U, \partial I/\partial V$ are derivatives with respect to the physical parameters and we evaluate using our physically based differentiable renderer. In our experiments, we use gradient descent for finding parametric adversarial examples where the gradient is the direction of $\partial I/\partial U, \partial I/\partial V$.

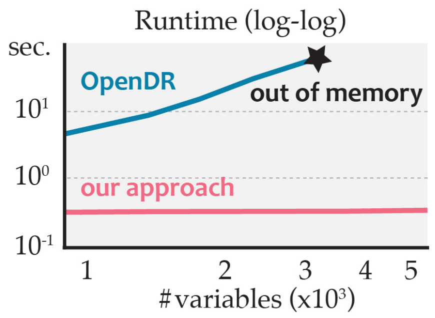


Figure 3.3: Our differentiable renderer based on analytical derivatives is faster and more scalable than the previous method.

3.3.1 Physically Based Differentiable Renderer

Rendering is the process of generating a 2D image from a 3D scene by simulating the physics of light. Light sources in the scene emit photons that then interact with objects in the scene. At each interaction, photons are either reflected, transmitted or absorbed, changing trajectory and repeating until arriving at a sensor such as a camera. A physically based renderer models the interactions mathematically [Pharr et al., 2016], and our task is to analytically differentiate the physical process.

We develop our differentiable renderer with common assumptions in real-time rendering [Akenine-Möller et al., 2008] – diffuse material, local illumination, and distant light sources. Our diffuse material assumption considers materials which reflect lights uniformly for all directions, equivalent to considering non-specular objects. We assume that variations in the material (texture) are piece-wise constant with respect to our triangle mesh discretization. The local illumination assumption only considers lights that bounce directly from the light source to the camera. Lastly, we assume light sources are far away from the scene, allowing us to represent lighting with one spherical function. For a more detailed rationale of our assumptions, we refer readers to App. 3.7.2).

These assumptions simplify the complicated integral required for rendering [Kajiya, 1986] and allow us to represent lighting in terms of *spherical harmonics*, an orthonormal basis for spherical functions analogous to Fourier transformation. Thus, we can *analytically* differentiate the rendering equation to acquire derivatives with respect to lighting, geometry, and texture (derivations found in App. 3.7.3).

Using analytical derivatives avoids pitfalls of previous differentiable renderers (see Sec. 3.2) and make our differentiable renderer orders of magnitude faster than the previous fully differentiable renderer OpenDR [Loper and Black, 2014] (see Fig. 3.3). Our approach is scalable to handle problems with more than 100,000 variables, while OpenDR runs out of memory for problems with more than 3,500 variables.

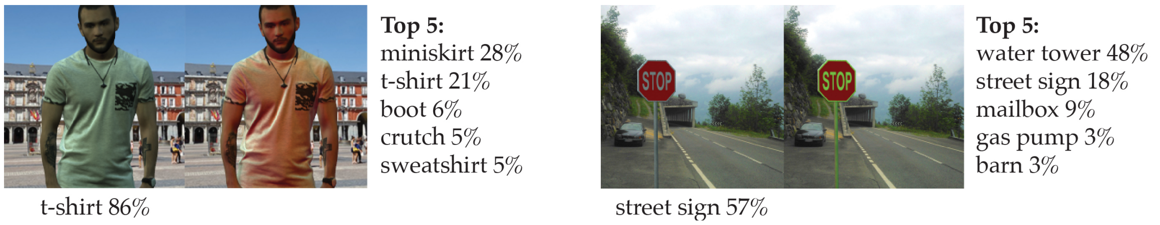


Figure 3.4: By changing the lighting, we fool the classifier into seeing miniskirt and water tower, demonstrating the existence of adversarial lighting.

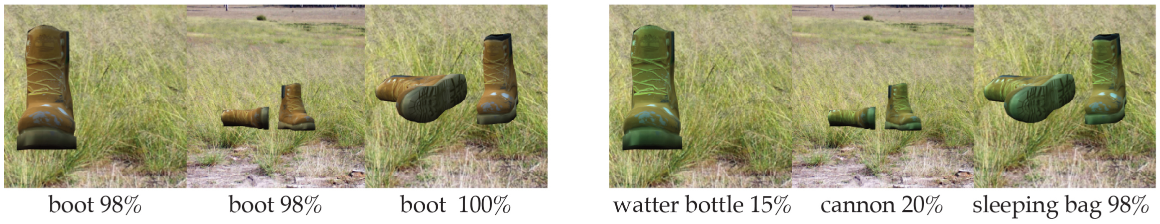


Figure 3.5: We construct a single lighting condition that can simultaneously fool the classifier viewing from different angles.

3.3.2 Adversarial Lighting and Geometry

Adversarial lighting denotes adversarial examples generated by changing the spherical harmonics lighting coefficients U [Green, 2003]. As our differentiable renderer allows us to compute $\partial I/\partial U$ analytically (derivation is provided in App. 3.7.3), we can simply apply the chain rule:

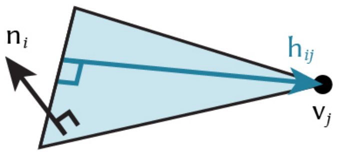
$$U \leftarrow U - \gamma \frac{\partial \mathcal{C}}{\partial I} \frac{\partial I}{\partial U}, \tag{3.3}$$

where $\partial \mathcal{C}/\partial I$ is the derivative of the cost function with respect to pixel colors and can be obtained by differentiating through the network. Spherical harmonics act as an implicit constraint to prevent unrealistic lighting because natural lighting environments everyday life are dominated by low-frequency signals. For instance, rendering of diffuse materials can be approximated with only 1% pixel intensity error by the first 2 orders of spherical harmonics [Ramamoorthi and Hanrahan, 2001a]. As computers can only represent a finite number of coefficients, using spherical harmonics for lighting implicitly filters out high-frequency, unrealistic lightings. Thus, perturbing the parametric space of spherical harmonics lighting generates more realistic images compared to image-pixel perturbations Fig. 3.1.

Adversarial geometry is an adversarial example computed by changes to the position of the shape's surface. The shape is encoded as a triangle mesh with $|V|$ vertices and $|F|$ faces, surface points are vertex positions $V \in \mathbb{R}^{|V|\times 3}$ which determine per-face normals $N \in \mathbb{R}^{|F|\times 3}$ which in turn determine the shading of the surface. We can compute adversarial shapes by applying the chain rule:

$$V \leftarrow V - \gamma \frac{\partial \mathcal{C}}{\partial I} \frac{\partial I}{\partial N} \frac{\partial N}{\partial V}, \tag{3.4}$$

where $\partial I / \partial N$ is computed via a derivation in App. 3.7.5. Each triangle only has one normal on its face, making $\partial N / \partial V$ computable analytically. In particular, the 3×3 Jacobian of a unit face normal vector $\mathsf{n}_i \in \mathbb{R}^3$ of the jth face of the triangle mesh V with respect to one of its corner vertices $\mathsf{v}_j \in \mathbb{R}^3$ is



$$\frac{\partial \mathsf{n}_i}{\partial \mathsf{v}_j} = \frac{\mathsf{h}_{ij} \mathsf{n}_i^\top}{\|\mathsf{h}_{ij}\|^2},$$

where $\mathsf{h}_{ij} \in \mathbb{R}^3$ is the height vector: the shortest vector to the corner v_j from the opposite edge.

3.4 Results

We have described how to compute adversarial examples by parametric perturbations, including lighting and geometry. In this section, we show that adversarial examples exist in the parametric spaces, then we analyze the characteristics of those adversaries and parametric norm-balls.

We use spherical harmonics from degrees 0 to 6 to represent environment lighting. This leads to 49×3 spherical harmonics coefficients initialized with real-world lighting conditions presented in [Ramamoorthi and Hanrahan, 2001a]. Camera parameters and the background images are empirically chosen to have correct initial classifications. In Fig. 3.4 we show that single-view adversarial lighting attack can fool the classifier (pre-trained ResNet-101 on ImageNet [He et al., 2016]). Fig. 3.5 shows multi-view adversarial lighting, which optimizes the summation of the cost functions for each view, thus the gradient is computed as the summation over all camera views:

$$U \leftarrow U - \sum_{i \in \text{cameras}} \gamma \frac{\partial \mathcal{C}}{\partial I_i} \frac{\partial I_i}{\partial U}. \tag{3.5}$$

If one is interested in a more specific subspace, such as outdoor lighting conditions governed by sunlight and weather, our adversarial lighting can adapt to it. In Fig. 3.7, we compute adversarial lights over the space of skylights by applying one more chain rule to the *Preetham skylight* parameters [Preetham et al., 1999; Habel et al., 2008]. Details about taking these derivatives are provided in App. 3.7.4. Although adversarial skylights exist, the low degrees of freedom (only three parameters) makes it more difficult to find adversaries.

In Fig. 3.8 and Fig. 3.9 we show the existence of adversarial geometry in both single-

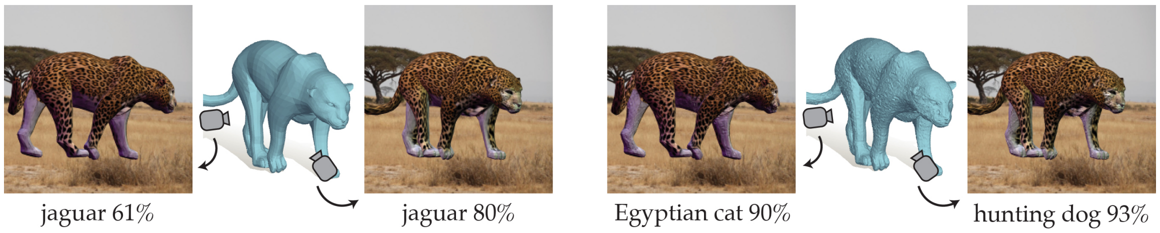


Figure 3.6: By specifying different target labels, we can create an optical illusion: a jaguar is classified as cat and dog from two different views after geometry perturbations.

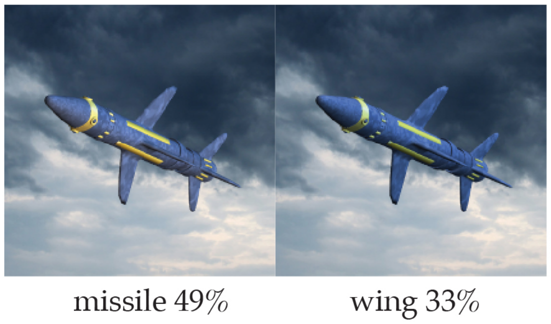


Figure 3.7: Even if we further constrain to a lighting subspace, skylight, we can still find adversaries.

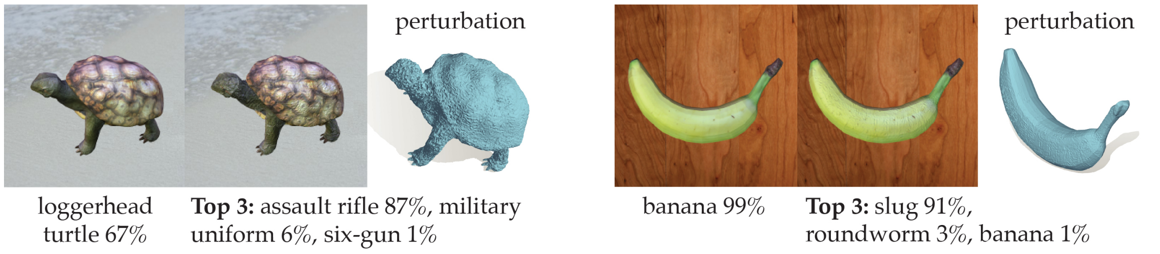

perturbation
loggerhead turtle 67%
Top 3: assault rifle 87%, military uniform 6%, six-gun 1%
perturbation
banana 99%
Top 3: slug 91%, roundworm 3%, banana 1%

Figure 3.8: Perturbing points on 3D shapes fools the classifier into seeing rifle/slug.

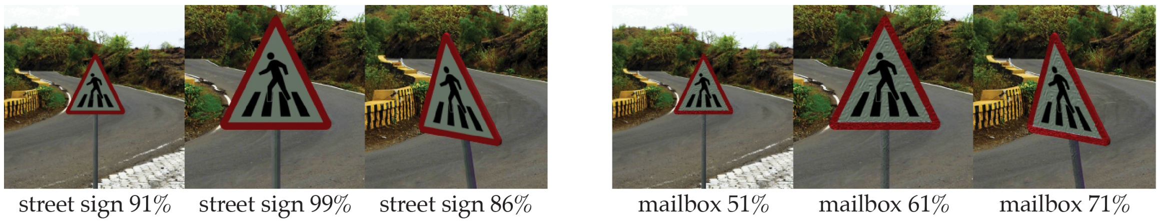


Figure 3.9: We construct a single adversarial geometry that fools the classifier seeing a mailbox from different angles.

Table 3.3: We evaluate ResNet adversaries on unseen models and show that parametric adversarial examples also share across models. The table shows the success rate of attacks (%).

	Alex	VGG	Squeeze	Dense
Lighting	81.2%	65.0%	78.6%	43.5%
Geometry	70.3%	58.9%	71.1%	40.1%

Table 3.4: We compute parametric adversaries using a subset of views (#Views) and evaluate the success rates (%) of attacks on unseen views.

#Views	0	1	5
Lighting	0.0%	29.4%	64.2%
Geometry	0.0%	0.6%	3.6%

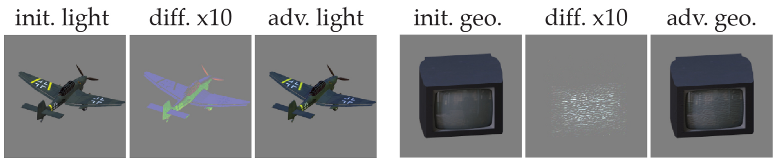
view and multi-view cases. Note that we upsample meshes to have >10K vertices as a preprocessing step to increase the degrees of freedom available for perturbations. Multi-view adversarial geometry enables us to perturb the same 3D shape from different viewing directions, which enables us to construct a *deep optical illusion*: The same 3D shape are classified differently from different angles. To create the optical illusion in Fig. 3.6, we only need to specify the L_i in Eq. (3.1) to be a dog and a cat for two different views.

3.4.1 Properties of Parametric Norm-Balls and Adversaries

To further understand parametric adversaries, we analyze how do parametric adversarial examples generalize to black-box models. In Table 3.3, we test 5,000 ResNet parametric adversaries on unseen networks including AlexNet [Krizhevsky et al., 2012], DenseNet [Huang et al., 2017], SqueezeNet [Iandola et al., 2016], and VGG [Simonyan and Zisserman, 2014]. Our result shows that parametric adversarial examples also trigger misclassifications across different classification models.

In addition to different models, we evaluate parametric adversaries on black-box viewing directions. This evaluation mimics the real-world scenario that a self-driving car would "see" a stop sign from different angles while driving. In Table 3.4, we randomly sample 500 correctly classified views for a given shape and perform adversarial lighting and geometry algorithms only on a subset of views, then evaluate the resulting adversarial lights/shapes on all the views. The results show that adversarial lights are more likely to fool unseen views. In contrast, adversarial shapes are not.

Switching from pixel norm-balls to parametric norm-balls only requires to change the norm-constraint from the pixel color space to the parametric space. For instance, we can perform a quantitative comparison between parametric adversarial and random perturbations in Fig. 3.10. We use L^∞*-norm* $= 0.1$ to constraint the perturbed magnitude of each lighting coefficient, and L^∞*-norm* $= 0.002$ to constrain the maximum displacement of surface points along each axis. The results show how many parametric adversaries can fool the classifier out of 10,000 adversarial lights and shapes respectively. Not only do the parametric norm-balls show the effectiveness of adversarial perturbation, evaluating robustness using parametric norm-balls has real-world implications.



	Adversarial	Random
Lighting	**61.0%**	9.1%
Geometry	**95.3%**	17.5%

Figure 3.10: A quantitative comparison using parametric norm-balls shows the fact that adversarial lighting/geometry perturbations have a higher success rate (%) in fooling classifiers comparing to random perturbations in the parametric spaces.

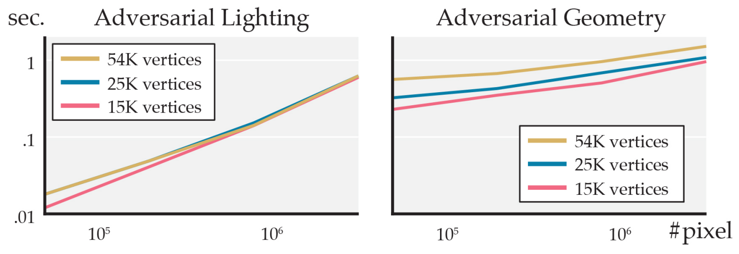


Figure 3.11: We reports the runtime of our method for adversarial lighting and geometry.

Runtime The inset presents our runtime per iteration for computing derivatives. An adversary normally requires less than 10 iterations , thus taking a few seconds. We evaluate our CPU Python implementation and the OpenGL rendering, on an Intel Xeon 3.5GHz CPU with 64GB of RAM and an NVIDIA GeForce GTX 1080. Our runtime depends on the number of pixels requiring derivatives.

3.5 Rendered Adversarial Data Augmentation Against Photos

We inject adversarial examples, generated using our differentiable renderer, into the training process of modern image classifiers. Our goal is to increase the robustness of these classifiers to real-world perturbations. Traditionally, adversarial training is evaluated against computer-generated adversarial images [Kurakin et al., 2017; Madry et al., 2018; Tramèr et al., 2017]. In contrast, our evaluation differs from the majority of the literature, as we evaluate performance against *real photos* (i.e., images captured using a camera), and not computer-generated images. This evaluation method is motivated by our goal of increasing a classifier's robustness to "perturbations" that occur in the real world and result from the physical processes underlying real-world image formation. We present preliminary steps towards this objective, resolving the lack of realism of pixel norm-balls and evaluating our augmented classifiers (i.e., those trained using our rendered adversaries) against real photographs.

Training We train the WideResNet (16 layers, 4 wide factor) [Zagoruyko and Komodakis, 2016] on CIFAR-100 [Krizhevsky and Hinton, 2009] augmented with adversarial lighting examples. We apply a common adversarial training method that adds a fixed number of adversarial examples each epoch [Goodfellow et al., 2015; Kurakin et al., 2017]. We refer readers to App. 3.7.6 for the training detail. In our experiments, we compare three training scenarios: (1) CIFAR-100, (2) CIFAR-100 + 100 images under random lighting, and (3) CIFAR-100 + 100 images under adversarial lighting. Comparing to the accuracy reported in [Zagoruyko and Komodakis, 2016], WideResNets trained on these three cases all have comparable performance ($\approx 77\%$) on the CIFAR-100 test set.

Testing We create a test set of real photos, captured in a laboratory setting with controlled lighting and camera parameters: we photographed oranges using a calibrated Prosilica GT 1920 camera under different lighting conditions, each generated by projecting different lighting patterns using an LG PH550 projector. This hardware lighting setup projects lighting patterns from a fixed solid angle of directions onto the scene objects. In the inset, we illustrate samples from the 500 real photographs of our dataset. We evaluate the robustness of our classifier models according to test accuracy. Of note, average prediction accuracies over five trained WideResNets on our test data under the three training cases are (1) 4.6%, (2) 40.4%, and (3) **65.8**%. This result supports the fact that training on rendered images can improve the networks' performance on real photographs. Our preliminary experiments motivate the potential of relying on rendered adversarial training to increase the robustness to visual phenomena present in the real-world inputs.

3.6 Conclusion

Using parametric norm-balls to remove the lack of realism of pixel norm-balls is only the first step to bring adversarial machine learning to real-world. More evaluations beyond the lab experimental data could uncover the potential of the rendered adversarial data augmentation. Coupling the differentiable renderer with methods for reconstructing 3D scenes, such as [Veeravasarapu et al., 2017b; Tremblay et al., 2018], has the potential to develop a complete pipeline for rendered adversarial training. We can take a small set of real images, constructing 3D virtual scenes which have real image statistics, using our approach to manipulate the predicted parameters to construct the parametric adversarial examples, then perform rendered adversarial training. This direction has the potential to produce limitless simulated adversarial data augmentation for real-world tasks.

Our differentiable renderer models the change of realistic environment lighting and geometry. Incorporating real-time rendering techniques from the graphics community could further improve the quality of rendering. Removing the locally constant texture assumption could improve our results. Extending the derivative computation to materials could enable "adversarial materials". Incorporating derivatives of the visibility change and prop-

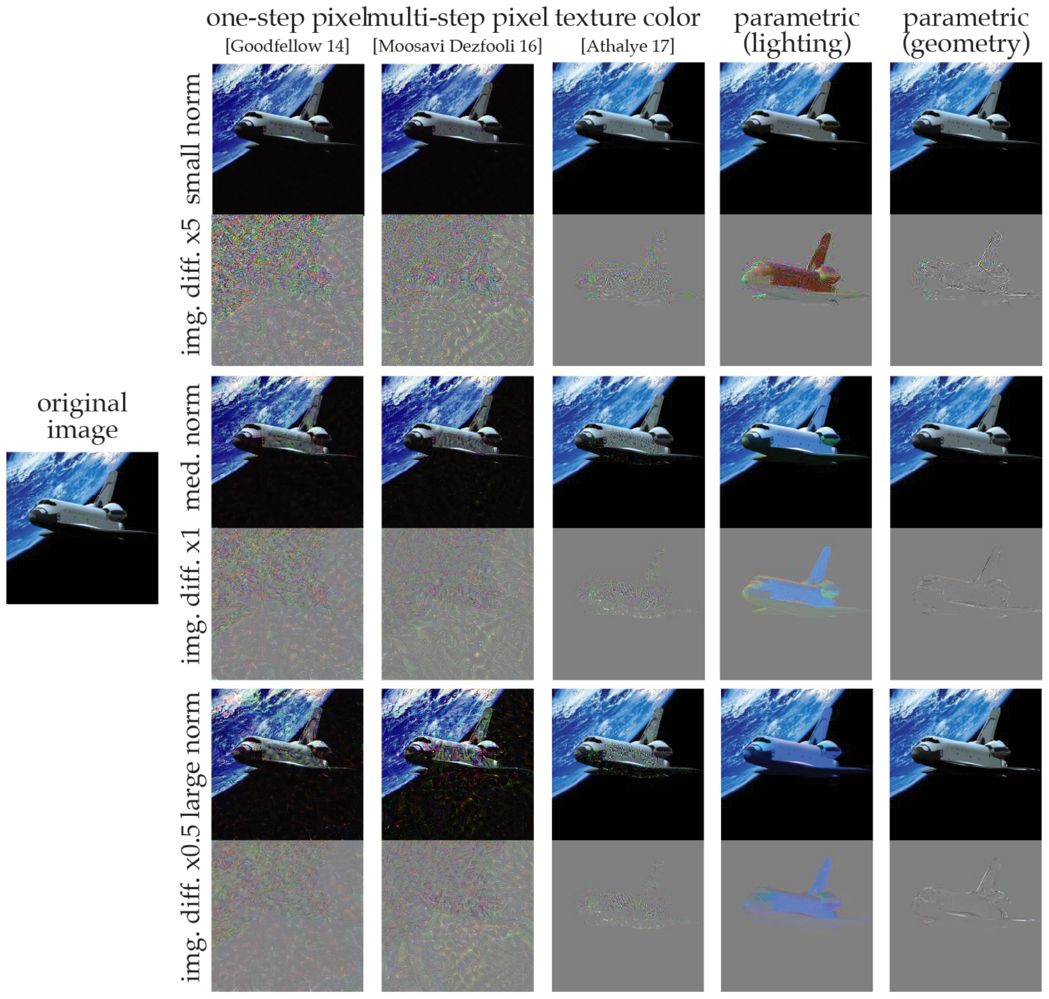


Figure 3.12: We compare our parametric perturbations (the first two columns) with pixel/color perturbations under the same L^{∞} pixel norm (small: 0.12, medium: 0.53, large: 0.82). As changing physical parameters corresponds to real-world phenomena, our parametric perturbation are more realistic.

agating gradient information to shape skeleton could also create "adversarial poses". These extensions offer a set of tools for modeling real security scenarios. For instance, we can train a self-driving car classifier that can robustly recognize pedestrians under different poses, lightings, and cloth deformations.

3.7 Appendix

3.7.1 Comparison Between Perturbation Spaces

We extend our comparisons against pixel norm-balls methods (Fig. 3.1) by visualizing the results and the generated perturbations (Fig. 3.12). We hope this figure elucidates that our parametric perturbation are more realistic at several scales of perturbations.

3.7.2 Physically Based Rendering

Physically based rendering (PBR) seeks to model the flow of light, typically the assumption that there exists a collection of light sources that generate light; a camera that receives this light; and a scene that modulates the flow light between the light sources and camera [Pharr et al., 2016]. What follows is a brief discussion of the general task of rendering an image from a scene description and the approximations we take in order to make our renderer efficient yet differentiable.

Computer graphics has dedicated decades of effort into developing methods and technologies to enable PBR to synthesize of photorealistic images under a large gamut of performance requirements. Much of this work is focused around taking approximations of the cherished Rendering equation [Kajiya, 1986], which describes the propagation of light through a point in space. If we let u_o be the output radiance, p be the point in space, ω_o be the output direction, u_e be the emitted radiance, u_i be incoming radiance, ω_i be the incoming angle, f_r be the way light be reflected off the material at that given point in space we have:

$$u_o(p,\omega_o) = u_e(p,\omega_o) + \int_{S^2} f_r(p,\omega_i,\omega_o)u_i(p,\omega_i)(\omega_i \cdot \mathbf{n})d\omega_i.$$

From now on we will ignore the emission term u_e as it is not pertinent to our discussion. Furthermore, because the speed of light is substantially faster than the exposure time of our eyes, what we perceive is not the propagation of light at an instant, but the steady state solution to the rendering equation evaluated at every point in space. Explicitly computing this steady state is intractable for our applications and will mainly serve as a reference for a plethora of assumptions and simplifications we will make for the sake of tractability. Many of these methods focus on ignoring light with nominal effects on the final rendered image vis a vis assumptions on the way light travels. For instance, light is usually assumed to have nominal interaction with air, which is described as the assumption that the space between objects is a vacuum, which constrains the interactions of light to the objects in a scene. Another common assumption is that light does not penetrate objects, which makes it difficult to render objects like milk and human skin[1]. This constrains the complexity of light propagation to the behavior of light bouncing off of object surfaces.

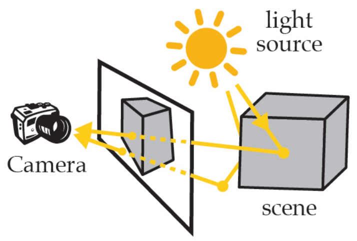


Figure 3.13: PBR models the physics of light that is emitted from the light source, interacts with the scene, and then arrive a camera.

Local Illumination

It is common to see assumptions that limits at the number of times light is allowed to bounce.In our case we chose to assume that the steady state is sufficiently approximated by

[1]this is why simple renderers make these sorts of objects look like plastic

an extremely low number of iterations: one. This means that it seems sufficient to model the lighting of a point in space by the light arriving at it directly from the light sources. Working with such a strong simplification does, of course, lead to a few artifacts. For instance, light occluded by other objects is ignored so shadows disappear and auxiliary techniques are usually employed to evaluate shadows [Williams, 1978; Miller, 1994].

When this assumption is coupled with a camera we approach what is used in standard *rasterization* systems such as OpenGL [Shreiner and Group, 2009], which is what we use. These systems compute the illumination of a single pixel by determining the fragment of an object visible through that pixel and only computing the light that traverses directly from the light sources, through that fragment, to that pixel. The lighting of a fragment is therefore determined by a point and the surface normal at that point, so we write the fragment's radiance as $R(p, \mathbf{n}, \omega_o) = u_o(p, \omega_o)$:

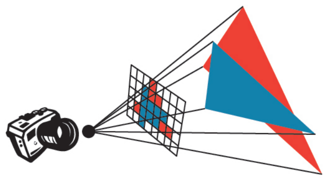


Figure 3.14: Rasterization converts a 3D scene into pixels.

$$R(p, \mathbf{n}, \omega_o) = \int_{S^2} f_r(p, \omega_i, \omega_o) u_i(p, \omega_i)(\omega_i \cdot \mathbf{n}) d\omega_i. \tag{3.6}$$

Lambertian Material

Each point on an object has a model approximating the transfer of incoming light to a given output direction f_r, which is usually called the material. On a single object the material parameters may vary quite a bit and the correspondence between points and material parameters is usually called the *texture map* which forms the *texture* of an object. There exists a wide gamut of material models, from mirror materials that transport light from a single input direction to a single output direction, to materials that reflect light evenly in all directions, to materials liked brushed metal that reflect differently along different angles. For the sake of this document we only consider diffuse materials, also called *Lambertian* materials, where we assume that incoming light is reflected uniformly, i.e f_r is a constant function with respect to angle, which we denote $f_r(p, \omega_i, \omega_o) = \rho(p)$:

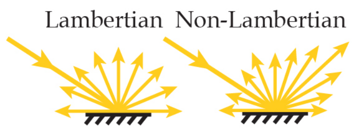


Figure 3.15: We consider the Lambertian material (left) where lights get reflected uniformly in every direction.

$$R(p, \mathbf{n}) = \rho(p) \int_{\Omega(\mathbf{n})} u(p, \omega)(\omega \cdot \mathbf{n}) d\omega. \tag{3.7}$$

This function ρ is usually called the *albedo*, which can be perceived as color on the surface for diffuse material, and we reduce our integration domain to the upper hemisphere $\Omega(\mathbf{n})$ in order to model light not bouncing through objects. Furthermore, since only the only ω and u are the incoming ones we can now suppress the "incoming" in our notation and just use ω and u respectively.

Environment Mapping

The illumination of static, distant objects such as the ground, the sky, or mountains do not change in any noticeable fashion when objects in a scene are moved around, so u can be written entirely in terms of ω, $u(p,\omega) = u(\omega)$. If their illumination forms a constant it seems prudent to pre-compute or cache their contributions to the illumination of a scene. This is what is usually called *environment mapping* and they fit in the rendering equation as a representation for the total lighting of a scene, i.e the total incoming radiance u_i. Because the environment is distant, it is common to also assume that the position of the object receiving light from an environment map does not matter so this simplifies u_i to be independent of position:

$$R(p,\mathbf{n}) = \rho(p) \int_{\Omega(\mathbf{n})} u(\omega)\,(\omega \cdot \mathbf{n})\, d\omega. \tag{3.8}$$

Spherical Harmonics

Despite all of our simplifications, the inner integral is still a fairly generic function over S^2. Many techniques for numerically integrating the rendering equation have emerged in the graphics community and we choose one which enables us to perform pre-computation and select a desired spectral accuracy: *spherical harmonics*. Spherical harmonics are a basis on S^2 so, given a spherical harmonics expansion of the integrand, the evaluation of the above integral can be reduced to a weighted product of coefficients. This particular basis is chosen because it acts as a sort of Fourier basis for functions on the sphere and so the bases are each associated with a frequency, which leads to a convenient multi-resolution structure. In fact, the rendering of diffuse objects under distant lighting can be 99% approximated by just the first few spherical harmonics bases [Ramamoorthi and Hanrahan, 2001a].

We will only need to note that the spherical harmonics bases Y_l^m are denoted with the subscript with l as the frequency and that there are $2l+1$ functions per frequency, denoted by superscripts m between $-l$ to l inclusively. For further details, please see App. 3.7.3.

If we approximate a function f in terms of spherical harmonics coefficients $f \approx \sum_{lm} f_{l,m} Y_l^m$ the integral can be precomputed as

$$\int_{S^2} f \approx \int_{S^2} \sum_{lm} f_{l,m} Y_l^m = \sum_{lm} f_{l,m} \int_{S^2} Y_l^m, \tag{3.9}$$

Thus we have defined a reduced rendering equation that can be efficiently evaluated using OpenGL while maintaining differentiability with respect to lighting and vertices. In the following appendix we will derive the derivatives necessary to implement our system.

3.7.3 Differentiable Renderer

Rendering computes an image of a 3D shape given lighting conditions and the prescribed material properties on the surface of the shape. Our differentiable renderer assumes Lambertian reflectance, distant light sources, local illumination, and piece-wise constant textures. We will discuss how to compute the derivatives used in the main text and give a detailed discussion about spherical harmonics and their advantages.

Spherical Harmonics

Spherical harmonics are usually defined in terms of the Legendre polynomials, which are a class of orthogonal polynomials defined by the recurrence relation

$$P_0 = 1 \tag{3.10}$$

$$P_1 = x \tag{3.11}$$

$$(l+1)P_{l+1}(x) = (2l+1)xP_l(x) - lP_{l-1}(x). \tag{3.12}$$

The *associated* Legendre polynomials are a generalization of the Legendre polynomials and can be fully defined by the relations

$$P_l^0 = P_l \tag{3.13}$$

$$(l-m+1)P_{l+1}^m(x) = (2l+1)xP_l^m(x) - (l+m)P_{l-1}^m(x) \tag{3.14}$$

$$2mxP_l^m(x) = -\sqrt{1-x^2}\left[P_l^{m+1}(x) + (l+m)(l-m+1)P_l^{m-1}(x)\right]. \tag{3.15}$$

Using the associated Legendre polynomials P_l^m we can define the spherical harmonics basis as

$$Y_l^m(\theta,\phi) = K_l^m \begin{cases} (-1)^m\sqrt{2}P_l^{-m}(\cos\theta)\sin(-m\phi) & m < 0 \\ (-1)^m\sqrt{2}P_l^m(\cos\theta)\cos(m\phi) & m > 0 \\ P_l^0(\cos\theta) & m = 0 \end{cases}. \tag{3.16}$$

$$\text{where } K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}}. \tag{3.17}$$

We will use the fact that the associated Legendre polynomials correspond to the spherical harmonics bases that are rotationally symmetric along the z axis ($m = 0$).

In order to incorporate spherical harmonics into Equation 3.8, we change the integral domain from the upper hemisphere $\Omega(\mathbf{n})$ back to S^2 via a max operation

$$R(p,\mathbf{n}) = \rho(p)\int_{\Omega(\mathbf{n})} u(\omega)(\omega\cdot\mathbf{n})d\omega \tag{3.18}$$

$$= \rho(p)\int_{\mathcal{S}^2} u(\omega)\max(\omega\cdot\mathbf{n},0)d\omega. \tag{3.19}$$

We see that the integral is comprised of two components: a lighting component $u(\omega)$ and a component that depends on the normal $\max(\omega\cdot\mathbf{n},0)$. The strategy is to pre-compute the two components by projecting onto spherical harmonics, and evaluating the integral via a dot product at runtime, as we will now derive.

Lighting in Spherical Harmonics

Approximating the lighting component $u(\omega)$ in Equation 3.19 using spherical harmonics Y_l^m up to band n can be written as

$$u(\omega) \approx \sum_{l=0}^{n} \sum_{m=-l}^{l} U_{l,m} Y_l^m(\omega),$$

where $U_{l,m} \in \mathbb{R}$ are coefficients. By using the orthogonality of spherical harmonics we can evaluate these coefficients as an integral between $u(\omega)$ and $Y_l^m(\omega)$

$$U_{l,m} = \langle u, Y_l^m \rangle_{\mathcal{S}^2} = \int_{\mathcal{S}^2} u(\omega) Y_l^m(\omega) d\omega,$$

which can be evaluated via quadrature.

Clamped Cosine in Spherical Harmonics

So far, we have projected the lighting term $u(\omega)$ onto the spherical harmonics basis. To complete evaluating Equation 3.19 we also need to approximate the second component $\max(\omega \cdot \mathbf{n}, 0)$ in spherical harmonics. This is the so-called the *clamped cosine* function.

$$g(\omega, \mathbf{n}) = \max(\omega \cdot \mathbf{n}, 0) = \sum_{l=0}^{n} \sum_{m=-l}^{l} G_{l,m}(\mathbf{n}) Y_l^m(\omega),$$

where $G_{l,m}(\mathbf{n}) \in \mathbb{R}$ can be computed by projecting $g(\omega, \mathbf{n})$ onto $Y_l^m(\omega)$

$$G_{l,m}(\mathbf{n}) = \int_{\mathcal{S}^2} \max(\omega \cdot \mathbf{n}, 0) Y_l^m(\omega) d\omega.$$

Unfortunately, this formulation turns out to be tricky to compute. Instead, the common practice is to analytically compute the coefficients for unit z direction $\tilde{G}_{l,m} = G_{l,m}(\mathbf{n}_z) = G_{l,m}([0, 0, 1]^\top)$ and evaluate the coefficients for different normals $G_{l,m}(\mathbf{n})$ by rotating $\tilde{G}_{l,m}$.

This rotation, $\tilde{G}_{l,m}$, can be computed analytically:

$$\begin{aligned}
\tilde{G}_{l,m} &= \int_{\mathcal{S}^2} \max(\omega \cdot \mathbf{n}_z, 0) Y_l^m(\omega) d\omega \\
&= \int_0^{2\pi} \int_0^{\pi} \max([\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta][0,0,1]^\top, 0) Y_l^m(\theta,\phi) \sin\theta d\theta d\phi \\
&= \int_0^{2\pi} \int_0^{\pi} \max(\cos\theta, 0) Y_l^m(\theta,\phi) \sin\theta d\theta d\phi \\
&= \int_0^{2\pi} \int_0^{\pi/2} \cos\theta \; Y_l^m(\theta,\phi) \sin\theta d\theta d\phi. \qquad (3.20)
\end{aligned}$$

In fact, because $\max(\omega \cdot \mathbf{n}_z, 0)$ is rotationally symmetric around the z-axis, its projection onto $Y_l^m(\omega)$ will have many zeros except the rotationally symmetric spherical harmonics Y_l^0. In other words, $\tilde{G}_{l,m}$ is non-zero only when $m = 0$. So we can simplify Equation 3.20 to

$$\tilde{G}_l = \tilde{G}_{l,0} = 2\pi \int_0^{\pi/2} \cos\theta \; Y_l^0(\theta) \sin\theta d\theta.$$

The evaluation of this integral can be found in Appendix A in [Basri and Jacobs, 2003]. We provide it here as well:

$$\tilde{G}_l = \begin{cases} \frac{\sqrt{\pi}}{2} & l = 0 \\ \sqrt{\frac{\pi}{3}} & l = 1 \\ (-1)^{\frac{l}{2}+1} \frac{(l-2)!\sqrt{(2l+1)\pi}}{2^l(\frac{l}{2}-1)!(\frac{l}{2}+1)!} & l \geq 2, \text{even} \\ 0 & l \geq 2, \text{odd} \end{cases}.$$

The spherical harmonics coefficients $G_{l,m}(\mathbf{n})$ of the clamped cosine function $g(\omega, \mathbf{n})$ can be computed by rotating $\tilde{G}_l$ [Sloan et al., 2005] using this formula

$$G_{l,m}(\mathbf{n}) = \sqrt{\frac{4\pi}{2l+1}} \tilde{G}_l \; Y_l^m(\mathbf{n}). \qquad (3.21)$$

So far we have projected the two terms in Equation 3.19 into the spherical harmonics basis. Orthogonality of spherical harmonics makes the evaluation of this integral straightforward:

$$\begin{aligned}
\int_{\mathcal{S}^2} u(\omega) \max(\omega \cdot \mathbf{n}, 0) d\omega &= \int_{\mathcal{S}^2} \left[\sum_{l,m} U_{l,m} Y_l^m(\omega) \right] \left[\sum_{j,k} G_{j,k}(\mathbf{n}) Y_j^k(\omega) \right] d\omega \\
&= \sum_{j,k,l,m} U_{l,m} G_{j,k}(\mathbf{n}) \delta_j^l \delta_k^m \qquad (3.22) \\
&= \sum_{l,m} U_{l,m} G_{l,m}(\mathbf{n}). \qquad (3.23)
\end{aligned}$$

This, in conjunction with Equation 3.21 allows us to derive the rendering equation using spherical harmonics lighting for Lambertian objects:

$$R(p,\mathbf{n}) = \rho(p) \sum_{l=0}^{n} \sum_{m=-l}^{l} U_{l,m} \sqrt{\frac{4\pi}{2l+1}} \tilde{G}_l \; Y_l^m(\mathbf{n}). \tag{3.24}$$

So far we have only considered the shading of a specific point p with surface normal $\mathbf{n}$. If we consider the rendered image I given a shape V, lighting U, and camera parameters η, the image I is the evaluation of the rendering equation R of each point in V visible through each pixel in the image. This pixel to point mapping is determined by η. Therefore, we can write I as

$$I(V,U,\eta) = \rho(V,\eta) \underbrace{\sum_{l=0}^{n} \sum_{m=-l}^{l} U_{l,m} \sqrt{\frac{4\pi}{2l+1}} \tilde{G}_l \; Y_l^m(N(V))}_{F(V,U)}, \tag{3.25}$$

where $N(V)$ is the surface normal. We exploit the notation and use $\rho(V,\eta)$ to represent the texture of V mapped to the image space through η.

Lighting and Texture Derivatives

For our applications we must differentiate Equation 3.25 with respect to lighting and material parameters. The derivative with respect to the lighting coefficients U can be obtained by

$$\frac{\partial I}{\partial U} = \frac{\partial \rho}{\partial U} F + \rho \frac{\partial F}{\partial U} \tag{3.26}$$

$$= 0 + \rho \sum_{l=0}^{n} \sum_{m=-l}^{l} \frac{\partial F}{\partial U_{l,m}}. \tag{3.27}$$

This is the Jacobian matrix that maps from spherical harmonics coefficients to pixels. The term ${\partial F}/{\partial U_{l,m}}$ can then be computed as

$$\frac{\partial F}{\partial U_{l,m}} = \sqrt{\frac{4\pi}{2l+1}} \tilde{G}_l \; Y_l^m(N(V)). \tag{3.28}$$

The derivative with respect to texture is defined by

$$\frac{\partial I}{\partial \rho} = \sum_{l=0}^{n} \sum_{m=-l}^{l} U_{l,m} \sqrt{\frac{4\pi}{2l+1}} \tilde{G}_l \; Y_l^m(N(V)). \tag{3.29}$$

Note that we assume texture variations are piece-wise constant with respect to our triangle mesh discretization.

3.7.4 Differentiating Skylight Parameters

To model possible outdoor daylight conditions, we use the analytical Preetham skylight model [Preetham et al., 1999]. This model is calibrated by atmospheric data and parameterized by two intuitive parameters: turbidity τ, which describes the cloudiness of the atmosphere, and two polar angles $\theta_s \in [0, \pi/2], \phi_s \in [0, 2\pi]$, which encode the direction of the sun. Note that θ_s, ϕ_s are not the polar angles θ, ϕ for representing incoming light direction ω in $u(\omega)$. The spherical harmonics representation of the Preetham skylight is presented in [Habel et al., 2008] as

$$u(\omega) = \sum_{l=0}^{6} \sum_{m=-l}^{l} U_{l,m}(\theta_s, \phi_s, \tau) Y_l^m(\omega).$$

This is derived by first performing a non-linear least squares fit to write $U_{l,m}$ as a polynomial of θ_s and τ which lets them solve for $\tilde{U}_{l,m}(\theta_s, \tau) = U_{l,m}(\theta_s, 0, \tau)$

$$\tilde{U}_{l,m}(\theta_s, \tau) = \sum_{i=0}^{13} \sum_{j=0}^{7} (p_{l,m})_{i,j} \theta_s^i \tau^j,$$

where $(p_{l,m})_{i,j}$ are scalar coefficients, then $U_{l,m}(\theta_s, \phi_s, \tau)$ can be computed by applying a spherical harmonics rotation with ϕ_s using

$$U_{l,m}(\theta_s, \phi_s, \tau) = \tilde{U}_{l,m}(\theta_s, \tau) \cos(m\phi_s) + \tilde{U}_{l,-m}(\theta_s, \tau) \sin(m\phi_s).$$

We refer the reader to [Preetham et al., 1999] for more detail. For the purposes of this article we just need the above form to compute the derivatives.

Derivatives

The derivatives of the lighting with respect to the skylight parameters (θ_s, ϕ_s, τ) are

$$\frac{\partial U_{l,m}(\theta_s, \phi_s, \tau)}{\partial \phi_s} = -m\tilde{U}_{l,m}(\theta_s, \tau) \sin(m\phi_s) + m\tilde{U}_{l,-m}(\theta_s, \tau) \cos(m\phi_s) \tag{3.30}$$

$$\frac{\partial U_{l,m}(\theta_s, \phi_s, \tau)}{\partial \theta_s} = \frac{\partial \tilde{U}_{l,m}(\theta_s, \tau) \cos(m\phi_s) + \tilde{U}_{l,-m}(\theta_s, \tau) \sin(m\phi_s)}{\partial \theta_s} \tag{3.31}$$

$$= \sum_{ij} i\theta_s^{i-1} \tau^j (p_{l,m})_{i,j} \cos(m\phi_s) + \sum_{ij} i\theta_s^{i-1} (p_{l,-m})_{i,j} \sin(m\phi_s) \tag{3.32}$$

$$\frac{\partial U_{l,m}(\theta_s, \phi_s, \tau)}{\partial \tau} = \sum_{ij} j\theta_s^i \tau^{j-1} (p_{l,m})_{i,j} \cos(m\phi_s) + \sum_{ij} j\theta_s^i \tau^{j-1} (p_{l,-m})_{i,j} \sin(m\phi_s) \tag{3.33}$$

3.7.5 Derivatives of Surface Normals

Taking the derivative of the rendered image I with respect to surface normals N is an essential task for computing the derivative of I with respect to the geometry V. Specifically, the derivative of the rendering equation Equation 3.25 with respect to V is

$$\frac{\partial I}{\partial V} = \frac{\partial \rho}{\partial V} F + \rho \frac{\partial F}{\partial V} \tag{3.34}$$
$$= \frac{\partial \rho}{\partial V} F + \rho \frac{\partial F}{\partial N}\frac{\partial N}{\partial V} \tag{3.35}$$

We assume the texture variations are piece-wise constant with respect to our triangle mesh discretization and omit the first term $\partial\rho/\partial V$ as the magnitude is zero. The expression for $\partial N/\partial V$ is provided in Sec. 3.3.2. The expression for $\partial F/\partial N_i$ on face i is

$$\frac{\partial F}{\partial N_i} = \sum_{l=0}^{n}\sum_{m=-l}^{l} U_{l,m}\sqrt{\frac{4\pi}{2l+1}}\tilde{G}_l \frac{\partial Y_l^m}{\partial N_i}, \tag{3.36}$$

where the $\partial Y_l^m/\partial N_i$ is the derivative of the spherical harmonics with respect to the face normal N_i.

To begin this derivation recall the relationship between a unit normal vector $\mathbf{n} = (n_x, n_y, n_z)$ and its corresponding polar angles θ, ϕ

$$\theta = \cos^{-1}\left(\frac{n_z}{\sqrt{n_x^2+n_y^2+n_z^2}}\right) \qquad \phi = \tan^{-1}\left(\frac{n_y}{n_x}\right),$$

we can compute the derivative of spherical harmonics with respect to the normal vector through

$$\begin{aligned}
&\frac{\partial Y_l^m(\theta,\phi)}{\partial \mathbf{n}} \\
&= K_l^m \begin{cases} (-1)^m\sqrt{2}\left[\frac{\partial P_l^{-m}(\cos\theta)}{\partial\theta}\frac{\partial\theta}{\partial\mathbf{n}}\sin(-m\phi) + P_l^{-m}(\cos\theta)\frac{\partial\sin(-m\phi)}{\partial\phi}\frac{\partial\phi}{\partial\mathbf{n}}\right] & m<0 \\ (-1)^m\sqrt{2}\left[\frac{\partial P_l^{m}(\cos\theta)}{\partial\theta}\frac{\partial\theta}{\partial\mathbf{n}}\cos(m\phi) + P_l^{m}(\cos\theta)\frac{\partial\cos(m\phi)}{\partial\phi}\frac{\partial\phi}{\partial\mathbf{n}}\right] & m>0 \\ \frac{\partial P_l^0(\cos\theta)}{\partial\theta}\frac{\partial\theta}{\partial\mathbf{n}} & m=0 \end{cases} \\
&= K_l^m \begin{cases} (-1)^m\sqrt{2}\left[\frac{\partial P_l^{-m}(\cos\theta)}{\partial\theta}\frac{\partial\theta}{\partial\mathbf{n}}\sin(-m\phi) - mP_l^{-m}(\cos\theta)\cos(-m\phi)\frac{\partial\phi}{\partial\mathbf{n}}\right] & m<0 \\ (-1)^m\sqrt{2}\left[\frac{\partial P_l^{m}(\cos\theta)}{\partial\theta}\frac{\partial\theta}{\partial\mathbf{n}}\cos(m\phi) - mP_l^{m}(\cos\theta)\sin(m\phi)\frac{\partial\phi}{\partial\mathbf{n}}\right] & m>0 \\ \frac{\partial P_l^0(\cos\theta)}{\partial\theta}\frac{\partial\theta}{\partial\mathbf{n}} & m=0 \end{cases}
\end{aligned} \tag{3.37}$$

Note that the derivative of the associated Legendre polynomials $P_l^m(\cos\theta)$ can be computed by applying the recurrence formula [Dunster, 2010]

$$\begin{aligned}\frac{\partial P_l^m(\cos\theta)}{\partial\theta} &= \frac{-\cos\theta(l+1)P_l^m(\cos\theta) + (l-m+1)P_{l+1}^m(\cos\theta)}{\cos^2\theta - 1} \times (-\sin\theta) \\ &= \frac{-\cos\theta(l+1)P_l^m(\cos\theta) + (l-m+1)P_{l+1}^m(\cos\theta)}{\sin\theta}.\end{aligned} \tag{3.38}$$

Thus the derivatives of polar angles (θ, ϕ) with respect to surface normals $\mathbf{n} = [n_x, n_y, n_z]$ are

$$\frac{\partial\theta}{\partial\mathbf{n}} = \left[\frac{\partial\theta}{\partial n_x}, \frac{\partial\theta}{\partial n_y}, \frac{\partial\theta}{\partial n_z}\right] = \frac{\left[n_x n_z,\ n_y n_z,\ -(n_x^2 + n_y^2)\right]}{(n_x^2 + n_y^2 + n_z^2)\sqrt{n_x^2 + n_y^2}}, \tag{3.39}$$

$$\frac{\partial\phi}{\partial\mathbf{n}} = \left[\frac{\partial\phi}{\partial n_x}, \frac{\partial\phi}{\partial n_y}, \frac{\partial\phi}{\partial n_z}\right] = \left[\frac{-n_y}{n_x^2 + n_y^2},\ \frac{n_x}{n_x^2 + n_y^2},\ 0\right]. \tag{3.40}$$

In summary, the results of Equation 3.37, Equation 3.38, Equation 3.39, and Equation 3.40 tell us how to compute $\partial Y_l^m / \partial N_i$. Then the derivative of the pixel j with respect to vertex p which belongs to face i can be computed as

$$\begin{aligned}\frac{\partial I_j}{\partial V_p} &\approx \rho_j \frac{\partial F}{\partial N_i}\frac{\partial N_i}{\partial V_p} \\ &= \rho_j \sum_{l=0}^{n}\sum_{m=-l}^{l} U_{l,m}\sqrt{\frac{4\pi}{2l+1}}\tilde{G}_l \frac{\partial Y_l^m(\theta,\phi)}{\partial N_i}\frac{\partial N_i}{\partial V_p}.\end{aligned} \tag{3.41}$$

3.7.6 Adversarial Training Implementation Detail

Our adversarial training is based on the basic idea of injecting adversarial examples into the training set at each step of training and continuously updating the adversaries according to the current model parameters [Goodfellow et al., 2015; Kurakin et al., 2017]. Our experiments inject 100 adversarial lighting examples to the CIFAR-100 data ($\approx$ 0.17% of the training set) and keep updating these adversaries at each epoch.

We compute the adversarial lighting examples using the orange models collected from `cgtrader.com` and `turbosquid.com`. We uses five gray-scale background colors with intensities 0.0, 0.25, 0.5, 0.75, 1.0 to mimic images in the CIFAR-100 which contains many pure color backgrounds. Our orthographic cameras are placed at polar angle $\theta = \pi/3$ with 10 uniformly sampled azimuthal angles ranging from $\phi = 0$ to 2π. Our initial spherical harmonics lighting is the same as other experiments, using the real-world lighting data provided in [Ramamoorthi and Hanrahan, 2001a]. Our stepsize for computing adversaries is 0.05 along the direction of lighting gradients. We run our adversarial lighting iterations until fooling the network or reaching the maximum 30 iterations to avoid too extreme lighting conditions, such as turning the lights off.

Our random lighting examples are constructed at each epoch by randomly perturbing

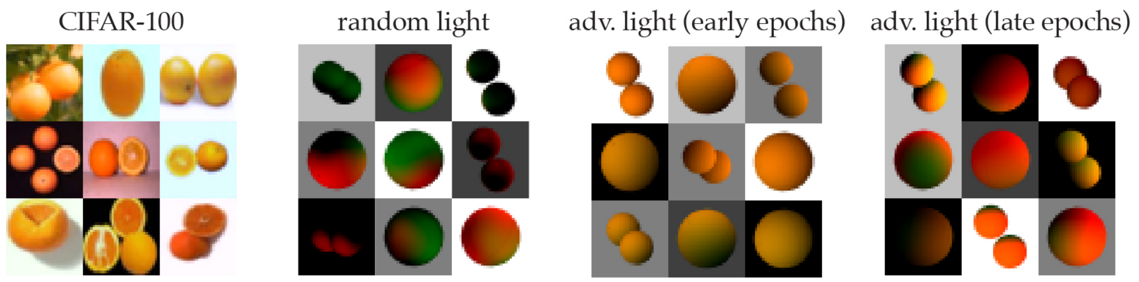


Figure 3.16: This figure visualizes the images of oranges from CIFAR-100, random lighting, and adversarial lighting. In early training stage, small changes in lighting are sufficient to construct adversarial examples. In late training stage, we require more dramatic changes as the model is becoming robust to differ lightings.

the lighting coefficients ranging from -0.5 to 0.5.

When training the 16-layers WideResNet [Zagoruyko and Komodakis, 2016] with wide-factor 4, we use batch size 128, learning rate 0.125, dropout rate 0.3, and the standard cross entropy loss. We implement the training using PyTorch [Paszke et al., 2019], with the SGD optimizer and set the Nesterov momentum 0.9, weight decay 5e-4. We train the model for 150 epochs and use the one with best accuracy on the validation set. Fig. 3.16 shows examples of our adversarial lights at different training stages. In the early stages, the model is not robust to different lighting conditions, thus small lighting perturbations are sufficient to fool the model. In the late stages, the network becomes more robust to different lightings. Thus it requires dramatic changes to fool a model or even fail to fool the model within 30 iterations.

3.7.7 Evaluate Rendering Quality

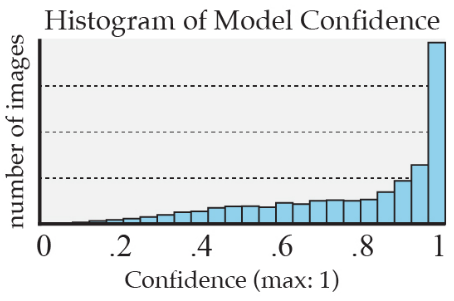


Figure 3.17: Prediction confidence on rendered images, showing our rendering quality is faithful enough to be confidently recognized by ImageNet models.

We evaluated our rendering quality by whether our rendered images are recognizable by models trained on real photographs. Although large 3D shape datasets, such as ShapeNet [Chang et al., 2015], are available, they do not have have geometries or textures at the resolutions necessary to create realistic renderings. We collected 75 high-quality textured 3D shapes from `cgtrader.com` and `turbosquid.com` to evaluate our rendering quality. We augmented the shapes by changing the field of view, backgrounds, and viewing directions, then kept the configurations that were correctly classified by a pre-trained ResNet-101 on ImageNet. Specifically, we place the centroid, calculated as the weighted average of the mesh vertices where the weights are the vertex areas, at the origin and normalize shapes to the range -1 to 1; the field of view is chosen to be 2 and 3 in the same unit with the normalized shape; background images include plain colors and real photos, which have small influence on model predictions; viewing directions are chosen to be 60 degree zenith and uniformly sampled 16 views from 0 to 2π azimuthal angle. In Fig.3.17, we show that the histogram of model confidence on the correct labels over 10,000 correctly classified rendered images from our differentiable renderer. The confidence is computed using the softmax function and the results show that our rendering quality is faithful enough to be recognized by models trained on natural images.

Chapter 4

Neural Subdivision

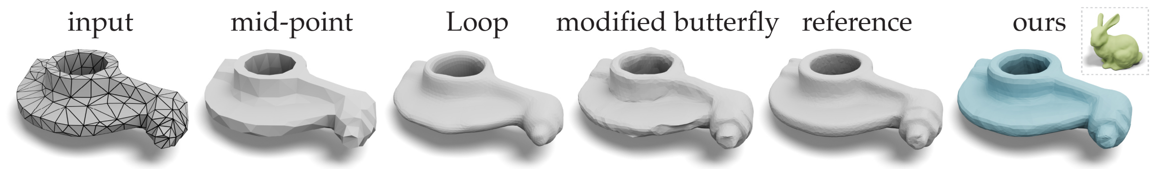


Figure 4.1: Our neural subdivision framework performs geometry-aware subdivision, reconstructing the reference rocker arm that we decimated to obtain the coarse input with high accuracy, even though it was only trained on one single model - the Stanford bunny. Neural subdivision does not suffer from the inherent limitations of classic subdivisions, such as volume shrinkage and over-smoothing ([Loop, 1987]), or amplification of tessellation artifacts (modified butterfly [Zorin et al., 1996]). Throughout this paper, we use green to denote the training shape, and blue for the neural subdivision output.

This paper introduces *Neural Subdivision*, a novel framework for data-driven coarse-to-fine geometry modeling. During inference, our method takes a coarse triangle mesh as input and recursively subdivides it to a finer geometry by applying the fixed topological updates of Loop Subdivision, but predicting vertex positions using a neural network conditioned on the local geometry of a patch. This approach enables us to learn complex non-linear subdivision schemes, beyond simple linear averaging used in classical techniques. One of our key contributions is a novel self-supervised training setup that only requires a set of high-resolution meshes for learning network weights. For any training shape, we stochastically generate diverse low-resolution discretizations of coarse counterparts, while maintaining a bijective mapping from the coarse geometry to the fine mesh that prescribes the exact target position of every new vertex inserted during the subdivision process. This leads to a very efficient and accurate loss function for conditional mesh generation, and enables us to train a method that generalizes across discretizations and favors preserving the manifold structure of the output. During training we optimize for the same set of network weights across all local mesh patches, thus providing an architecture that is not constrained to a specific input mesh, fixed genus, or category. Our network encodes patch geometry in a local frame in a rotation- and translation-invariant manner. Jointly, these design choices enable our method to generalize well, and we demonstrate that even when trained on a single

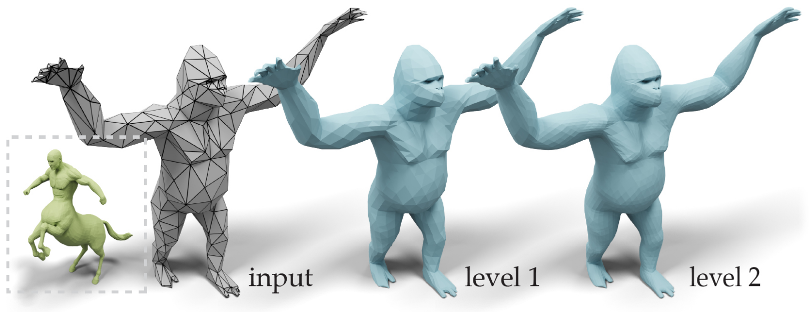


Figure 4.2: Neural subdivision refines different parts of a mesh differently, conditioned on the local geometry. Here, the network was trained on the centaur model (green) and then evaluated on a coarse gorilla mesh (gray).

high-resolution mesh our method generates reasonable subdivisions for novel shapes.

4.1 Introduction

Subdivision surfaces are defined by deterministic, recursive upsampling of a discrete surface mesh. Classic methods work by performing two steps: each input mesh element is *divided* into many elements (e.g., one triangle becomes four) by splitting edges and adding vertices. The positions of the mesh vertices are then smoothed by taking a weighted average of their neighbors' positions according to a weighting scheme based purely on the local mesh connectivity. Subdivision surfaces are well studied and have rich theory connecting their limit surfaces (applying an infinite number of subdivide-and-smooth iterations) to traditional splines. They are a standard paradigm in surface modeling tools, allowing modelers to sculpt shapes in a coarse-to-fine manner. A modeler may start with a very coarse cage, adjust vertex positions, then subdivide once, adjust the finer mesh vertices, and repeat this process until satisfied.

While existing subdivision methods are well-suited for this sort of interactive modeling, they fall short when used to automatically upsample a low resolution asset. Without a user's guidance, classic methods will overly smooth the entire shape (see Fig. 4.1). Popular methods based on simple linear averaging do not identify details to maintain or accentuate during upsampling. They make no use of the geometric context of a local patch of a surface. Furthermore, classic methods based on fixed one-size-fits-all weighting rules are determined for their general convergence and smoothness properties. This ignores an opportunity to leverage the massive amount of information lurking in the wealth of existing 3D models.

We propose *Neural Subdivision*. We recursively subdivide an input triangle mesh by applying the same fixed topological updates of classic Loop Subdivision, but move vertices according to a neural network conditioned on the local patch geometry. We train the shared weights of this network to learn a geometry-dependent non-linear subdivision that goes

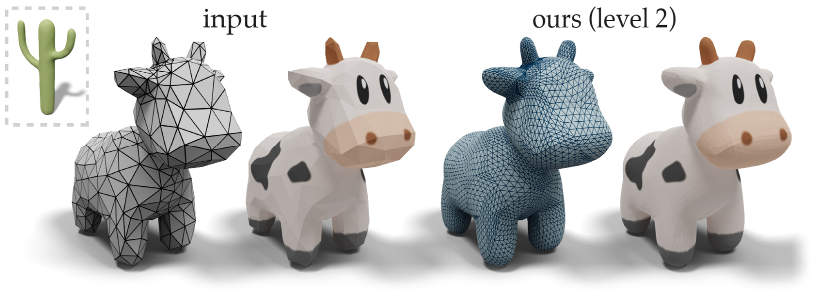


Figure 4.3: Neural subdivision can adapt to different input triangulations and output a high-resolution surface mesh accordingly. This enables us to use it directly in the graphics pipeline such as texture mapping.

beyond classic linear averaging (see Fig. 4.2). The choice of training data tailors the network to a particular class, type or diversity of geometries.

An immediate challenge is how to collect training data pairs. There is an ever-growing number of 3D models available. However, many if not most of them were not created using a subdivision modeling tool. Even among those that were, the final model does not retain information to replay the modeler's vertex displacements. In the absence of paired data for a supervised training approach, we propose a novel method to *self-supervise* given only high-resolution surface meshes of arbitrary origin/connectivity at training time. We stochastically generate candidate low-resolution versions of a training exemplar while maintaining a bijective correspondence between their surfaces. This correspondence enables a novel loss function that is more efficient and accurate compared to existing methods. By construction, this training regime ensures *generalization across discretization*.

In contrast to existing generative models for surfaces, our output is a surface mesh with deterministic connectivity based on the input, enabling direct use in the standard graphics pipeline such as texture mapping (see Fig. 4.3). By sharing weights and training across all local patches of all the training meshes, we learn a rule based on the local neighborhood rather than the entire shape. Compared to existing methods, this frees our network from being constrained to a fixed genus, relying on a template, or requiring an extremely large collection of shapes during training. We demonstrate that even when trained on a single shape, our method can generalize to novel meshes. We design our network to encode vertex position data in a local frame ensuring rotation and translation invariances without resorting to handcrafted predefined feature descriptors.

We demonstrate the effectiveness of our method with a variety of qualitative and quantitative experiments. Our method generates subdivided meshes that are closer to the true high-resolution shapes than traditional interpolatory and non-interpolatory subdivision methods, even when trained with a small number of very dissimilar exemplars. We introduce a quantitative benchmark and show significant gains over classic subdivision methods when measuring upsampling fidelity. Finally, we show prototypical applications of Neural

Subdivision to low-poly mesh upsampling and 3D modeling.

4.2 Related Work

Our work builds directly upon the foundations of classic subdivision surfaces and connects to the rapidly advancing field of neural geometry learning. We focus this section on establishing context with past subdivision schemes and contrasting our geometric learning contributions with contemporary works.

4.2.1 Subdivision Surfaces

The basic idea of subdivision is to "define a smooth curve or surface as the limit of sequence of successive refinements" [Zorin et al., 2000]. This broad definition admits a wide variety or "zoo" of different subdivision schemes that would be outside the scope of this paper to cover thoroughly. The history of subdivision surfaces reaches back to the early work on irregular polygon meshes [Doo and Sabin, 1998; Doo, 1978] and the now ubiquitous Catmull-Clark subdivision which produces quad meshes [Catmull and Clark, 1998]. The linear method of Loop [1987] for triangle meshes has reached similar popularity, and is the basis for our non-linear neural subdivision.

Classic linear subdivision methods are defined by a combinatorial update (splitting faces, adding vertices, and/or flipping edges [Kobbelt, 2000]) and a vertex smoothing (repositioning step) based on local averaging of neighboring vertex positions. Subdivision methods are well studied from a theoretical perspective in terms of existence, direct evaluation, and continuity of the limit surface [Stam, 1998; Zorin, 2007; Karciauskas and Peters, 2018]. Modelers typically manipulate a subdivision surface in a coarse to fine fashion. Most modeling tools already visualize the limit surface or some approximation of it, while the user manipulates the coarse level (cage) (see Fig. 4.23). Beyond moving vertices, users can control the surface by adding creases (sharp edges) [Hoppe et al., 1994; DeRose et al., 1998]. Non-interpolating methods such as Catmull-Clark or Loop appear to be the most popular, but interpolating methods do exist (e.g., [Dyn et al., 1990; Kobbelt, 1996; Zorin et al., 1996]) and have similar smoothness guarantees, although *fairness* is harder to achieve (see Fig. 4.1). Linear methods are easier to analyze and design to guarantee smoothness. As a result, capturing details is left to the modeler or a deterministic procedural routine (e.g., [Tobler et al., 2002a,b; Velho et al., 2002]).

Our neural subdivision acts similar to non-linear subdivision methods, with the subdivision rule in this case being a non-linear function learned by a neural network. Non-linear subdivision has been studied from the mathematical perspective [Floater and Micchelli, 1997; Schaefer et al., 2008] and also as a mechanism to maintain certain geometric invariants during each level of subdivision (e.g., circle-preserving [Sabin and Dodgson, 2004], quad planarity [Liu et al., 2006; Bobenko et al., 2020], developability [Tang et al., 2014; Ra-

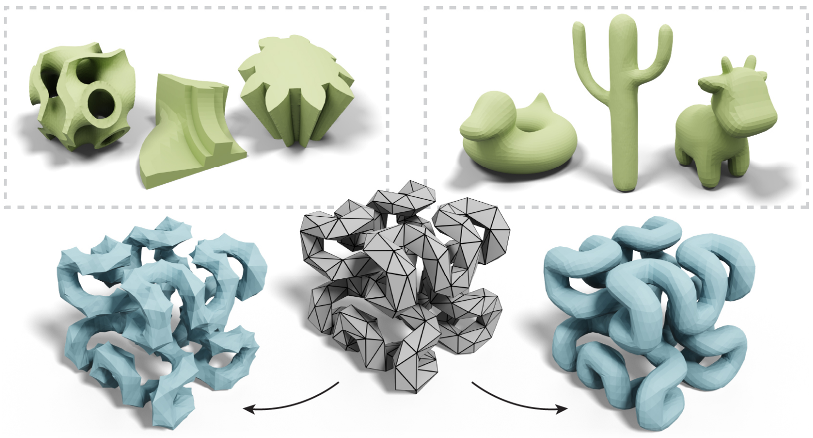


Figure 4.4: Our subdivision is data-driven. Training on a set of mechanitical objects (left, green) or a set of smooth organic objects (right, green) leads to drastically different styles (blue). ©Gyroid Puzzl by eemmett (top left) and Hilbert Cube by tbuser (bottom) under CC BY-SA.

binovich et al., 2018], Möbius-regularity [Vaxman et al., 2018], cloth wrinkliness [Kavan et al., 2011]). One general approach is to combine a linear subdivision with an online geometric optimization, and recursively apply the non-linear rule an arbitrary, if not infinite number of times, akin to classic linear rules. Our approach can be viewed as an extreme form of precomputation, where the optimization is the training procedure and the fixed network is applied generally as a non-linear function evaluation. The choice of data in the training will influence the "style" of our non-linear subdivision (see Fig. 4.4). Although our method is non-linear, it is trained to work well for a pre-specified finite number of times.

Recently, Preiner et al. [2019] introduced a new non-linear subdivision method that treats the coarse shape probabilistically. Their contributions are orthogonal to ours, and while we base our method on Loop subdivision, we could in theory extend our network to learn on top of this more powerful subdivision method.

4.2.2 Neural Geometry Learning

Recent advances in generative neural networks enabled the use of learnable components in 3D modeling applications such as shape completion [Li et al., 2019], single-view [Tatarchenko et al., 2019] and multi-view [Sitzmann et al., 2019] reconstruction, and modeling-by-parts [Chaudhuri et al., 2020].

The closest to our neural mesh subdivision application are the deep point cloud upsampling techniques [Yu et al., 2018b; Li et al., 2019; Wang et al., 2019b]. The disadvantage of using a point cloud as input is that it lacks connectivity information, and requires the neural network to implicitly estimate the structure of the underlying manifold. Meshes can also be more efficient at representing feature-less regions with larger planar elements, providing a wider receptive field to our mesh-based neural network. Mesh output is preferred for many

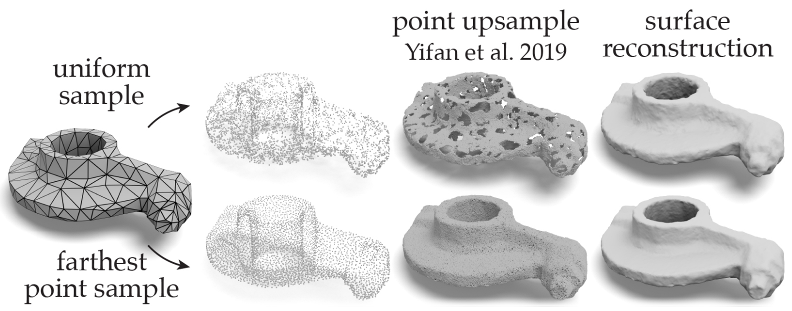


Figure 4.5: One can use existing point upsampling methods to refine coarse meshes by (1) sampling, (2) upsampling [Wang et al., 2019b], and (3) reconstruction [Kazhdan and Hoppe, 2013]. However, this may lead to artifacts since it lacks information about the surface, and requires the use of expensive surface reconstruction as a post-process.

standard graphics pipelines, thus, a post process is often required [Kazhdan and Hoppe, 2013] to convert the output of point-based methods to meshes, which prevents building an end-to-end trainable system. Fig. 4.5 illustrates the output of a point upsampling method that was pre-trained on a collection of statues [Wang et al., 2019b] (see App. 4.8.1 for implementation details).

Our work is related to other neural mesh generation techniques. Free-form generation of meshes as a set of vertices and faces is infeasible with current deep learning methods, due to the lack of regular structure, uneven discretization, and combinatorial variability in the possible outputs, limiting such approaches to very coarse outputs [Dai and Nießner, 2019]. A common alternative is to deform a global template either by predicting vertex coordinates [Tan et al., 2018; Ranjan et al., 2018] or by training a deformation network that warps the entire 3D domain conditioned on a latent vector that encodes the deformation target [Groueix et al., 2018a; Yifan et al., 2020]. While these approaches usually produce meshes with higher resolution, their output is limited to deformations of a single shape. Some techniques propose using generic templates such as spheres [Wang et al., 2018a; Wen et al., 2019] or 2D atlases [Groueix et al., 2018b], which place limitations on the topology of the output. In contrast to these techniques, our method refines the mesh locally, and thus, respects the topology of the input (which could be arbitrary). Another advantage of our local refinement approach is that we do not require co-aligned training data with a well-defined object space, the output of our subdivision networks is translation and rotation invariant since it can be described in a local coordinate system of the input patch.

There are several options for analyzing a mesh patch with a neural network, such as using a local [Masci et al., 2015; Poulenard and Ovsjanikov, 2018] or global [Maron et al., 2017] parameterization to unfold a mesh into 2D grid, or apply graph-based techniques adapted for meshes [Kostrikov et al., 2018; Wang et al., 2019a] (see [Bronstein et al., 2017] for a more comprehensive survey). Our approach is inspired by MeshCNN [Hanocka et al., 2019]. Their method directly learns filters over the local mesh structure via undirected

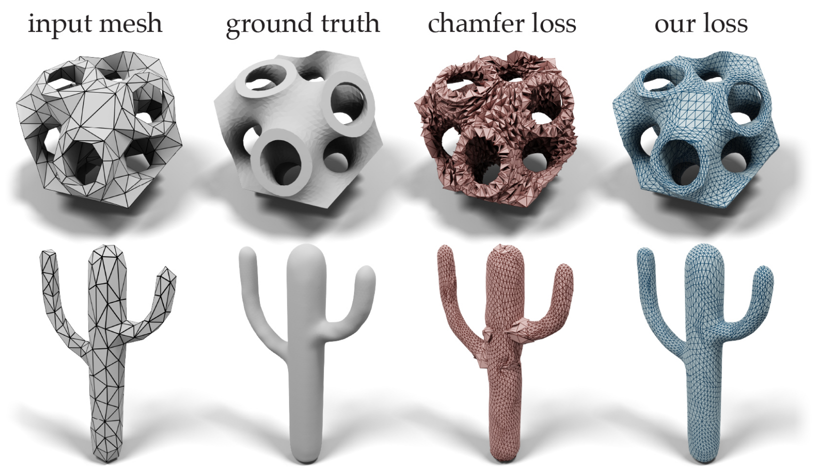


Figure 4.6: We compare the same model trained using (a) chamfer distance (which only measures error between point sets) and (b) our ℓ^2 loss based on shape correspondences. The model trained using the chamfer distance fails to capture the surface topology (red). In contrast, our loss function leads to manifold output meshes (blue). ©Gyroid Puzzle by emmett (top) under CC BY-SA.

edges, and shows applications in deterministic tasks. In contrast, we focus on generative tasks and develop a novel set of features over the *half-flaps* – an edge along with its two adjacent triangles. Each half-flap has a canonical orientation which gives us well-defined local frames which are crucial for our network's rigid motion invariance.

Geometry generation techniques are typically trained with reconstruction losses that measure how well the generated surface approximates the known target. Surface-to-surface distances are commonly employed, with correspondences defined via closest-point queries (aka chamfer distance) [Barrow et al., 1977; Fan et al., 2017]. However, the closest-point approach matches many points to the same point, while leaving other points unmatched, resulting in self-overlaps and unrepresented areas (see Fig. 4.6).

Indeed, prior work demonstrates that using higher quality correspondence (e.g., ground truth mapping) significantly improves results [Groueix et al., 2018a]. While the latter is not available in our setting, we propose a data generation technique for creating various coarse variants of the same high-res mesh with a low-distortion bijective map. Bijectivity is crucial for the quality of our training data, ensuring that no self-overlaps exist and that every part of the target surface has a pre-image on the coarse mesh.

4.3 Neural Subdivision

In the following we overview the main components of our neural subdivision: the test-time inference pipeline, training and loss (Sec. 4.4), data-generation (Sec. 4.4), and finally the network architecture (Sec. 4.5). Later sections will discuss these components in detail.

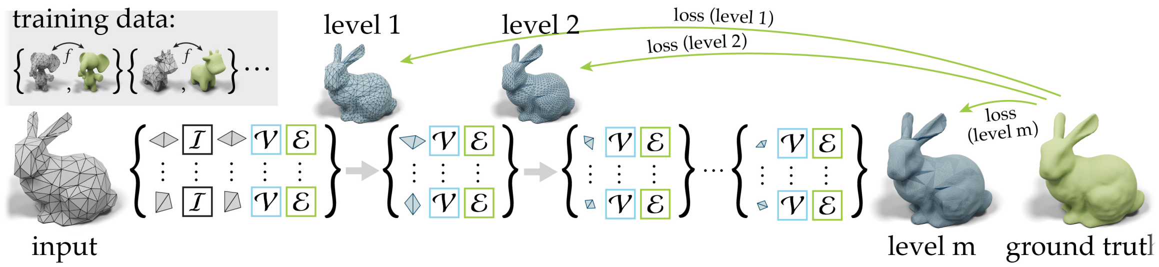


Figure 4.7: Neural subdivision takes a coarse triangle mesh (gray) as input and outputs a sequence of subdivided meshes (blue) with different levels of details. During training, we minimize the ℓ^2 loss from the ground truth (green) to the output meshes (blue) across levels. Our training data consists of pairs of coarse and fine meshes (top left) with a bijective map f between each pair.

Inference. As illustrated in Fig. 4.7, our method takes a coarse triangle mesh (gray) as input and recursively refines it by subdividing each triangle to create additional vertices and faces. The output is a sequence of subdivided meshes (blue) with different levels of details. Our subdivision process follows a simple topological update rule (same as Loop), namely inserting new vertices at the midpoints of all edges. It then uses a neural network to predict new positions for all vertices, at each new level of subdivision.

Training and loss. The data we generate provides us with correspondences between predicted vertices and points inside the triangle on the ground truth shape. We train our network with the simple ℓ^2 loss, by measuring the distance between each predicted vertex position at every level of subdivision and its corresponding point on the original shape (green). As there is no existing dataset consisting of pairs of high-quality meshes and subdivision surfaces in correspondence, we instead develop a novel technique for generating training data, comprising coarse and fine meshes with bijective mappings between them.

Data generation. We first note that each vertex v created from a subdivision step has a well-defined mapping back to the coarse mesh, defined by mapping that vertex to its corresponding midpoint. Thus, each subdivided mesh at any level of subdivision can be mapped back to the initial coarse mesh via a sequence of mid-point-to-vertex or vertex-to-vertex maps. In practice we use barycentric coordinates to encode this subdivided-to-coarse bijective mapping, g. Hence, if we had a bijective mapping f between the coarse mesh and the original mesh, we could define a unique point on the original mesh corresponding to v, by compositing the two maps: $f(g(v))$.

Thus, the only missing part is to create coarse and fine meshes with bijective mappings between them. We achieve this by taking a high-resolution training mesh and sequentially coarsening it by applying random sequences of edge collapses, thereby generating a sequence of coarsened meshes. We maintain low-distortion correspondences be-

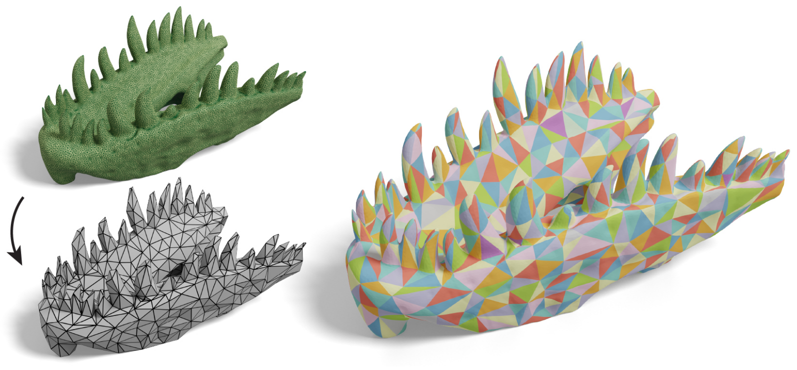
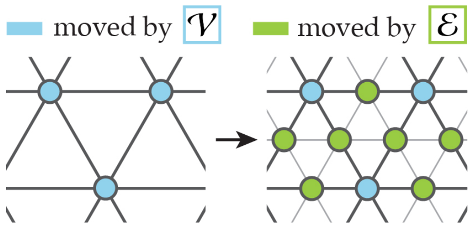


Figure 4.8: Given an edge collapse algorithm of choice, we plug in our successive self-parameterization described in Sec. 4.4.1 to compute a bijective map between the original mesh (green) and its decimated version (gray). We visualize the map by coloring the fine mesh using the triangulation of the coarse mesh (right). ©Tarbosaurus Skull by gpvillamil under CC BY-SA.

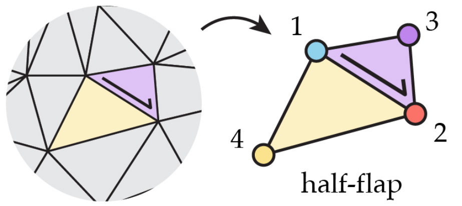
tween the coarsened and original mesh by computing a conformal map between the 1-ring edge neighborhood (before the collapse) and the 1-ring vertex neighborhood (after the collapse). Composition of these maps creates a dense bijective map f between the coarse and original meshes, which is then directly applied to the training (Fig. 4.8).

Advantages of our training approach. In comparison to closest point losses that are commonly used to train generative neural networks, our correspondence-based loss is aware of the manifold structure (Fig. 4.6) and is orders of magnitude faster to compute (Fig. 4.17). Bijectivity and continuity of the map ensure that the entire ground truth surface is captured by some region of the coarse mesh (Fig. 4.10). The low distortion encourages uniformity, which in turn enables the reproduction of the target surface with just a few uniform subdivisions, and, more importantly reduces the variance in the signal the network needs to learn. We can further leverage the low-distortion map to map an additional signal, such as texture (Fig. 4.3). As our training data contains many pairs with different random decimations of the same ground truth (App. 4.8.5), our network is able to learn how to generalize across discretization.

Network architecture. Similarly to the subdivision process, the learnable modules of our network are applied recursively. They operate over local mesh neighborhoods and predict differential features (meaning they represent geometry in the local coordinates of the mesh, and not in world coordinates). These features are then used to compute vertex coordinates at the new level of subdivision. We define three types of modules applied at three sequential steps. During the Initialization step, we first compute differential per-vertex quantities that are based on the local coordinate frame (defined in Sec. 4.5). A learnable module $\mathcal{I}$ is applied to the 1-ring neighborhood of every vertex to map these differential quantities to a high-dimensional feature vector stored at the vertex. Note that this high-dimensional feature vector is a concatenation of a learnable latent space which *encodes* local geometry of the patch, and differential quantities which directly *represent* local geometry and enable us to reconstruct the vertex coordinates.



For each subsequent subdivision iteration, we assumes that the topology is updated following the Loop subdivision scheme, splitting each edge at the midpoint, and consequently, subdividing each triangle into four (see inset). A Vertex step uses the module $\mathcal{V}$ to predict vertex features for the next level of subdivision based on its 1-ring neighborhood, where vertices affected by this step only involve corners of triangles at the previous mesh level. Then, an Edge step uses the module $\mathcal{E}$ to compute features of vertices added at midpoints based on the pair of vertices that were connected by an edge at the previous mesh level.



Our modules share a very similar architecture and heavily rely on a learnable operator defined over *a half-flap*: a directed edge and its two adjacent triangles (see the inset). We use the directed edge to define the local coordinate frame which is used to estimate the differential features of either input or output of learnable modules. Note also that the directed edge allows us to order the four adjacent vertices of the flap in a canonical way. We concatenate their features and feed them into shallow multi-layer perceptrons (MLP). The weights of the MLPs are shared within each module type and across all levels of subdivision. Both modules $\mathcal{I}$ and $\mathcal{V}$ process all half-flaps defined by an outgoing edge and use average pooling to combine the half-flap features into per-vertex features. The module $\mathcal{E}$ also combines features from two half-flaps (both directions of the edge) via average pooling. Since our architecture is local, and uses input and output features that are invariant to rigid motions, it exhibits an impressive ability to generalize from example, even when trained on a single fine mesh.

4.4 Data Generation and Training

While our network architecture and invariant layers are crucial for its ability to learn subdivisions, it by its own is only half of the two main components that together facilitate high-quality neural subdivisions. The other half consists of the training process, data and the loss function.

Consider a naive approach to the subdivision training: generate pairs of coarse/fine meshes by a decimation algorithm; measure the distance between the network's predicted subdivision and the ground truth, for instance by the average distance between predicted points and their projections on the ground truth mesh; iterate over coarse/fine pairs while

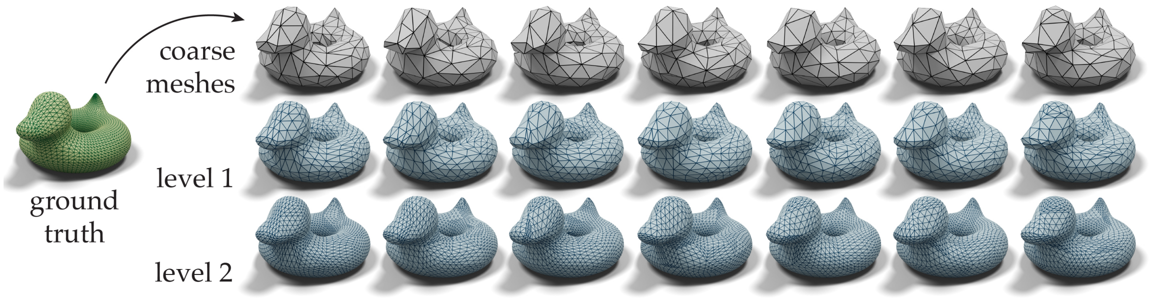


Figure 4.9: Given a ground truth shape (green), we use random edge collapses to create several coarse meshes (gray). For each coarse mesh, we subdivide the mesh and use the bijective map to determine the position on the ground truth for all the vertices across different levels. The blue meshes are the ground truth subdivisions that exhibit one-to-one point correspondences to the network predictions.

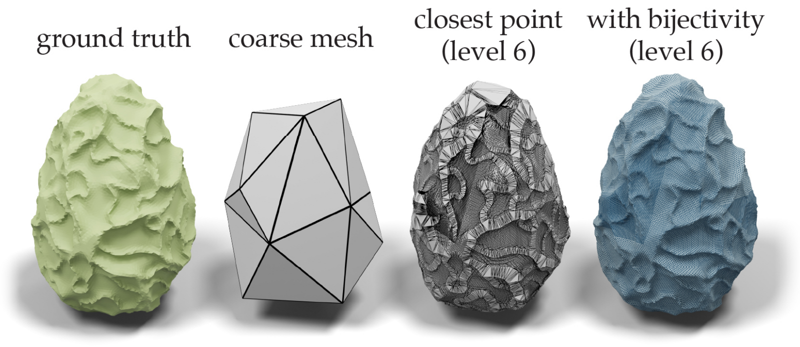


Figure 4.10: Given a ground truth/coarse mesh pair, naively using "closest-point-on-mesh" to estimate correspondences between the level-6 subdivided mesh and the ground truth results in a non-bijective map, causing the loss function to fail to capture the entire ground truth mesh (third column). Our successive self-parameterization ensures bijectivity, which implies the entire ground truth surface will be captured (right).

optimizing the loss. This naive approach has a major caveat. Computing correspondences using the chamfer-like loss (Fig. 4.6) or point-to-mesh distance (Fig. 4.10) is known to lead to incorrect and self-overlapping matches between shapes. This leads to a poor training set up because the loss itself exhibits artifacts.

In lieu of this naive approach, we consider the fact that a pair of coarse and fine meshes both approximate the same underlying smooth surface. This motivates us to compute the correspondences based on the intrinsic geometry, instead of an ad-hoc correspondence. The outcome is a high-quality bijective map between each pair of coarse and fine meshes, enabling us to obtain one-to-one point correspondences. Therefore a simple ℓ^2 loss is sufficient to correctly measure the error between every level of neural subdivision and the ground truth shape.

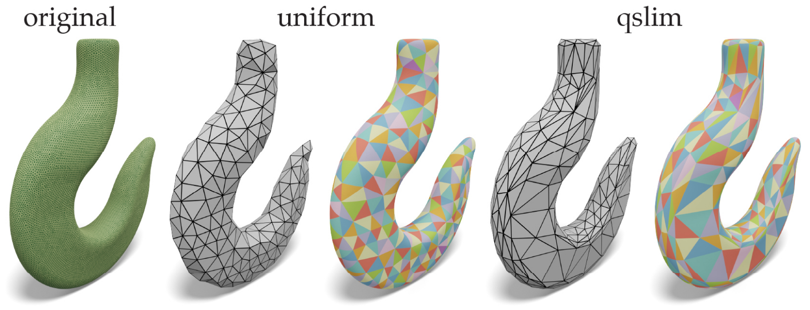


Figure 4.11: Different edge-collapse algorithms can be used in a plug-and-play manner to create, for instance, a uniform-area parameterization (middle) and an appearance-preserving parameterization (right). This flexibility is used to create training data with diverse types of discretizations.

4.4.1 Successive Self-Parameterization

One possible way to obtain point correspondences is to apply general shape matching techniques. But ensuring bijectivity in general shape matching is difficult. For instance, it requires the two shapes to have the same number of vertices [Vestner et al., 2017b], or a user-guided common domain [Schreiner et al., 2004; Praun et al., 2001], or user-provided landmark correspondences [Kraevoy and Sheffer, 2004; Aigerman et al., 2014, 2015] (see [Van Kaick et al., 2011] for a survey). However, our problem is considerably simpler, since we aim to construct a map between different discretizations of the same shape, and we have full control over the decimation procedure.

The closest solution to our problem is a seminal work – MAPS [Lee et al., 1998] – on self-parameterization. Given an input mesh, MAPS computes the bijective map by successively removing vertices of the maximum independent set. Since then, several improvements have been proposed [Guskov et al., 2000, 2002; Khodakovsky et al., 2003]. Unfortunately, they cannot be directly applied to edge collapses for creating training data for our learning task (see App. 4.8.4). We need an algorithm that has the flexibility to be used with any edge decimation method, so that we can generate a diverse collection of coarse meshes (see Fig. 4.11). Fortunately, the idea from [Cohen et al., 1997, 1998, 2003] for minimizing mesh/texture deviation leads us to generalize the idea of MAPS to any edge collapses.

Our method for computing the bijective map, designed specifically for creating data to train neural subdivision, combines the idea of self-parameterization from MAPS [Lee et al., 1998] and the idea of successive mapping from [Cohen et al., 1997, 1998, 2003]. Thus, we call it *successive self-parameterization*. This combination enables us to compute the parameterization intrinsically to avoid the requirement of having a given UV map, such as in the method of Liu et al. [2017c]. The result of the combination is extremely simple. It is a two-step module that can be applied to any choice of edge-collapse algorithm (see Fig. 4.11) and it outputs a bijective map after the decimation. Hence, the inputs to successive self-parameterization are a triangle mesh and an edge collapse algorithm of choice, and the

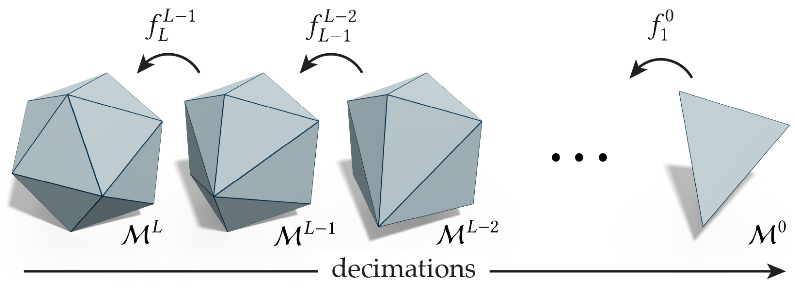


Figure 4.12: We compute a bijective map for each edge collapse. The bijective map from the coarsest mesh $\mathcal{M}^0$ to the input mesh $\mathcal{M}^L$ is then computed by composing all the maps f_l^{l-1}.

output is a decimated mesh with a corresponding bijective map between the input and the decimated model. For the sake of reproducibility, we reiterate the core ideas from [Lee et al., 1998; Cohen et al., 1997, 1998], and describe how to combine both ideas.

We denote the input triangle mesh as $\mathcal{M}^L = (\mathsf{V}^L, \mathsf{F}^L)$, where $\mathsf{V}^L, \mathsf{F}^L$ are vertex positions and face information respectively at the original level L. The input mesh $\mathcal{M}^L$ is successively simplified into a series of meshes $\mathcal{M}^l = (\mathsf{V}^l, \mathsf{F}^l)$ with $0 \leq l \leq L$, where $\mathcal{M}^0 = (\mathsf{V}^0, \mathsf{F}^0)$ is the coarsest mesh. For each edge collapse $\mathcal{M}^l \to \mathcal{M}^{l-1}$, we compute the bijective map $f_l^{l-1} : \mathcal{M}^{l-1} \to \mathcal{M}^l$ (see Fig. 4.12) on the fly. The final map $f_L^0 : \mathcal{M}^0 \to \mathcal{M}^L$ is computed via composition,

$$f_L^0 = f_L^{L-1} \circ \cdots \circ f_1^0. \tag{4.1}$$

We now focus our discussion on the computation of a bijective map for a single edge collapse.

4.4.2 Single Edge Collapse

In each edge collapse, the triangulation remains the same, except for the neighborhood of the collapsed edge. Let $\mathcal{N}(i)$ be the neighboring vertices of a vertex i and let $\mathcal{N}(j,k) = \mathcal{N}(j) \cup \mathcal{N}(k)$ denote the neighboring vertices of an edge (j,k). After each collapse, the algorithm computes the bijective map for the edge's 1-ring $\mathcal{N}(j,k)$, in two stages. It first parameterizes the neighborhood $\mathcal{N}(j,k)$ (prior to the collapse) into 2D. It then performs the edge collapse both on the 3D mesh, and in UV space, as depicted in Fig. 4.13. The key observation from [Cohen et al., 1997, 1998] is that the boundary vertices of $\mathcal{N}(j,k)$ before the collapse become the boundary vertices of $\mathcal{N}(i)$ after the collapse. Hence the UV parameterization of the 1-ring remains injective after the collapse. Then, for any given point $p^{l-1} \in \mathcal{M}^{l-1}$ (represented in barycentric coordinates), we can utilize the shared UV parameterization to map p^{l-1} to its corresponding barycentric point $p^l \in \mathcal{M}^l$ and vice-versa, as shown in Fig. 4.14.

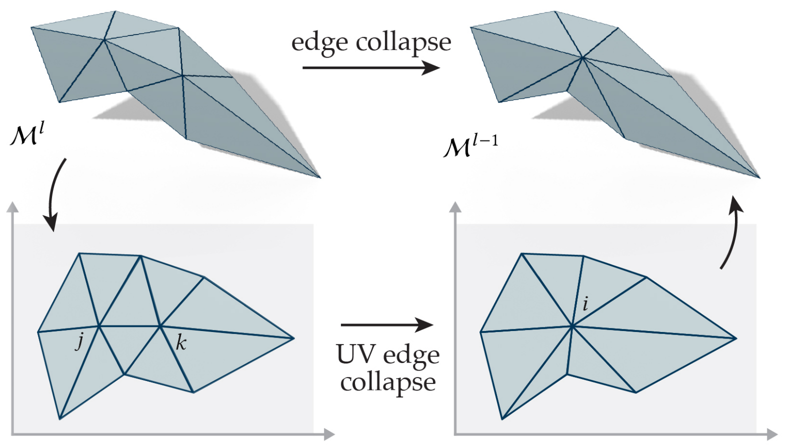


Figure 4.13: For each edge collapse, we simultaneously collapse the edge on the 3D mesh (top) and the UV domain (bottom). As the boundary vertices of the edge's 1-ring are preserved through the edge collapse, we constrain the flattened boundary in UV space to be at the same position when computing an as-conformal-as-possible parameterization of the post-collapse 1-ring.

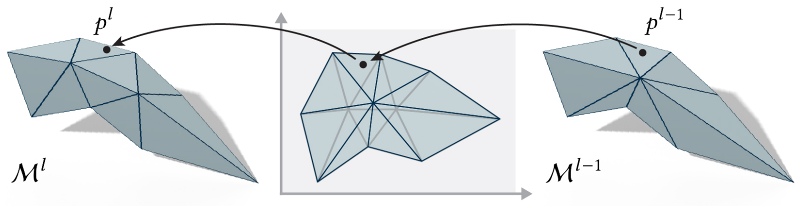


Figure 4.14: Since both the pre-collapse and post-collapse parameterizations of the 1-ring map it into the same 2D domain, we can easily use the shared UV space to map a point back and forth between $\mathcal{M}^l$ and $\mathcal{M}^{l-1}$.

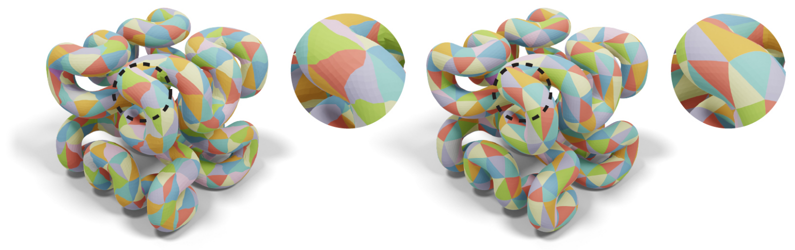


Figure 4.15: Using a different parameterization technique that does not result in a conformal flattening leads to a distorted parameterization (left), in contrast to the conformal parameterization we use, that reduces the amount of angle distortion accumulated throughout the edge collapse sequence (right). ©Hilbert Cube by tbuser under CC BY-SA.

Figure 4.16: Checking the criteria of collapsible edges is crucial for the robustness of the successive self-parameterization. From left to right, ©Psycho by Aeva (2nd, CC BY-SA), Parametric Sculpture by MCompeau (4th, CC BY-NC), Deer Head by TakeshiMurata (5th, CC BY-SA), Brain Slug by Zarquon (6th, CC BY-NC-SA), Spiral Light Bulb by benglish (7th, CC BY-SA), and Metratron by addy (9th, GNU).

Following the idea of MAPS [Lee et al., 1998], we use conformal flattening [Mullen et al., 2008] to compute the UV parameterization of the 1-rings, Fig. 4.13. After collapsing an edge and inserting the new vertex $\mathrm{v} \in \mathbb{R}^3$, we determine the UV location of the inserted vertex by performing another conformal flattening of its 1-ring patch with the boundary vertices fixed to the UV locations before the collapse. The conformality of the map is crucial, as it minimizes angle distortion which would otherwise accumulate throughout the successive parameterizations, leading to distorted, skewed correspondences and hindering learning of the network (see Fig. 4.15).

4.4.3 Implementation

Successive self-parameterization can be used with any edge collapse algorithm simply by adding two additional steps (see App. 4.8.2). The actual edge collapse algorithm, such as QSLIM [Garland and Heckbert, 1997], takes $\mathcal{O}(N \log N)$ time, and the flattening is a constant cost on top of each collapse (assuming valence is bounded). Thus the complexity of the entire algorithm containing both edge collapses and successive self-parameterization is still $\mathcal{O}(N \log N)$.

The robustness of the parameterization algorithm relies heavily on the robustness of the underlying edge collapse algorithm. Edge collapses that may lead to self-intersections can result in unusable maps. In App. **??**, we summarize our criteria for checking the validity of an edge collapse. This is crucial to ensure that we can generate training data using a wide range of shapes (see Fig. 4.16).

4.4.4 Training Data & Loss Computation

Our training data is constructed by applying the successive self-parameterization on top of random edge collapses. In Fig. 4.9, given a high-resolution shape (green), we use QSLIM [Garland and Heckbert, 1997] with a random sequence of edge collapses to construct

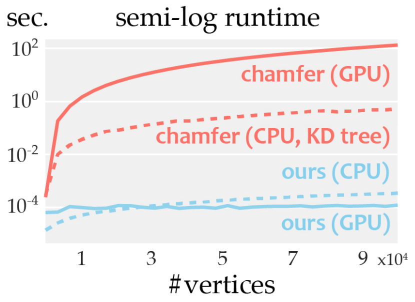


Figure 4.17: Our loss computation is orders of magnitude faster than the chamfer loss on the GPU (Kaolin [J. et al., 2019]) or the CPU (our KD-tree-based implementation).

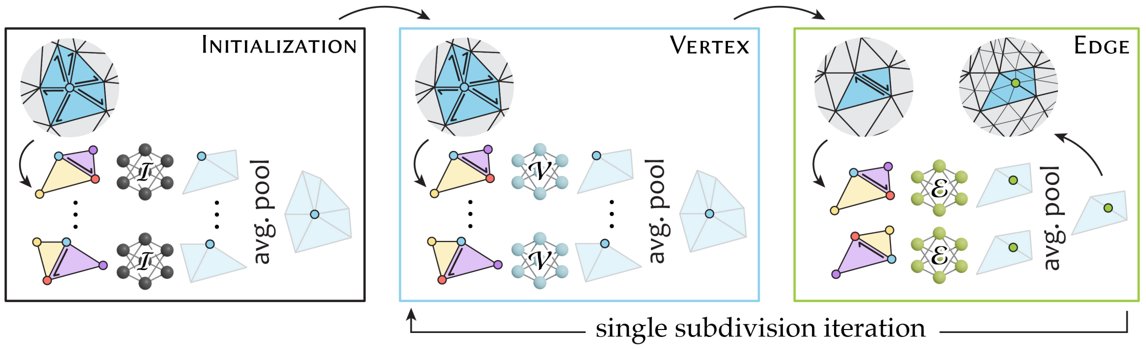


Figure 4.18: Our neural subdivision consists of three sequential steps: Initialization , Vertex , and Edge , with three network modules: $\mathcal{I}$, $\mathcal{V}$, and $\mathcal{E}$ for each step respectively. In both Initialization and Vertex steps, we apply $\mathcal{V}$ and $\mathcal{E}$ for the half-flaps of all the outgoing edges of a vertex, and use average pooling to combine the output features back to the center vertex (blue). In the Edge step, we apply $\mathcal{E}$ to both half-flaps of an undirected edge and use average pooling to map the output features to the center vertex (green) of the edge.

several different decimated models (gray). During the collapse, we plug in our parameterization to obtain a high-quality bijective map for each coarse and fine pair.

After the network subdivides the coarse mesh, we use the map to retrieve one-to-one correspondences to the input shape. Specifically, when retrieving the correspondences, we use the Loop topology update to add points in the middle of each edge, e.g., the point with barycentric coordinates $(0.5, 0.5, 0)$ in a triangle of the coarse mesh. We use the barycentric coordinates b on the coarse mesh to obtain the barycentric coordinates $f(\mathsf{b})$ on the fine mesh, as illustrated in Fig. 4.14 using the bijective map f. During training, suppose $\mathcal{E}(\mathsf{b})$ is the vertex position output by the network $\mathcal{E}$. We measure the per-vertex loss with the ℓ^2 distance $\|f(\mathsf{b}) - \mathcal{E}(\mathsf{b})\|_2$. Compared to the chamfer distance [Barrow et al., 1977], a widely used distance in training 3D generative models [Fan et al., 2017], our loss computation is orders of magnitude faster (see Fig. 4.17).

4.5 Network Architecture

Given a mesh at a previous level of subdivision along with a known topological update rule (mid-point subdivision as used by Loop), our neural network computes all vertex coordinates for the subdivided mesh. Our process involves three main steps illustrated in Fig. 4.18. The Initialization step uses a learnable neural module $\mathcal{I}$ to map input per-vertex features to high-dimensional feature vector at each vertex. In each subdivision iteration, the Vertex step uses a learnable module $\mathcal{V}$ to update features at corners of triangles of the input mesh, and the Edge step uses a learnable module $\mathcal{E}$ to compute features of vertices that were generated at mid-points of edges of the input mesh. Our network is inspired by classical subdivision algorithms which have two sets of rules: to update (1) *even* vertices from previous iterations, and (2) the newly inserted *odd* vertices. One difference of our approach is that we apply $\mathcal{V}$ and $\mathcal{E}$ in sequence, instead of in parallel. This allows us to harness neighborhood information from previous steps.

We make several design choices that are critical to the ability of our network to generalize well even from very small amounts of training data. First, even though all mesh update steps are global (i.e., they affect every vertex of the mesh), our learnable modules that are used in these steps operate over local mesh patches and share weights. Thus, even a single training pair provides many local mesh patches to train our neural modules. Second, our modules operate over original discrete elements of the mesh, and do not require re-parameterizing or re-sampling the surface. Representing input and output using the mesh discretization enables us to preserve the topology of the input, and generalize to novel meshes with different topology. Third, we represent our vertices using differential quantities with respect to a local coordinate frame instead of using global coordinates. Thus our neural modules operate over a representation that is invariant to rigid motion which simplifies training and improves their ability to generalize.

The key component of our neural module is a learnable operator that takes *half-flap*, a 2-face flap adjacent to a half-edge, inspired by the edge convolution approach of Hanocka et al. [2019]. We choose to use half-flap (instead of a flap around an undirected edge) since it provides a unique canonical orientation for the four vertices at the corners of adjacent faces. It also provides a well-defined local coordinate frame which we will use to define differential vertex quantities for the input and output (see the inset in the next page). Each flap operator is a shallow multi-layer perceptron (MLP) defined over features of four ordered points. We train one operator per module ($\mathcal{I}$, $\mathcal{V}$, $\mathcal{E}$) across all levels of subdivision and training examples.

Equipped with the half-flap operator, we use average pooling to aggregate features from different half-flaps to per-vertex features in all our neural subdivision steps. Initialization and Vertex steps apply the half-flap operator to every outgoing edge in a 1-ring neighborhood of a vertex, and average pooling aggregates per-half-flap outputs into a per-vertex feature. Edge step only considers per-vertex features at two endpoints of a subdivided edge

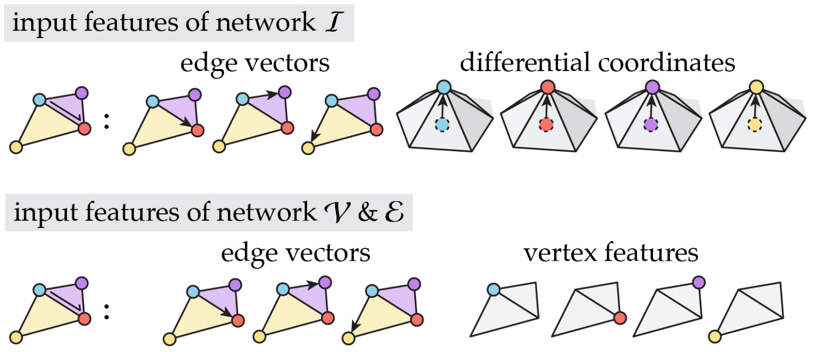
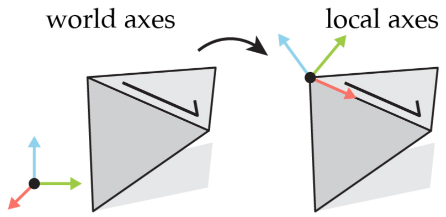

input features of network $\mathcal{I}$
edge vectors
differential coordinates
input features of network $\mathcal{V}$ & $\mathcal{E}$
edge vectors
vertex features

Figure 4.19: The input feature to module $\mathcal{I}$ consists of three edge vectors from the source vertex (blue) and vectors of the differential coordinates for the four vertices. The input features to module $\mathcal{V}$ and $\mathcal{E}$ are three edge vectors with per-vertex high-dimensional features from the previous steps.

to compute the feature of the inserted vertex. Thus, it simply applies the half-flap operator for each direction of the edge and again uses average pooling to get the vertex feature.



The final critical element of our architecture design is the representation for the input and output. As mentioned before, we use local differential quantities to ensure invariance to rigid transformation. The input features for the half-flaps used in Initialization step by module $\mathcal{I}$ consist of three edge vectors (originating at the source vertex of the half-flap) and differential coordinates of each vertex, as illustrated in Fig. 4.19, top. The vector of differential coordinates stores the discrete curvature information and is defined as the difference between the absolute coordinates of a vertex and the average of its immediate neighbors in the mesh [Sorkine, 2005]. To achieve rotation invariance we represent our differential quantities in the local frame of each half-flap (see the inset), where we treat the half-edge direction as the x-axis, the edge normal computed via averaging the two adjacent face normals as the z-axis, and the cross product of the previous two axes becomes our y-axis. The input to half-flap operators used in Vertex and Edge steps is similar (Fig. 4.19, bottom), where we use edge vectors and per-vertex high-dimensional learned features (either produced by Initialization step or by previous subdivision iteration). The output of half-flaps used in Vertex and Edge steps includes high-dimensional learned features and differential quantities that can be used to reconstruct the vertex position. For the latter we use the vertex displacement vector from the mid-point subdivided mesh (see Fig. 4.20) in the local coordinate system of the half-flap. For the Initialization and the Vertex networks, the predicted displacements live on the vertices; for the Edge network, the predicted displacements live on the edge midpoints. In our experiments, we notice there is no difference between predicting from the mid-point subdivided surface or other subdivision surfaces (see App. 8.9.4), so we choose mid-point subdivision for simplicity.

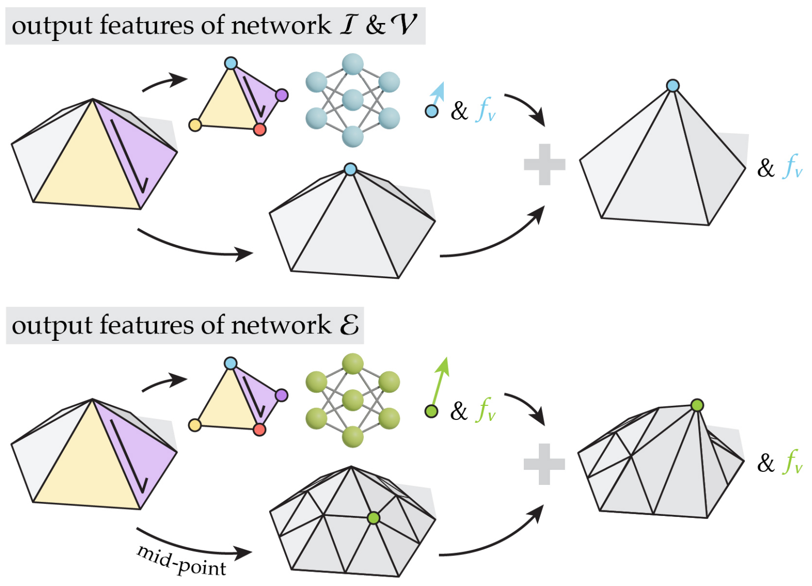

output features of network $\mathcal{I}$ & $\mathcal{V}$
& f_v
& f_v
output features of network $\mathcal{E}$
& f_v
& f_v
mid-point

Figure 4.20: The outputs of modules $\mathcal{I}$ and $\mathcal{V}$ are the displacement vector from the starting vertex and a learned feature vector f_v stored at the source vertex (blue). The outputs of the module $\mathcal{E}$ are the displacement from the edge mid-point (green) and the feature f_v stored at the mid-point.

We estimate global coordinates of vertices after each step to visualize intermediate levels of subdivision and compute the loss function, and convert global coordinates to local differential per-vertex quantities before each step to ensure that each network only observes translation- and rotation- invariant representations.

Fig. 4.21 illustrates that an invariant representation is critical to the quality of results. We demonstrate that even when trained on an identical true shape, a slight rigid motion of that shape renders learned weights completely inapplicable at inference time. We also observe that incorporating the differential coordinates as part of the input features makes the training converge faster (see App. 8.9.4). Thanks to our local half-flap operators and invariant representations we can train our architecture even with shallow 2-layer MLPs (see

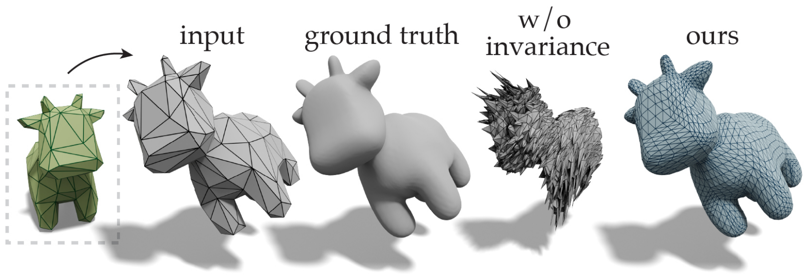


Figure 4.21: We use differential quantities stored in the local frames as our inputs and outputs. This design makes our network invariant to rigid motions and significantly boosts the quality compared to an approach without invariance.

Table 4.1: Hyperparameters of our sub-networks. All networks are fully-connected multi-layer perceptrons with two hidden layers.

	network $\mathcal{I}$	*network* $\mathcal{V}$	*network* $\mathcal{E}$
f_{in}	$3 \cdot 3 + 4 \cdot 3$	$3 \cdot 3 + 4 \cdot 32$	$3 \cdot 3 + 4 \cdot 32$
fc_1	32	32	32
fc_2	32	32	32
f_{out}	$3 + 29$	$3 + 29$	$3 + 29$

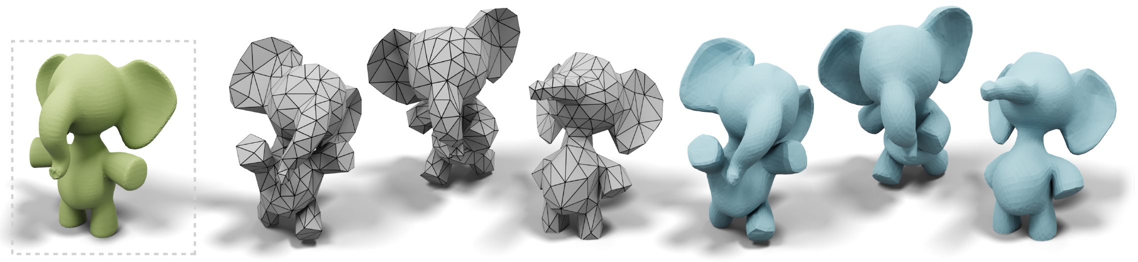


Figure 4.22: We train our network on a single pose (green) and the network is able to generalize to unseen poses (blue).

Table 4.1 for network hyper-parameters). We further evaluate other design decisions and conclude that details such as whether to predict displacements from the mid-point or the Loop subdivision, whether to recursively apply the module $\mathcal{V}$, whether to measure loss across all levels, and whether to use input features proposed in [Hanocka et al., 2019] offer small improvements to the convergence (see App. 8.9.4 for details).

We implemented our network in PyTorch [Paszke et al., 2019]. We use ReLu activation [Nair and Hinton, 2010], and the ADAM optimizer [Kingma and Ba, 2015] with learning rate 0.002.

4.6 Evaluations

We evaluate our neural subdivision with a range of results of increasing complexity. We start by showing that we can generalize to isometric deformations, non-isometric deformations, shapes from different classes, and shapes from different types of discretizations. We summarize the details of our experiments in App. 4.8.6.

In practice, modelers often manipulate the coarse subdivision cage of a character into different poses, and then apply the subdivision operator. This scenario implies that being able to train on one single pose and generalize to unseen poses is important for character animation. In Fig. 4.22, we train on a single pose (in green) and show that our network can generalize to unseen poses under (approximately) isometric deformations.

In addition to poses, in Fig. 4.23 we mimic the real scenario by manually changing the coarse cage and show that the learned subdivision can also generalize to non-isometric

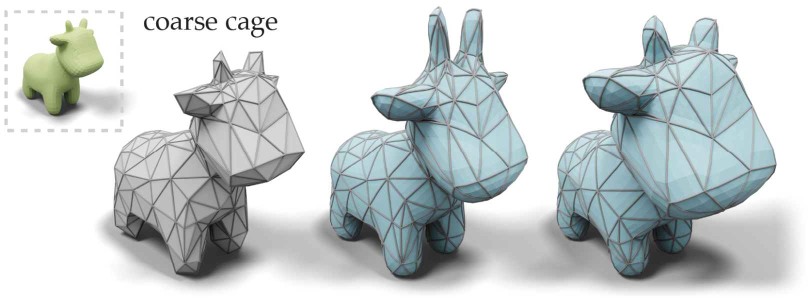


Figure 4.23: We mimic the modeling scenario by applying non-isometric deformations to the coarse cage (gray). Our subdivision network is able to generalize to unseen non-isometric deformations.

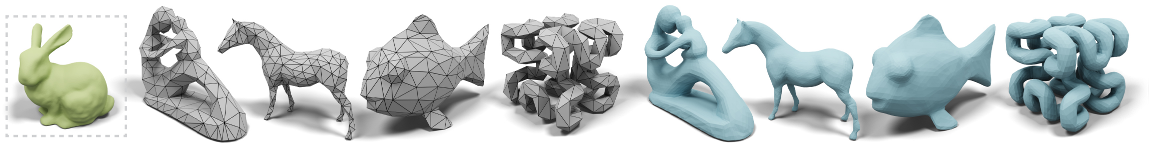


Figure 4.24: Even when trained on only a single shape (green bunny), our network can generalize to subdividing different geometries (blue). ©Hilbert Cube by tbuser (right) under CC BY-NC.

deformations.

Subdivision operators are often used to create novel 3D content, which implies the importance of generalizing to totally different shapes. In Fig. 4.24 we show that even when trained on only a single shape (green), our network is able to generalize to many other shapes (blue). We also show that our network trained on classic Loop subdivision sequences is able to reproduce Loop subdivision on unseen shapes (App. 4.8.7).

We further evaluate neural subdivision on shape discretizations created in a totally independent way. In Fig. 4.25 we obtain coarse shapes created by artists, instead of from edge collapses, and show that neural subdivision can still generalize well.

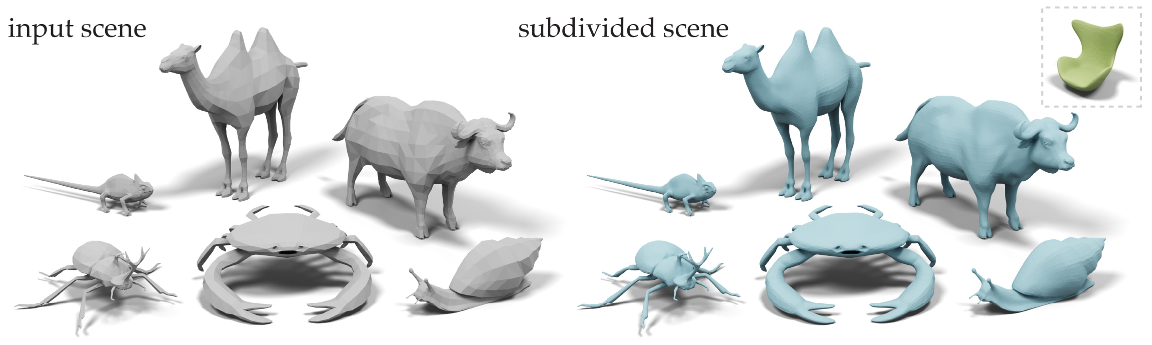


Figure 4.25: In addition to subdividing meshes constructed via decimation, our network can also generalize to subdivide meshes created by artists.

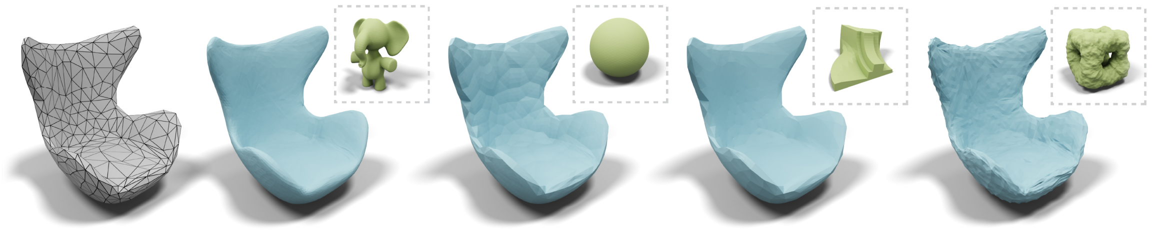


Figure 4.26: Using different shapes in training leads to stylized subdivision results (blue) biased towards the training shapes (green). ©Egg Chair by TeamTeamUSA (left) under CC BY.

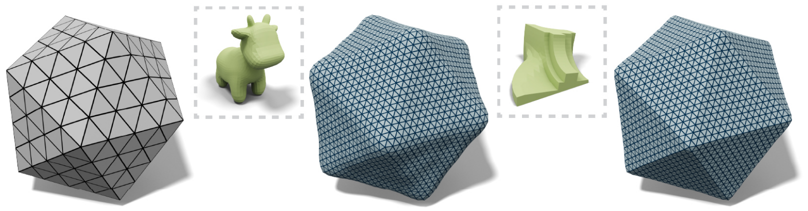


Figure 4.27: Training on a smooth shape leads to a smoother subdivision result (middle). Training on a man-made object can preserve the sharp creases (right).

The ability to generalize even when trained on a single shape gives us the opportunity to do stylized subdivisions. In Fig. 4.26 our neural subdivision operators are aware of the "style" of the training shape and are able to create different results from the same coarse geometry. In Fig. 4.27, we show different results when trained on a smooth organic shape vs a man-made object with sharp contours.

To quantitatively analyze how our network generalizes to unseen shapes, we take the TOSCA dataset [Bronstein et al., 2009] which contains 80 shapes with 9 categories to perform quantitative analysis. For the top table of Table 4.2, we train on a single category (*Centaur*) and test on the remaining categories. Our test shapes are generated by coarsening source meshes with QSLIM down to 350-450 vertices. We measure the error between the two-level subdivided mesh and the original shape using Hausdorff distance, as well as mean surface distance computed by METRO [Cignoni et al., 1998b]. Our method consistently produces smaller errors compared to the classic Loop [Loop, 1987] and modified butterfly [Zorin et al., 1996] subdivisions.

We further evaluate our method when trained on multiple shapes and categories. In Fig. 4.28, we train the network on an increasing number of objects and observe that the results are visually similar. But our quantitative analysis in the bottom table of Table 4.2 shows that training on more categories (*Centaur, David, Horse*) can slightly reduce the error.

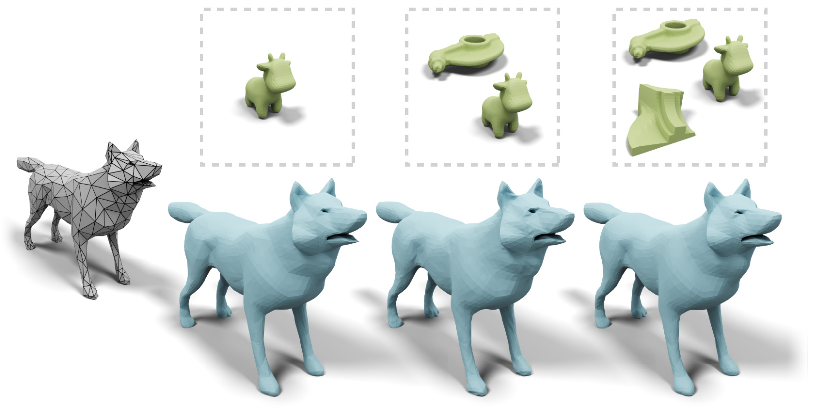


Figure 4.28: We train our subdivision network on a mixture of organic and non-organic shapes. We observe that training on more objects does not significantly change visual quality.

Table 4.2: We train on a single category, *Centaur* (top table), and three categories, *Centaur, David, Horse* (bottom table), separately, and evaluate by subdividing the rest of the TOSCA shapes. The results indicate that neural subdivision outperforms classic Loop subdivision [Loop, 1987] and modified butterfly subdivision [Zorin et al., 1996] on two popular metrics: Hausdorff distance $\mathbb{H}$, and mean surface distance $\mathbb{M}$ computed via METRO [Cignoni et al., 1998b].

Category	$\mathbb{H}_{\text{loop}}$	$\mathbb{H}_{\text{m.b.}}$	$\mathbb{H}_{\text{ours}}$	$\mathbb{M}_{\text{loop}}$	$\mathbb{M}_{\text{m.b.}}$	$\mathbb{M}_{\text{ours}}$
Cat	2.75	2.17	**2.08**	0.73	0.21	**0.17**
David	2.95	2.13	**1.83**	0.88	0.27	**0.20**
Dog	3.26	2.32	**2.11**	0.84	0.31	**0.26**
Gorilla	4.53	3.17	**2.56**	1.27	0.48	**0.36**
Horse	5.87	4.53	**4.04**	1.51	0.50	**0.45**
Michael	3.88	2.71	**2.24**	1.12	0.38	**0.28**
Victoria	4.25	3.01	**2.36**	1.12	0.39	**0.30**
Wolf	2.83	1.74	**1.63**	0.69	0.23	**0.21**

Category	$\mathbb{H}_{\text{loop}}$	$\mathbb{H}_{\text{m.b.}}$	$\mathbb{H}_{\text{ours}}$	$\mathbb{M}_{\text{loop}}$	$\mathbb{M}_{\text{m.b.}}$	$\mathbb{M}_{\text{ours}}$
Cat	2.75	2.17	**2.09**	0.73	0.21	**0.16**
Dog	3.26	2.32	**2.12**	0.84	0.31	**0.25**
Gorilla	4.53	3.17	**2.89**	1.27	0.48	**0.34**
Michael	3.88	2.71	**2.15**	1.12	0.38	**0.27**
Victoria	4.25	3.01	**2.49**	1.12	0.39	**0.28**
Wolf	2.83	1.74	**1.65**	0.69	0.23	**0.20**

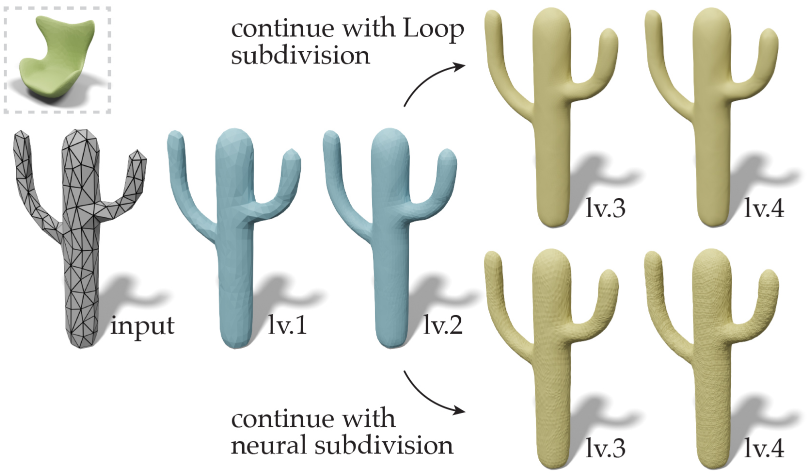

continue with Loop subdivision
input
lv.1
lv.2
lv.3
lv.4
continue with neural subdivision
lv.3
lv.4

Figure 4.29: Since our method induces a non-linear subdivision, there is no guarantee of the existence of a limit surface (bottom). An alternative is to apply neural subdivision at the trained levels, and continue with classic subdivision (top) to ensure a smooth limit surface.

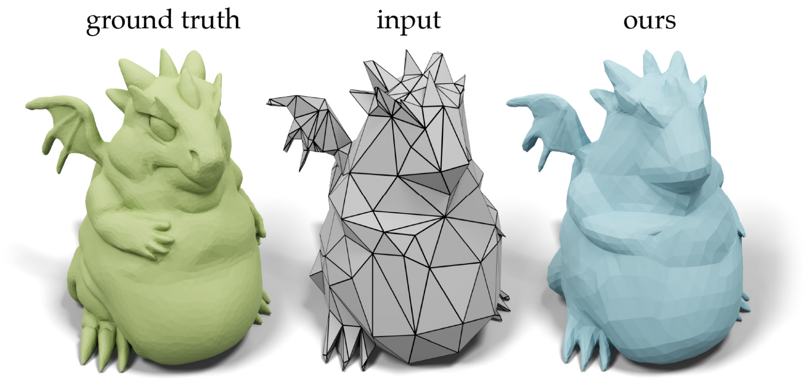


Figure 4.30: Our approach is based on local geometry, and thus fails to hallucinate semantic features. ©Bratty Dragon by Splotchy Ink under CC BY.

4.7 Limitations & Future Work

Extending the neural subdivision framework to quadrilateral meshes and surface with boundaries would be closer to real-world modeling scenarios. Making neural subdivision scale-invariant and converge to a limit surface (see Fig. 4.29) are also desirable in practice. Incorporating global information in the training could help the network hallucinate semantic features (see Fig. 4.30). Applying architectures (e.g., Recurrent Neural Net) that are more suitable for sequence predictions could help the network to harness information from a wider neighborhood and to dive to a deeper subdivision level. Training on data that contain a wide range of triangle aspect ratios and curvature information could further improve the robustness of the network. Since our data-generation algorithm is extremely efficient, it could be naturally used in an online-learning setting, where our algorithm constantly draws new randomly-coarsened meshes on-the-fly. This can be extremely useful in, e.g., a GAN setting. As a first step towards neural subdivision, we showed reconstruction of fine meshes from coarse ones. Fully-fledged super-resolution, detail hallucination, and surface stylization are interesting next steps. All of these questions provide interesting topics for the future research on neural subdivision.

4.8 Appendix

4.8.1 Implementation of Point Cloud Upsampling

An alternative way to upsample a mesh is to first convert the mesh into a point cloud via sampling over the surface, run point cloud upsampling algorithms, and then perform a surface reconstruction to convert the upsampled point cloud back to a mesh. However, this procedure is expensive to incorporate into the interactive graphics pipeline, fails to produce surfaces with different levels of detail (see Fig. 4.2), and fails to preserve textures (see Fig. 4.3). In addition, many non-trivial design decisions such as the number of samples to use and how to sample the surface would influence the quality of the results. For example in Fig. 4.5, we first sample 5000 points with uniform and farthest point sampling, followed by the method of Wang et al. [2019b] pre-trained on statues to upsample the point cloud by 16$\times$, and then use the screened poisson reconstruction [Kazhdan and Hoppe, 2013] to reconstruct the surface. In the figure we show that different sampling methods lead to different results. The lack of connectivity information also results in some surface artifacts.

4.8.2 Implementation of Successive Self-Parameterization

Incorporating successive self-parameterization only requires adding two additional local conformal parameterizations to the edge collapse algorithm of choice. Suppose we want to collapse an edge (j,k). We first flatten the edge's 1-ring $\mathcal{N}(j,k)$ by minimizing a conformal energy [Lévy et al., 2002], then we collapse the edge, then we perform another conformal flattening on the 1-ring $\mathcal{N}(i)$ of the newly inserted vertex i after the collapse, with the boundary held in place from the previous flattening. This yields a bijective map with small computational cost because each flattening only involves a 1-ring (assuming the vertex valence is bounded).

4.8.3 Criteria for Collapsible Edges

During edge collapses, many issues such as flipped faces and non-manifold edges may appear. Resolving these issues is crucial to the robustness of successive self-parameterization (see Fig. 4.16). We summarize our criteria for checking the validity of an edge collapse. If invalid, we simply avoid collapsing the edge at that iteration.

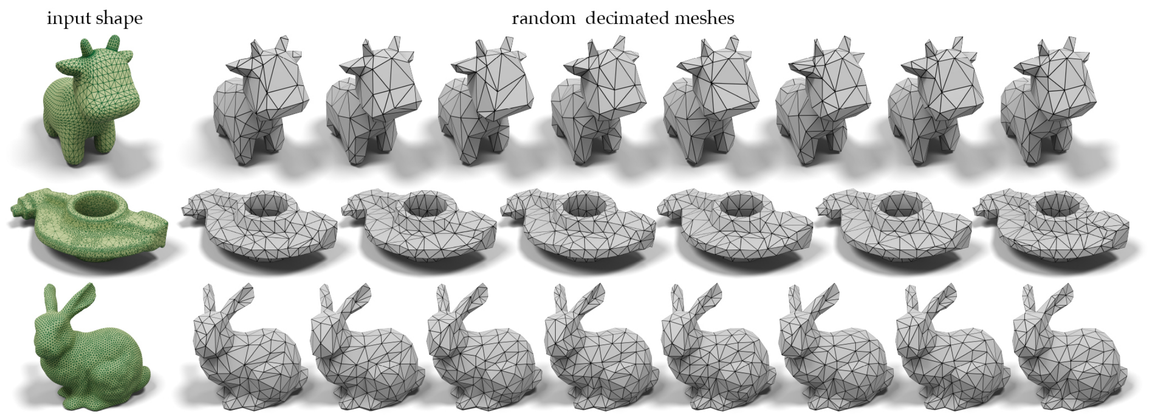
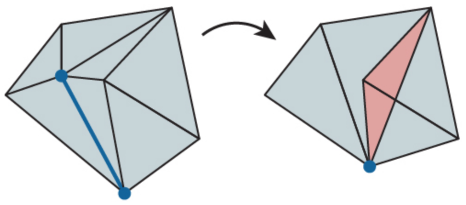


Figure 4.31: We perform QSLIM with a random sequence of edge collapses to create different coarse discretizations (gray) from a single ground truth mesh (green).

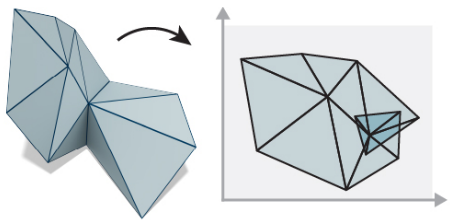


Euclidean face flips Certain faces in the Euclidean space may suffer from normal flips after an edge collapse. To prevent flipped faces, we simply compare the unit face normal n of each neighboring face f_i before and after the collapse

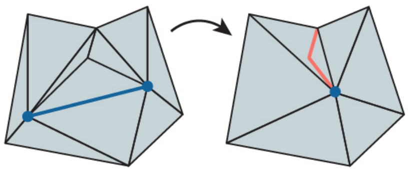
$$\mathsf{n}_{f_i}^{\text{before}} \cdot \mathsf{n}_{f_i}^{\text{after}} > \delta. \tag{4.2}$$

We set $\delta = 0.2$ as default to avoid face flips in all our experiments.

UV face flips Flipped faces may also appear in the UV space due to both the conformal flattening and the edge collapse. We simply check whether the signed area of each UV face is positive before and after collapses to prevent having UV face flips.



Overlapped UV faces Even if all the UV faces are oriented correctly, some of the faces may still overlap with each other depending on the flattening algorithm in use. We check whether the total angle sum of each interior vertex is 2π to determine the validity of a collapse.



Non-manifold edges To prevent the appearance of non-manifold edges, we must check the *link condition* [Dey et al., 1999; Hoppe et al., 1993]. Briefly, the link condition says that if an edge e_{ij} connecting vertices i, j is valid, the intersection between the vertex 1-ring of i and the vertex 1-ring of j must contain only two vertices, and the two vertices cannot be an edge.

Skinny triangles To prevent badly shaped triangles from causing numerical issues, we need to keep track of the triangle quality for each edge collapse. The quality of a triangle is measured by

$$Q_{ijk} = \frac{4\sqrt{3}\, A_{ijk}}{l_{ij}^2 + l_{jk}^2 + l_{ki}^2} \tag{4.3}$$

where A is the area of the triangle and l are the lengths of triangle edges. When $Q \to 1$, the triangle approaches an equilateral triangle; when $Q \to 0$, it approaches a skinny degenerated one. For each edge, we check Q for all the neighboring faces in both UV and Euclidean domains after the collapse. By default, a valid edge requires $Q > 0.2$ for all neighboring triangles.

4.8.4 Comparison to [Lee et al., 1998]

One possible solution is to construct a bijective map between the input and the decimated model via MAPS [Lee et al., 1998]. However, MAPS constructs the parameterization via successively removing the maximum vertex independent sets. The main reason for removing the maximum independent set is to bound the number of levels of the mesh hierarchy, but it leads to limitations such as sensitivity to the input triangulation.

One experiment to verify this is to apply subdivision remeshing presented in Sec. 4.1 in [Lee et al., 1998]. In Fig. 4.32 we create a stress test using a very uneven triangulation, and MAPS suffers from creating non-uniform parameterization. In contrast our successive self-parameterization enjoys the benefits of area-weighted QSLIM to obtain a more uniform parameterization.

4.8.5 Data Generation from Random Collapses

The training data for neural subdivision is a sequence of subdivided meshes where the vertex positions are computed using successive self-parameterization (Fig. 4.9). For each dense input mesh, we perform semi-random edge collapses in order to generate many different coarse meshes. The goal is to help the network being robust to different discretizations. In Fig. 4.31 we show input meshes (left) can be decimated differently to get many coarse meshes that have different numbers of vertices and triangulations.

Our semi-random edge collapse starts by randomly selecting 100 edges and finding the one with the minimum *quadric error* [Garland and Heckbert, 1997] to collapse. For each edge collapse, we insert the new vertex the same way as QSLIM does. We terminate the edge collapses when a randomly selected target number of vertices between 150 and 300 is reached.

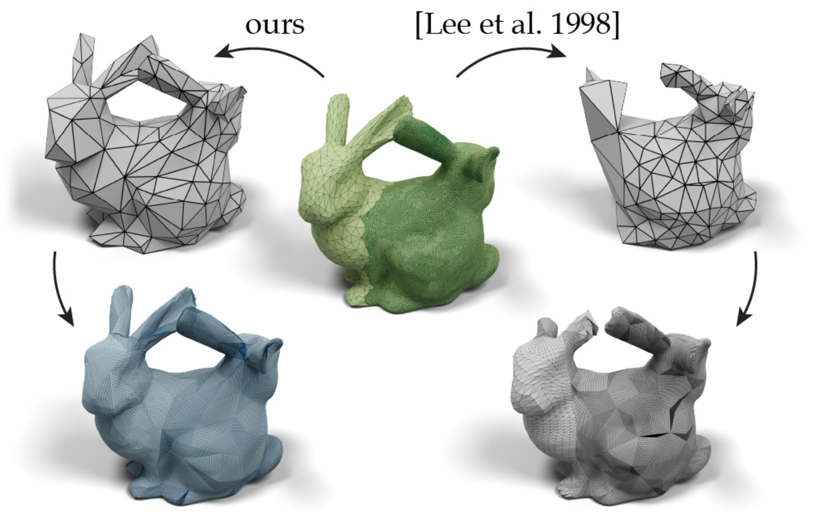


Figure 4.32: We decimate the mesh down to the same number of vertices and compare our method with MAPS on the task of subdivision remeshing. Our method creates a more uniform parameterization (left), but MAPS is more sensitive to the input triangulation (right).

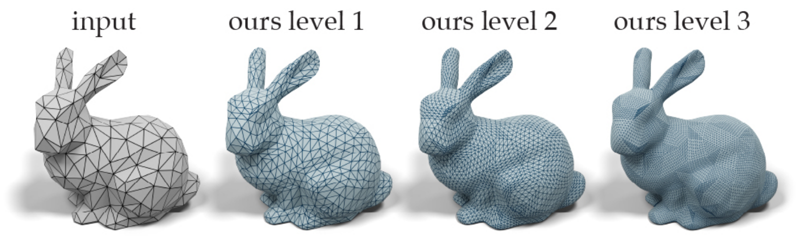


Figure 4.33: Although most experiments are trained on performing 2-level subdivisions, our neural subdivision network can still be trained on more level of the subdivisions.

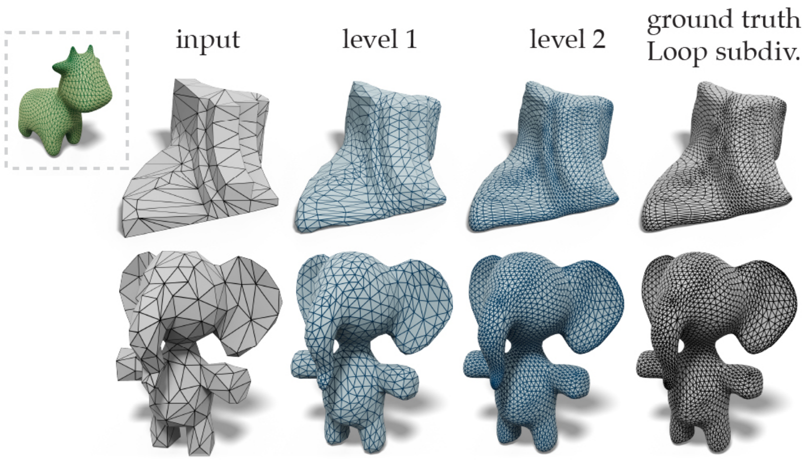


Figure 4.34: When trained on meshes created by classic Loop subdivision (green), our network can reproduce the Loop scheme on new meshes, and creates visually indistinguishable results (blue) compared to the ground truth created by the classic Loop method (right).

4.8.6 Experimental Setup

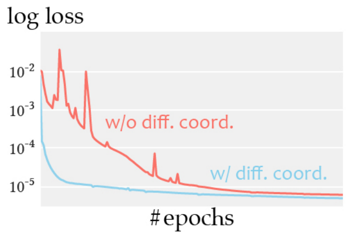
Our experimental setup is consistent throughout the document. The training shape is presented in green in every figure. For each shape, we use the parameters described in App. 4.8.5 to generate 200 training discretizations and train for 700 epochs. Our method can learn to produce several subdivision levels Fig. 4.33, but we set the number of training subdivisions to two levels for consistency across the experiments. If the experiment consists of multiple training shapes, such as the experiments in Fig. 4.28 and Table 4.2, we evenly distribute the number of training discretizations so that they still sum up to 200 discretizations in total.

4.8.7 Learning Classic Loop Subdivision

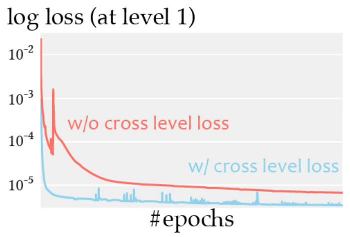
Although we have shown in Sec. 4.6 that neural subdivision is able to subdivide a mesh adaptively, one might be interested in seeing whether neural subdivision can also learn to reproduce classic Loop subdivision with appropriate training data. In Fig. 4.34, we trained our network on a sequence of meshes created with Loop subdivision. Given an original mesh, we create 200 meshes using random edge collapses, then refined each coarsened mesh for two levels using Loop subdivision to obtain the corresponding ground truth subdivided sequences for measuring the reconstruction loss. We see that when testing on novel meshes, the network is able to reproduce the Loop scheme to create visually indistinguishable results. The average per-vertex numerical error is just 0.3% of the bounding box diagonal.

4.9 Ablation Studies (continued)

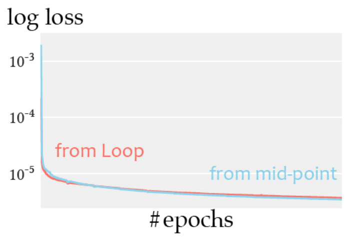
This section summarizes the ablation studies of other design decisions we made in the network design. These components are not as crucial as the components mentioned in the main text, but they still offer improvements while training. The first analysis is the influence of adding differential coordinates in the input (see Fig. 4.19). Our result in the inset indicates that adding differential coordinates can improve convergence.



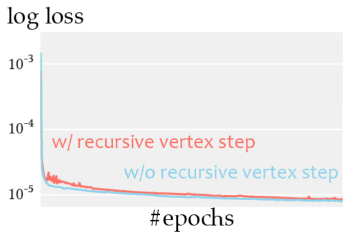
We also measure the effect of adding cross-level loss compared to only measuring the loss at the final level. In the inset, we visualize the error in the intermediate level. The result suggests that adding cross-level loss can improve subdivision results in the intermediate levels, which is important for creating meshes with different levels of detail (see Fig. 4.2).



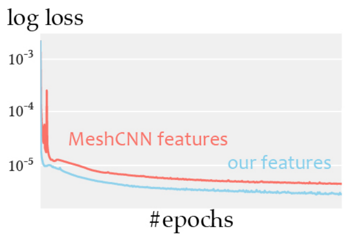
The third study analyzes the effects of the starting position of the predicted displacement vector as shown in Fig. 4.20. Specifically, we compare predicting the displacement from the mid-point of an edge with predicting displacement from the Loop-subdivided mesh. Our result in the inset suggests that using different starting positions has no influence to the quality of the output. Thus we choose the mid-point for simplicity.



The fourth study analyzes the effects of the number of vertex steps to perform. In Fig. 4.18, we can actually recursively perform the vertex step to gather information from larger rings. However our experiments in the inset indicates that recursively performing the vertex step does not offer improvements. Thus we only perform the vertex step once. We suspect that the 2-ring information on the coarse mesh (one from initialization, one from the vertex step) may already be sufficient for the network to perform subdivisions.



In MeshCNN, Hanocka et al. [2019] propose a set of features to characterize an undirected edge (via features of a flap), including the dihedral angle, two inner angles, and two edge length ratios (see Sec. 3 in [Hanocka et al., 2019]). We tried their proposed features in our neural subdivision network. In the inset, we observe that using our features, edge vectors and the vectors of differential coordinates, converges to a better solution.



Chapter 5

Cubic Stylization

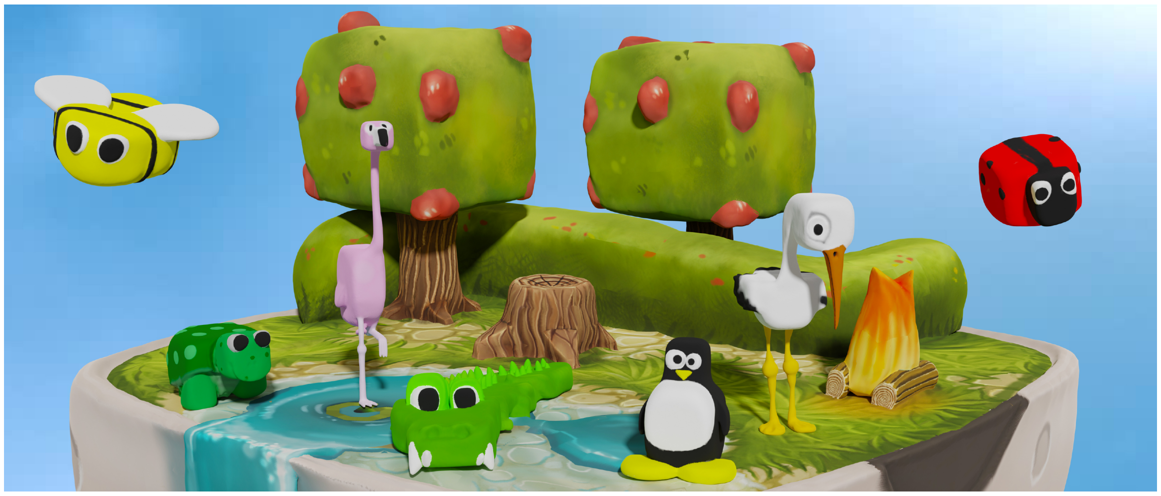


Figure 5.1: *Cubic stylization* deforms a given 3D shape into the style of a cube while maintaining textures and geometric features. This can be used as a non-realistic modeling tool for creating stylized 3D virtual world. We obtain 3D assets from `sketchfab.com` by smeerws and Jesús Orgaz under CC BY 4.0.

We present a 3D stylization algorithm that can turn an input shape into the style of a cube while maintaining the *content* of the original shape. The key insight is that cubic style sculptures can be captured by the *as-rigid-as-possible* energy with an ℓ^1-regularization on rotated surface normals. Minimizing this energy naturally leads to a detail-preserving, cubic geometry. Our optimization can be solved efficiently without any mesh surgery. Our method serves as a non-realistic modeling tool where one can incorporate many artistic controls to create stylized geometries.

5.1 Introduction

The availability of image stylization filters and *non-photorealistic rendering* techniques has dramatically lowered the barrier of creating artistic imagery to the point that even a non-professional user can easily create stylized images. In stark contrast, direct stylization of

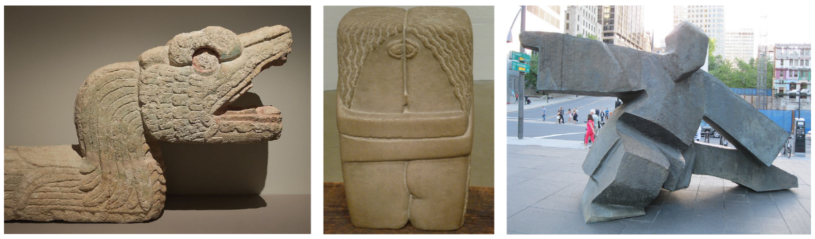


Figure 5.2: The cubic style have been attracting artists' attention over centuries, such as the *Serpend à' Plumes* found in Chichén Itzá (left), *The Kiss* by Constantin Brâncuși (middle), and the *Taichi* by Ju Ming (right). We obtain images from `wikimedia.com` photographed by Jebulon under CC0 1.0, from `flickr.com` by Art Poskanzer under CC BY 2.0, and from `wikimedia.com` by Jeangagnon under CC BY-SA 3.0.

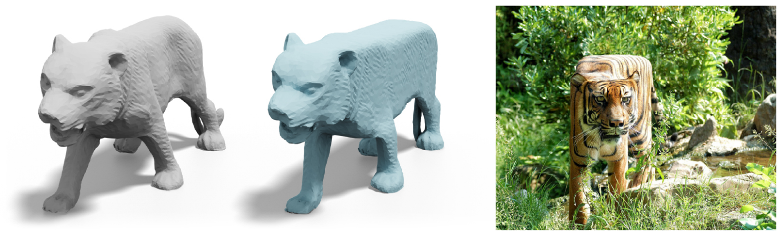


Figure 5.3: A digital art form – Anicube – by Aditya Aryanto produces cubic style images (right). Our method takes an input tiger (left) and outputs a "3D anicube" tiger while maintaining geometric details (middle). ©Aditya Aryanto (right). Used under permission.

3D shapes or *non-realistic modeling* has received far less attention. In professional industries such as visual effects and video games, trained modelers are still required to meticulously create non-realistic geometric assets. This is because investigating geometric styles is more challenging due to arbitrary topologies, curved metrics, and non-uniform discretization. The scarcity of tools to generate artistic geometry remains a major roadblock to the development of geometric stylization.

In this paper, we focus on the specific style of *cubic* sculptures. The cubic style is prevalent across art history, for instance the ancient sculptures from the post-classic era (900-1250 CE), Maya sculptures, block statues in Egypt, and modern abstract sculptures such as the ones from Constantin Brâncuși and Ju Ming (Fig. 5.2). In addition, the cubic style is a popular digital art form, such as the award-winning *Anicube* by Aditya Aryanto (Fig. 5.3). Complementing their presence in art, cubic shapes also present themselves in fabrication and furnitures (Fig. 5.4). We contribute to the rich history of cubic sculpting by providing a stylization tool that takes a 3D shape as input and outputs a deformed shape that has the same style as cubic sculptures.

We present *cubic stylization* which formulates the task as an energy optimization that naturally preserves geometric details while cubifying a shape. Our proposed energy combines an *as-rigid-as-possible* (ARAP) energy with an ℓ^1 regularization. This energy can be minimized efficiently using the local-global approach with *alternating direction method of*

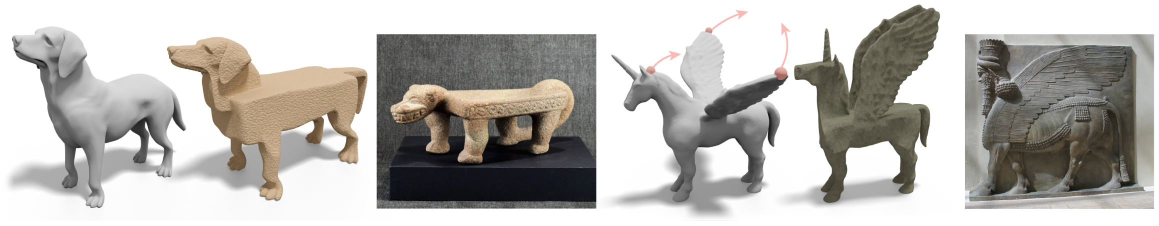


Figure 5.4: One can control the cubic stylization by incorporating constraints. For instance, we can fix some parts of a shape to mimic the style of a Jaguar metate from ancient Costa Rica (top) or add point constraints to mimic the Assyrian Lamassu wall sculpture (bottom). ©Antiques & Artifacts LLC (top). Used under permission.

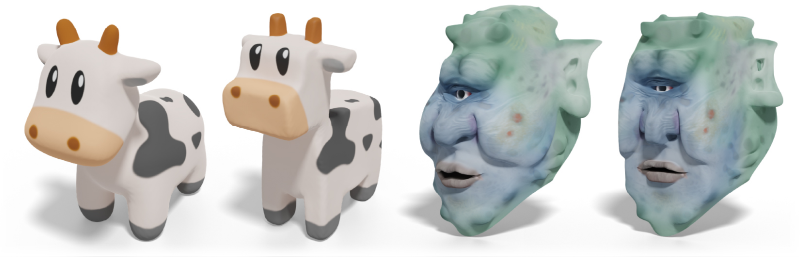


Figure 5.5: Our cubic stylization requires no remeshing, thus vertex attributes such as textures are preserved during the optimization. Our ARAP term encourges locally isometric deformations to help maintain nice textures.

multipliers (ADMM). This variational approach affords the flexibility of incorporating many artistic controls, such as applying constraints, non-uniform cubeness, and different global/local cube orientations (Sec. 5.4). Moreover, our method requires no remeshing (Fig. 5.5) and generalizes to polyhedral stylization (Fig. 5.24). Our proposed tool for non-realistic modeling goes beyond the 2D stylization and opens up the possibility of, for instance, creating non-realistic 3D worlds in virtual reality (Fig. 5.1).

5.2 Related Work

Our work shares similar motivations to a large body of work on image stylization [Kyprianidis et al., 2013], non-photorealistic rendering [Gooch and Gooch, 2001], and motion stylization [Hertzmann et al., 2009]. While their outputs are images or stylized animations, we take a 3D shape as input and output a stylized shape. Thus we focus our discussion on methods for processing geometry, including the study of geometric styles and deformation methods that share technical similarities.

Discriminative Geometric Styles The growing interest in understanding geometric styles has been inspiring recent works on building *discriminative* models for style analysis. One of the main challenges is to define a similarity metric aligned with human perception. Many

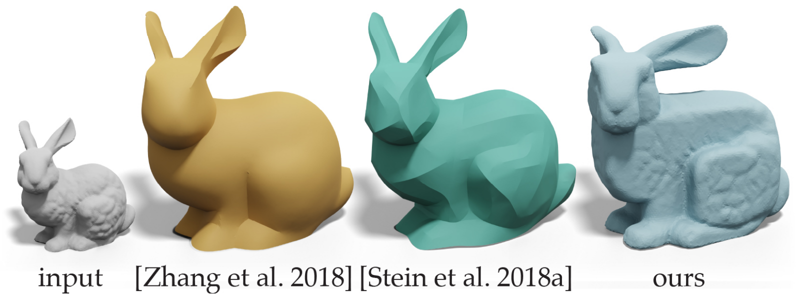


Figure 5.6: Our energy-based deformation shares similarities with many energy-based geometric flows and mesh filters, such as the methods of [Zhang et al., 2018] and [**?**].

works propose to compare projected feature curves [Li et al., 2013; Yu et al., 2018a], subcomponents of a shape [Xu et al., 2010; Lun et al., 2015; Hu et al., 2017], or using learned features [Lim et al., 2016]. These models enable users to synthesize style compatible scenes [Liu et al., 2015] or transfer style components across shapes [Ma et al., 2014; Lun et al., 2016; Berkiten et al., 2017]. However, these methods are designed for discerning and transfering styles, instead of generating 3D stylized shapes directly.

Generative Geometric Styles Direct 3D stylization has been an important topic in computer graphics. Many *generative* models have been proposed for producing specific styles, without relying on identifying and transferring style components from other shapes. This includes creating the collage art [Gal et al., 2007; Theobalt et al., 2007], voxel/lego art [Testuz et al., 2013; Luo et al., 2015], *neuronal homunculus* [Reinert et al., 2012], the manga style shapes [Shen et al., 2012], shape abstraction [Mehra et al., 2009; Kratt et al., 2014; Yumer and Kara, 2012], and *bas-relief* sculptures [Weyrich et al., 2007; Song et al., 2007; Kerber et al., 2009; Bian and Hu, 2011; Schüller et al., 2014]. While not pitched as stylization techniques, many geometric flows and filters can also be used for creating stylized geometry, such as creating edge-preserving smoothing geometry [Zhang et al., 2018], piece-wise planar [He and Schaefer, 2013; Stein et al., 2018b] or developable shapes [Stein et al., 2018a], and stylized shapes prescribed by image filters [Liu et al., 2018] (see Fig. 5.6). Our method contributes to the field of direct 3D stylization, focusing on the style of cubic sculptures (Fig. 5.7).

Shape Deformation Many works deal with the question of how to deform shapes given modeling constraints. One of the most popular choices is the ARAP energy [Igarashi et al., 2005a; Sorkine and Alexa, 2007; **?**; Chao et al., 2010], which measures local rigidity of the surface and leads to detail-preserving deformations. Not just deformations, similar formulations to ARAP can also be extended to other tasks such as constrained shape optimization [Bouaziz et al., 2012], parameterization [Liu et al., 2008], and simulating mass-spring systems [Liu et al., 2013a]. Ever since, optimizing the ARAP energy has been substantially ac-

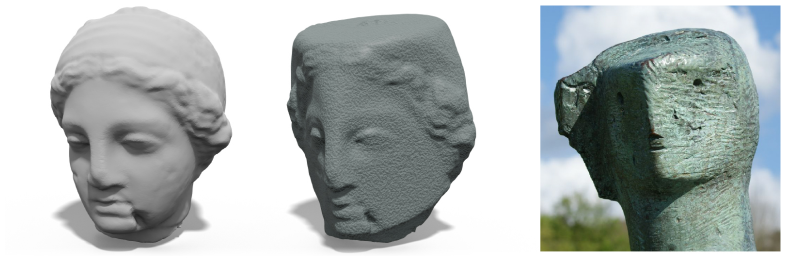


Figure 5.7: Cubic style sculptures are common throughout history, such as the *Draped Seated Woman* by Henry Moore (right). Our method offers an instrument to create cubic geometry (middle). The photo is taken by puffin11k under CC BY-SA 2.0.

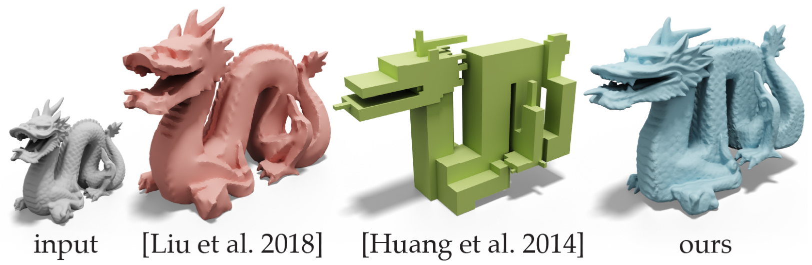


Figure 5.8: Paparazzi [Liu et al., 2018] with image quantization and polycube method (e.g., [Huang et al., 2014]) can create cubic style shapes (red, green), but unlike our method (blue) they do not preserve geometric details.

celerated by a large amount of work, such as [Kovalsky et al., 2016; Rabinovich et al., 2017; Shtengel et al., 2017; Peng et al., 2018; Zhu et al., 2018]. However, having nearly interactive performance on highly detailed meshes still remains a major challenge. An alternative strategy to speed it up is to use the hierarchical deformation which optimizes ARAP on a low resolution model and then recover the original details back afterwards [Manson and Schaefer, 2011]. This class of accelerations shares similar characteristics to multiresolution modeling (see [Garland, 1999; Zorin, 2006]). We take advantage of the ARAP energy for detail preservation and adapt the method of Manson and Schaefer [2011] to accelerate our cubic stylization to meshes with millions of faces.

Axis-Alignment in Polycube Maps Axis-alignment is an important property for many geometry processing tasks, such as [Muntoni et al., 2018; Stein et al., 2019]. Especially, this concept is one of the main instruments in the construction of polycube maps [Tarini et al., 2004], including defining polycube segmentations [Livesu et al., 2013; Fu et al., 2016; Zhao et al., 2017] and the cost function for polycube deformations [Gregson et al., 2011; Huang et al., 2014]. Although polycube methods can obtain cubic geometry, they fail to preserve details (Fig. 5.8) because detail preservation is not required in their intended applications such as parameterization and hexahedral meshing [Wang et al., 2007; Lin et al., 2008; Wang et al., 2008; He et al., 2009; García Fernández et al., 2013; Yu et al., 2014; Cherchi et al., 2016;

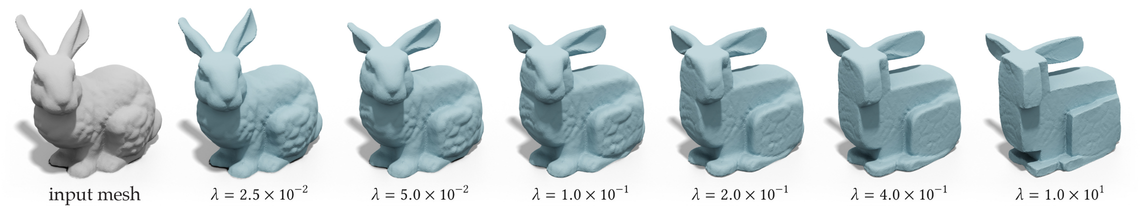
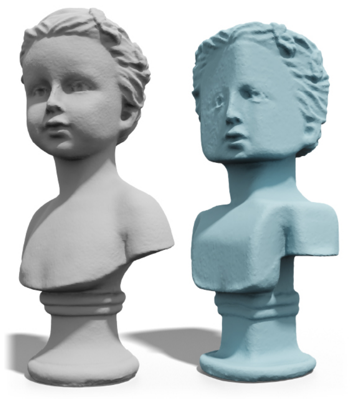

input mesh
$\lambda = 2.5 \times 10^{-2}$
$\lambda = 5.0 \times 10^{-2}$
$\lambda = 1.0 \times 10^{-1}$
$\lambda = 2.0 \times 10^{-1}$
$\lambda = 4.0 \times 10^{-1}$
$\lambda = 1.0 \times 10^{1}$

Figure 5.9: One can control the CUBENESS by changing the λ parameter in Eq. (5.1).

Fang et al., 2016].



One tempting direction for creating cubic geometry is to use voxelization. However, voxelization fails to capture the details depicted by the artists and cannot capture the wide spectrum of cubeness across cubic sculptures. Another tempting direction is to recover geometric features from the polycube results. This would lead to a multi-step algorithm and suffer from limitations of particular detail encoding schemes (e.g., bump maps). Even if we stop the polycube algorithm earlier such as the method of [Gregson et al., 2011] to maintain details, it does not provide a satisfactory solution (see the inset for a comparison with Fig. 5 in [Gregson et al., 2011]). More importantly, many artistic controls in Sec. 5.4 would be nontrivial to add on. Building stylization on top of polycube methods would also suffer from slow performance. For instance, Huang et al. [2014] propose a polycube method that minimizes the ℓ^1-norm of the normals on the deformed tetrahedral mesh with ARAP for regularization. Their formulation involves minimizing a complicated non-linear function and requires minutes to hours to optimize. Thus a stylization built on top of this method would be even slower. In contrast, our formulation is a single energy optimization which can easily incorporate many artistic controls (Sec. 5.4). Our energy is similar to the polycube energy of [Huang et al., 2014] in that we also minimize the ARAP energy with an ℓ^1 regularization, but the key difference is that we define the ℓ^1-norm on the *rotated normals* of the *original* mesh instead. This allows us to optimize our energy much faster using the local-global approach with ADMM in only a few seconds (Table 7.1).

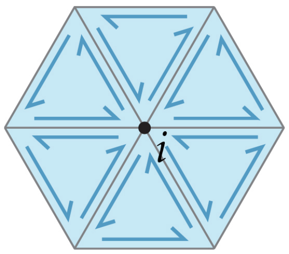
5.3 Method

The input to our method is a manifold triangle mesh with/without boundaries. Our method outputs a *cubified* shape where each subcomponent has the style of an axis-aligned cube. Meanwhile, our stylization maintains the geometric details of the original mesh.

Let V be a $|\mathsf{V}| \times 3$ matrix of vertex positions at the rest state and $\widetilde{\mathsf{V}}$ be a $|\mathsf{V}| \times 3$ matrix containing the deformed vertex positions. We denote by $\mathsf{d}_{ij} = [\mathsf{v}_j - \mathsf{v}_i]^\top$ and $\widetilde{\mathsf{d}}_{ij} = [\widetilde{\mathsf{v}}_j -$

$\widetilde{\mathsf{v}}_i]^\top$ the edge vectors between vertices i, j at the rest and deformed states respectively. The energy for our cubic stylization is as follows

$$\underset{\widetilde{\mathsf{V}},\{\mathsf{R}_i\}}{\text{minimize}} \sum_{i\in V} \sum_{j\in\mathcal{N}(i)} \underbrace{\frac{w_{ij}}{2}\|\mathsf{R}_i\mathsf{d}_{ij} - \widetilde{\mathsf{d}}_{ij}\|_F^2}_{\text{ARAP}} + \underbrace{\lambda a_i\|\mathsf{R}_i\mathsf{n}_i\|_1}_{\text{CUBENESS}}. \tag{5.1}$$



The first term is the ARAP energy [?], where R_i is a 3-by-3 rotation matrix, w_{ij} is the cotangent weight [Pinkall and Polthier, 1993], and $\mathcal{N}(i)$ denotes the "spokes and rims" edges of the ith vertex [Chao et al., 2010] (see the inset). In the second term, n_i denotes the unit area-weighted normal vector of a vertex i in $\mathbb{R}^3$. The $a_i \in \mathbb{R}^+$ is the barycentric area of vertex i, which is crucial for λ to exhibit the similar cubeness across different mesh resolutions. Intuitively, minimizing the ℓ^1-norm of the rotated normal encourages $\mathsf{R}_i\mathsf{n}_i$ to align with one of coordinate axes because the ℓ^1-norm encourages sparsity. Combining the two, the optimal rotation $\{\mathsf{R}_i^\star\}$ would simultaneously preserve the local structure (ARAP) and encourage the output normal to align with one of the axes (CUBENESS).

We adapt the standard local-global update strategy to optimize our energy [?] (see Alg. 3). Our global step, updating $\widetilde{\mathsf{V}}$, is achieved by solving a linear system, the same as Equation 9 in [?]. Our local step, finding the optimal rotation, is however different from the previous literature due to the ℓ^1 term.

5.3.1 Local-Step

Our local step for each vertex i can be written as

$$\mathsf{R}_i^\star = \underset{\mathsf{R}_i\in SO(3)}{\arg\min}\ \frac{1}{2}\|\mathsf{R}_i\mathsf{D}_i - \widetilde{\mathsf{D}}_i\|_{\mathsf{W}_i}^2 + \lambda a_i\|\mathsf{R}_i\mathsf{n}_i\|_1, \tag{5.2}$$

where W_i is a $|\mathcal{N}(i)| \times |\mathcal{N}(i)|$ diagonal matrix of cotangent weights, D_i and $\widetilde{\mathsf{D}}_i$ are $3 \times |\mathcal{N}(i)|$ matrices of rim/spoke edge vectors at the rest and deformed states respectively. We denote $\|\mathsf{X}\|_\mathsf{Y}^2 = \mathrm{Tr}(\mathsf{XYX}^\top)$ for notational convenience. By setting $\mathsf{z} = \mathsf{R}_i\mathsf{n}_i$, we can rewrite Eq. (5.2) as

$$\begin{aligned} \underset{\mathsf{z},\mathsf{R}_i\in SO(3)}{\text{minimize}} \quad & \frac{1}{2}\|\mathsf{R}_i\mathsf{D}_i - \widetilde{\mathsf{D}}_i\|_{\mathsf{W}_i}^2 + \lambda a_i\|\mathsf{z}\|_1 \\ \text{subject to} \quad & \mathsf{z} - \mathsf{R}_i\mathsf{n}_i = 0. \end{aligned} \tag{5.3}$$

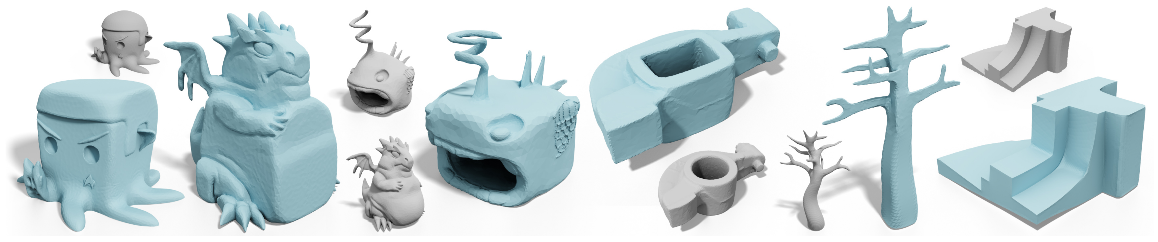


Figure 5.10: We turn 3D shapes into the cubic style (blue) with Alg. 3. ©Angelo Tartanian (top left), Splotchy Ink (top), Dan Slack (top right) under CC BY.

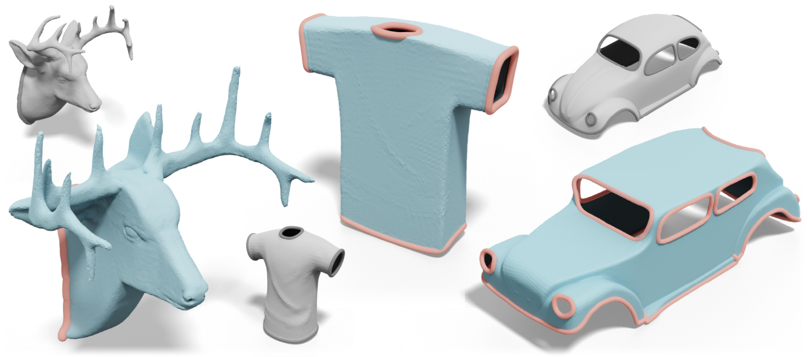


Figure 5.11: We can also turn meshes with boundaries (red) into the cubic style. ©Takeshi Murata (left) under CC BY.

Eq. (5.3) is a standard ADMM formulation. We solve this local step using the scaled-form ADMM updates [Boyd et al., 2011]:

$$\mathsf{R}_i^{k+1} \leftarrow \underset{\mathsf{R}_i \in SO(3)}{\arg\min} \frac{1}{2} \|\mathsf{R}_i \mathsf{D}_i - \widetilde{\mathsf{D}}_i\|^2_{\mathsf{W}_i} + \frac{\rho^k}{2} \|\mathsf{R}_i \mathsf{n}_i - \mathsf{z}^k + \mathsf{u}^k\|_2^2 \tag{5.4}$$

$$\mathsf{z}^{k+1} \leftarrow \underset{\mathsf{z}}{\arg\min}\ \lambda a_i \|\mathsf{z}\|_1 + \frac{\rho^k}{2} \|\mathsf{R}_i^{k+1} \mathsf{n}_i - \mathsf{z} + \mathsf{u}^k\|_2^2 \tag{5.5}$$

$$\tilde{\mathsf{u}}^{k+1} \leftarrow \mathsf{u}^k + \mathsf{R}_i^{k+1} \mathsf{n}_i - \mathsf{z}^{k+1} \tag{5.6}$$

$$\rho^{k+1}, \mathsf{u}^{k+1} \leftarrow \mathit{update}\ (\rho^k) \tag{5.7}$$

where $\rho \in \mathbb{R}_+$ is the penalty and u is the scaled dual variable.

Eq. (5.4) is an instance of the *orthogonal Procrustes* method [Gower et al., 2004]

$$\mathsf{R}_i^{k+1} \leftarrow \underset{\mathsf{R}_i \in SO(3)}{\arg\max}\ \mathrm{Tr}(\mathsf{R}_i \mathsf{M}_i)$$

$$\mathsf{M}_i = \begin{bmatrix} \mathsf{D}_i & \mathsf{n}_i \end{bmatrix} \begin{bmatrix} \mathsf{W}_i & \\ & \rho^k \end{bmatrix} \begin{bmatrix} \widetilde{\mathsf{D}}_i^\top \\ (\mathsf{z}^k - \mathsf{u}^k)^\top \end{bmatrix}.$$

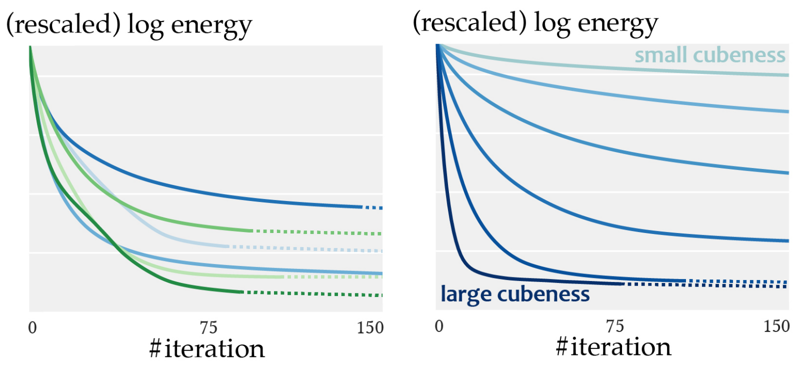


Figure 5.13: We show the convergence behavior of different meshes in Fig. 5.10 (left, blue), Fig. 5.16 (left, green), and different cubenesses in Fig. 5.9 (right). Note that the dotted line imply the optimization has stopped.

One can derive the optimal R_i from the singular value decomposition of $\mathsf{M}_i = \mathcal{U}_i \Sigma_i \mathcal{V}_i^\top$:

$$\mathsf{R}_i^{k+1} \leftarrow \mathcal{V}_i \mathcal{U}_i^\top, \tag{5.8}$$

up to changing the sign of the column of $\mathcal{U}_i$ so that $\det(\mathsf{R}_i) > 0$.

Eq. (5.5) is an instance of the *lasso* problem [Tibshirani, 1996; Boyd et al., 2011], which can be solved with a *shrinkage* step:

$$\mathsf{z}^{k+1} \leftarrow \mathcal{S}_{\lambda a_i/\rho^k}(\mathsf{R}_i^{k+1}\mathsf{n}_i + \mathsf{u}^k) \tag{5.9}$$
$$\mathcal{S}_\kappa(\mathsf{x})_j = (1 - \kappa/|x_j|)_+ \, x_j$$

We update the penalty ρ (Eq. (5.7)) according to Sec. 3.4.1 in [Boyd et al., 2011] where u needs to be rescaled accordingly after updating ρ.

In short, local fitting is performed by running Eq. (5.8), (5.9), (5.6), and (5.7) iteratively until the norms of primal/dual residuals are small. Warm starting the local-step parameters from the previous iteration can significantly speed up the optimization. Specifically, we initialize z, u with zeros, and set the initial $\rho = 10^{-4}$, $\epsilon^{\text{abs}} = 10^{-5}$, $\epsilon^{\text{rel}} = 10^{-3}$, $\mu = 10$, and $\tau^{\text{incr}} = \tau^{\text{decr}} = 2$ (the same notation as used in Sec. 3 of [Boyd et al., 2011]). Then $\mathsf{z}, \mathsf{u}, \rho$ are reused in consecutive iterations. Note that for extremely large λ one may need to increase the initial value of ϵ^{abs} accordingly in order to avoid bad local minima. We stop the optimization when the relative displacement, the infinity norm of relative per vertex displacements, is lower than 3×10^{-3} (see Fig. 5.13 for the convergence plots). More elaborate stopping criteria, such as

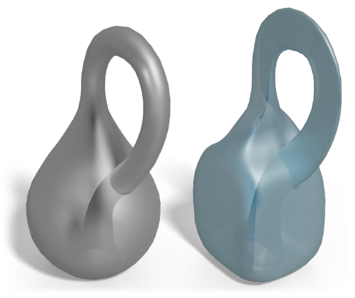


Figure 5.12: Our method can cubify non-orientable surfaces such as the Klein bottle.

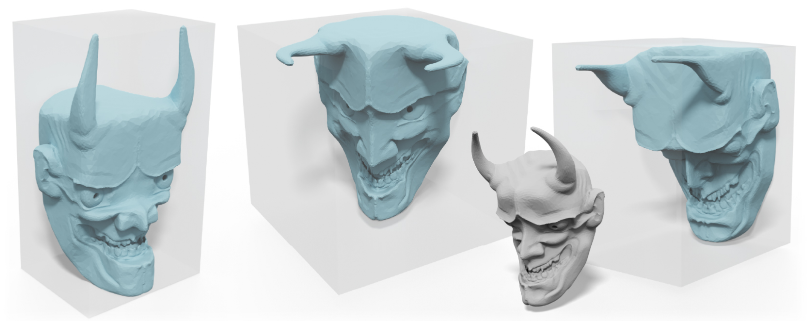


Figure 5.14: The global orientation of the shape influences the ℓ^1 term in Eq. (5.1). Applying different rotations to the mesh lead to different results. ©My Dog Justice under CC BY.

Algorithm 3: *Cube Stylization* (λ)

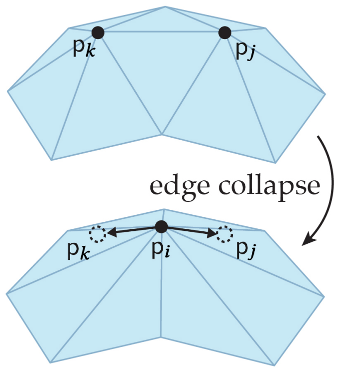
```
Input  : A triangle mesh $V, F$
Output: Deformed vertex positions $\widetilde{V}$
1. $\widetilde{V} \leftarrow V$
2. while not converge do
3.     $R \leftarrow$ local-step $(V, \widetilde{V}, \lambda)$
4.     $\widetilde{V} \leftarrow$ global-step $(R)$
```

the method of [?], could also be used.

At this point we have completed the cubic stylization algorithm summarized in Alg. 3, enabling us to efficiently create cubified shapes (see Fig. 5.10). In Fig. 5.11 and 5.12 we show that this formulation is applicable to meshes with boundaries and non-orientable surface respectively. As the CUBENESS is dependent on the orientation of the mesh, one can apply different rotations to control how the stylization runs (Fig. 5.14). We expose the weighting λ as a design parameter controlling the cubeness of a shape (Fig. 5.9).

However, the "vanilla" cube stylization shares the same caveat as other distortion minimization algorithms: having slow runtime on high resolution meshes.

5.3.2 Affine Progressive Meshes



A hierarchical approach to accelerate ARAP deformations was proposed by Manson and Schaefer [2011]. The main idea is to deform a low-resolution model and recover the details back after convergence.

Specifically, Manson and Schaefer [2011] propose a progressive mesh [Hoppe, 1996] representation which first simplifies a given mesh via a sequence of edge collapses, and then represents the mesh as its coarsest form together with a sequence of vertex splits. After applying some deformations to the coarsest mesh, each

```
Algorithm 4: Fast Cube Stylization ($\lambda, m$)
  Input  : A triangle mesh $\mathsf{V}, \mathsf{F}$
  Output: Deformed vertex positions $\widetilde{\mathsf{V}}$
  // pre-processing
  1. $m \leftarrow$ target number of faces
  2. $\mathsf{V}_c, \mathsf{F}_c \leftarrow$ edge collapses $(\mathsf{V}, \mathsf{F}, m)$
  // cubic stylization
  3. $\widetilde{\mathsf{V}}_c \leftarrow \mathsf{V}_c$
  4. while not converge do
  5.     $\mathsf{R} \leftarrow$ local-step $(\mathsf{V}_c, \widetilde{\mathsf{V}}_c, \lambda)$
  6.     $\widetilde{\mathsf{V}}_c \leftarrow$ global-step $(\mathsf{R})$
  7. $\widetilde{\mathsf{V}}, \mathsf{F} \leftarrow$ affine vertex splits $(\widetilde{\mathsf{V}}_c, \mathsf{F}_c)$
```

"deformed" vertex split is computed by fitting the best local rigid transformation. This approach is suitable for deformations that are locally rigid (e.g., ARAP), but our cubic stylization is *less* rigid for larger λ.

So we fit the best *affine* transformation at each vertex split, rather than using rigid transformations. Specifically, at each edge collapse we store the displacement vectors from the newly inserted vertex p_i to the endpoints $\mathsf{p}_j, \mathsf{p}_k$ (see the inset) together with a matrix A:

$$\mathsf{A} = (\mathsf{Q}_i \mathsf{Q}_i^\top)^{-1} \mathsf{Q}_i.$$

Q_i is a $3 \times |\mathcal{N}(i)|$ matrix where each column is the vector from p_i to one of its one-rings neighbors $\mathcal{N}(i)$. If $(\mathsf{Q}_i \mathsf{Q}_i^\top)$ is singular (e.g., in planar regions), we remedy the issue with the Tikhonov regularization [Tikhonov et al., 2013]. Then A is used to computed the deformed displacements for each vertex split as

$$\widetilde{\mathsf{p}}_j - \widetilde{\mathsf{p}}_i = \widetilde{\mathsf{Q}}_i \mathsf{A}^\top (\mathsf{p}_j - \mathsf{p}_i),$$

where $\widetilde{\mathsf{p}}_i$ denotes the position of vertex i in the cubified coarsened shape, and $\widetilde{\mathsf{Q}}_i$ is a $3 \times |\mathcal{N}(i)|$ matrix containing vectors from $\widetilde{\mathsf{p}}_i$ to its one-rings neighbors.

Affine progressive meshes allows us to losslessly recover the original meshes undergoing affine transformations. For smooth non-affine transformations such as our cube stylization, it could still be approximately recovered (see Fig. 5.15). We summarize our cubic stylization with the affine progressive mesh in Alg. 4. Note that edge collapse is just a pre-processing step. In the online stage, one only needs to run cubic stylization on the coarsest mesh and then apply a sequence of vertex splits to visualize the result on the original resolution. This offers a huge speed-up when interactively modifying the parameter λ on highly detailed models (see Fig. 5.16).

An interesting observation is that the number of faces m in the coarsest mesh not only

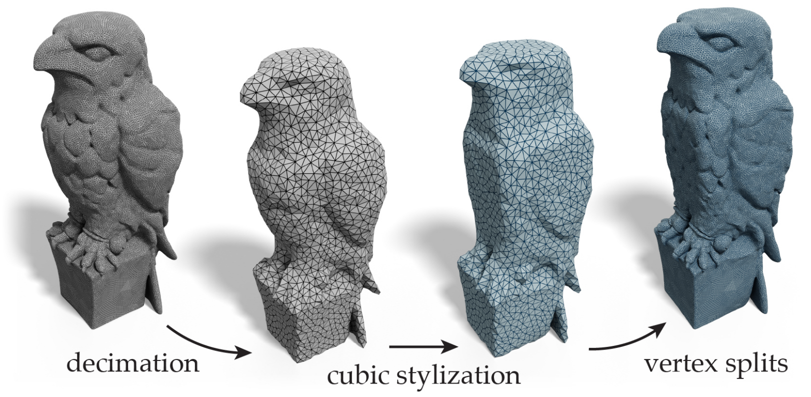


Figure 5.15: Affine progressive meshes allow us to run cubic stylization on a low-resolution model and then recover original details when converged. ©Colin Freeman under CC BY.

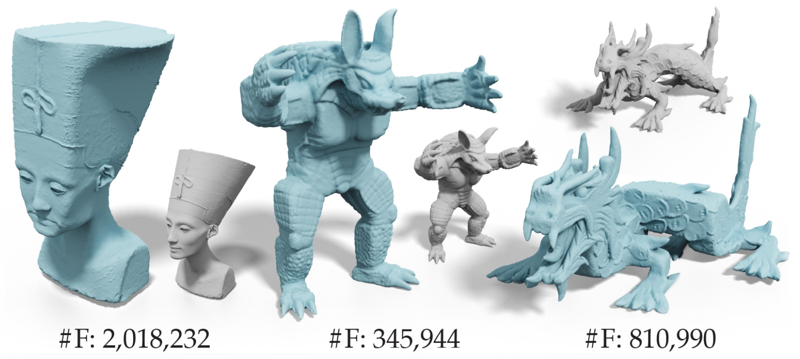


Figure 5.16: With affine progressive meshes, we can scale the cubic stylization to meshes with millions of faces. The Nefertiti mesh (left) was scanned by Nora Al-Badri and Jan Nikolai Nelles from the Nefertiti bust.

controls the runtime, but implicitly controls the frequency level of geometric details that gets preserved. In Fig. 5.17 we show that, for the same λ, a smaller m keeps details across a wider frequency range; in contrast, a larger m only keeps details at higher frequencies. Therefore one can manipulate the level of preserved features by manipulating m.

5.3.3 Implementation

We implement the cubic stylization in C++ using LIBIGL [Jacobson et al., 2018] and evaluate our runtime on a MacBook Pro with an Intel i5 2.3GHz processor. Table 5.1 lists the parameters and the runtime for the stylization shown in Fig. 5.10 (top) and Fig. 5.16. We test our methods on meshes in the *Thingi10K* [Zhou and Jacobson, 2016] and show that we can obtain stylized geometry within a few seconds. This is important for users to receive quick feedback on their parameter choices and iterate on their designs, such as the cubeness λ in Fig. 5.9 and the level of detail m in Fig. 5.17.

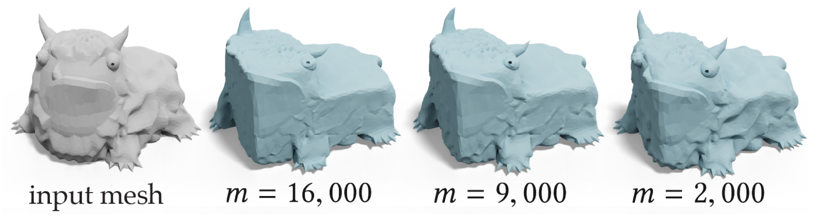
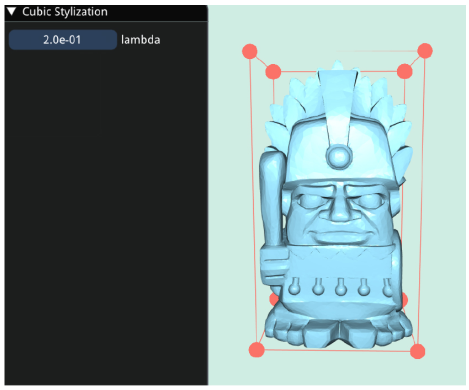


Figure 5.17: The number of faces m used in the decimated mesh not only controls the runtime but also the frequency level of details that get preserved. ©Joseph Larson under CC BY.

Table 5.1: For each example in Fig. 5.10 and Fig. 5.16, we report the number of faces in the original model ($|F|$), $l1$ weight (λ), number of faces of the coarsest mesh (m), number of iterations (*Iters.*), pre-processing time (*Pre.*), and runtime at the online stage (*Runtime*).

Model	$\lvert F \rvert$	λ	m	*Iters.*	*Pre.*	*Runtime*
Fig. 5.10, left	39K	0.20	n/a	106	n/a	**5.08s**
Fig. 5.10, mid.	41K	0.20	n/a	93	n/a	**4.50s**
Fig. 5.10, right	21K	0.4	n/a	86	n/a	**2.26s**
Fig. 5.16, left	2018K	0.20	20K	83	64.19*s*	**3.93s**
Fig. 5.16, mid.	346K	0.40	20K	222	10.69*s*	**4.59s**
Fig. 5.16, right	811K	0.30	40K	173	30.44*s*	**8.38s**



User study We build a user interface (see the inset) to conduct an informal user study with six participants (4 male, 2 female) between the ages of 24 and 29. Participant 3D modeling experience ranged from none (complete novice) to three years of hobbyist use. Each participant was instructed for three minutes on how to use our software to load a mesh and control the cubeness parameter λ. Then we asked them to cubify a shape of their choosing from a collection of ten shapes. The results of their work are show in Fig. 5.18. All users reported that they were satisfied with the cubeness of their resulting shape. One user said that controlling the cubeness of their resulting shape is very easy because it only requires tuning a single parameter.

5.4 Artistic Controls

In addition to the two parameters λ, m, we expose many variants of our stylization to incorporate artistic controls. As a non-realistic modeling tool, this is important for users to realize their creativity.

We first focus our discussion on a variety of artistic controls that are related to the cube-

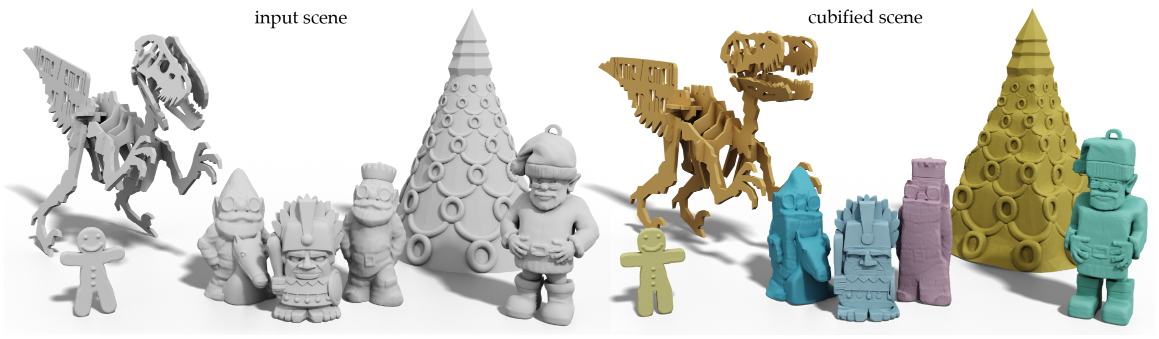


Figure 5.18: Even non-professional users can effortlessly turn an input scene (top) into a cubified scene (bottom). Different colors are results created by different users. From left to right, ©Peter Leppik, Cleven, TerenceKing, MakerBot, TerenceKing, PerryEngel, and Christina Chun under CC BY.

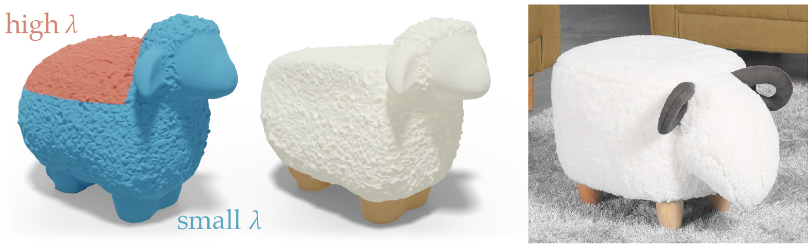


Figure 5.19: We vary λ across the surface to have different cubeness for different parts. We apply higher λ on the red region and smaller λ for the blue region to create an ottoman-like shape (middle). ©pmoews under CC BY.

ness parameter λ. Although Eq. (5.1) only has a single λ for an entire shape, we can actually specify different λ_i for each vertex independently to have non-uniform cubeness, which leads to the expression $\lambda_i a_i \|\mathsf{R}_i \mathsf{n}_i\|_1$. In Fig. 5.19, we use this approach to make the back of the sheep much more cubic than the rest of the shape to create an ottoman-like geometry. We can also specify the non-uniform cubeness λ_i in a different way, instead of painting on the surface directly. In Fig. 5.20 we *paint* a function on the Gauss map in which the surface normal pointing towards the left has higher cubeness. When we map this function back to the surface, we get a cubified owl that is more cubic when initial normals point towards the left and less cubic when they point towards the right. Similarly, we can have different $\lambda_x, \lambda_y, \lambda_z$ for different axes. In Fig. 5.21, we replace the CUBENESS in Eq. (5.1) with $a_i(\lambda_x|(\mathsf{R}_i\mathsf{n}_i)_x| + \lambda_y|(\mathsf{R}_i\mathsf{n}_i)_y| + \lambda_z|(\mathsf{R}_i\mathsf{n}_i)_z|)$ and specify a different values for each $\lambda_x, \lambda_y, \lambda_z$ to have the style of a rectangular prism.

If one wants to fix certain parts of the shape, we can easily add constraints in the global step, as in the method of **?**. In Fig. 5.4 we add the parts constraint by fixing the position of some vertices when solving the linear system; we add the points constraint by specifying some deformed vertices $\widetilde{\mathsf{V}}_i$ at user-desired positions. We can also use the same methodology to constrain some parts of the geometry lying on certain planes. For instance, setting $(\widetilde{\mathsf{V}}_i)_x = 0$ can force vertex i to lie on the yz-plane. In Fig. 5.22 we use this plane constraint

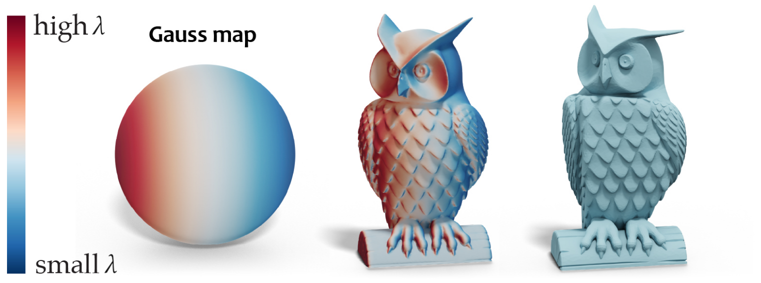


Figure 5.20: We can paint the λ function on the Gauss map to have non-uniform λ over the surface. In the figure, we have higher λ for the original normals pointing to the left. ©Tom Cushwa under CC BY.

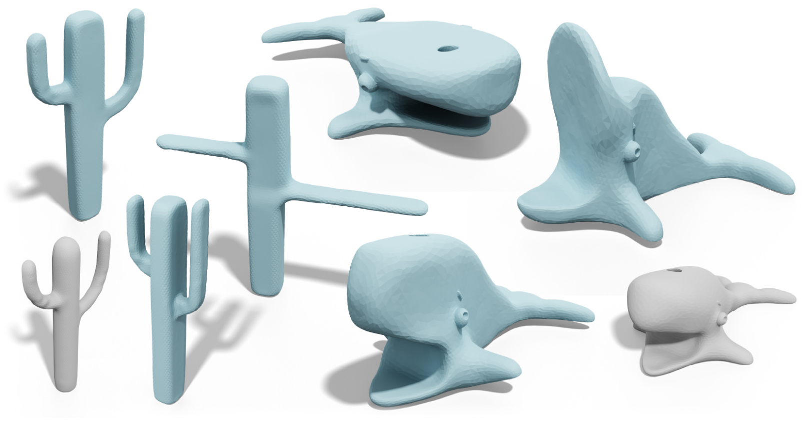


Figure 5.21: We can vary λ for different axes to turn inputs into the biased cubic style (blue) towards x,y,z axes respectively. ©MakerBot (right) under CC BY.

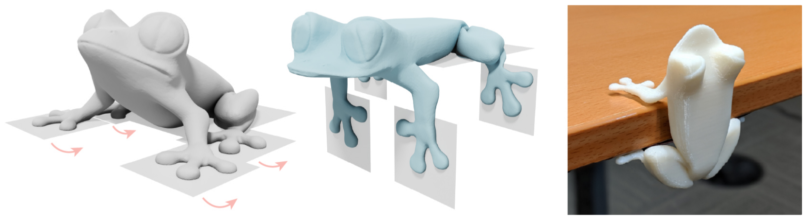


Figure 5.22: We constrain certain parts of the geometry lying on certain planes to create a 3D printed table clinger (right). ©Morena Protti under CC BY.

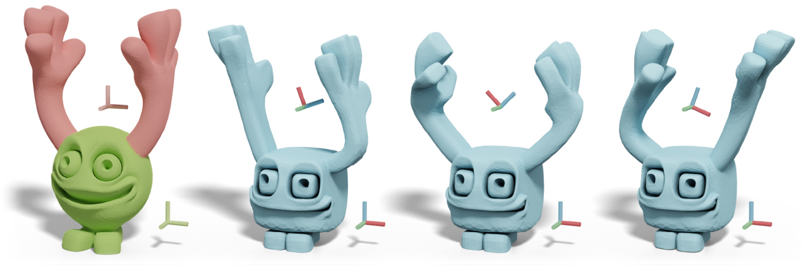


Figure 5.23: We can define the ℓ^1-norm on different coordinate systems for different parts of the shape, instead of using the world coordinates. In the figure the hands and the body use different coordinate systems (left). By changing them, we can vary the cube orientations for different parts. ©David Hagemann under CC BY.

to create a table clinger.

In addition, one can utilize the property of the ℓ^1-norm to have different artistic effects. Because the CUBENESS term is orientation dependent, in Fig. 5.14 we can apply different rotations to the mesh before the stylization to control the results. Rather than rotating the mesh, another way is to encode the normal vector in a different coordinate system $\lambda a_i \|\mathsf{R}_i \mathsf{n}_i^{\text{local}}\|_1$, where we use $\mathsf{n}_i^{\text{local}}$ to denote the user-desired coordinate system for vertex i. This perspective allows us to define the ℓ^1-norm on different coordinate systems for different parts of the shape to obtain different cube orientations (Fig. 5.23). Beyond the cubic stylization, in Fig. 5.24, 5.25 we apply a coordinate transformation B inside the ℓ^1-norm $\lambda a_i \|\mathsf{B}\mathsf{R}_i \mathsf{n}_i\|_1$ to achieve polyhedral stylization, for which we provide the details in App. 5.6.1. Once we obtain the stylized shapes, they are ready to be used by standard deformation techniques in animations (Fig. 5.26).

5.5 Limitations & Future Work

Accelerating the stylization to real-time would enable faster iterations between designs. Developing a more robust stylization for bad quality triangles, non-manifold meshes, or even point cloud could be useful for stylizing real-world geometric data. Guaranteeing results to be self-intersection free would be desirable for downstream tasks. Extending our energy to be invariant to discretizations could achieve more consistent results across dif-

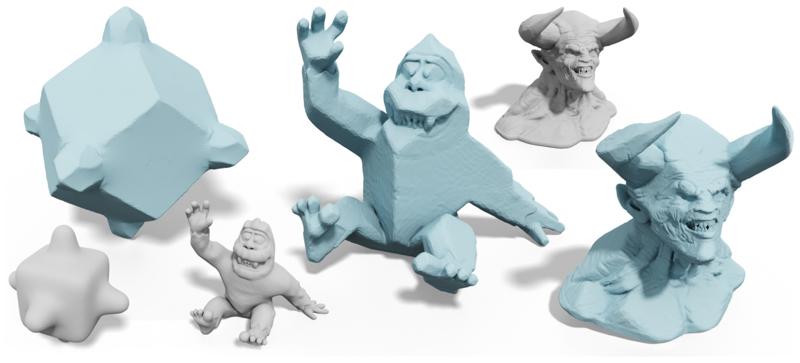


Figure 5.24: We apply a coordinate transformation inside the ℓ^1-norm to generalize cubic stylization to polyhedrons. ©Proto Paradigm (middle), Ola Sundberg (right) under CC BY.

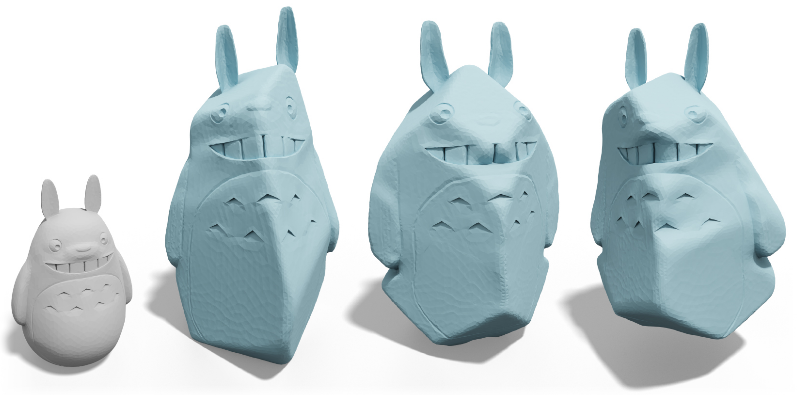


Figure 5.25: We apply non-symmetric coordinate transformations inside the ℓ^1-norm to create irregular polyhedral stylization. ©Johannes under CC BY.

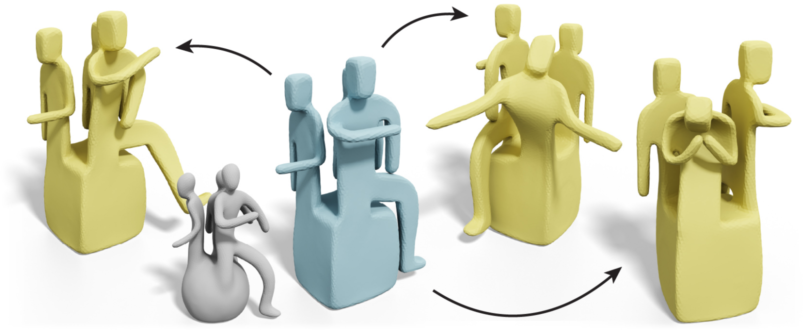


Figure 5.26: Once we have the cubic geometry (blue), standard deformation techniques (e.g., [**?**]) can be used to manipulate the cubified shape (yellow).

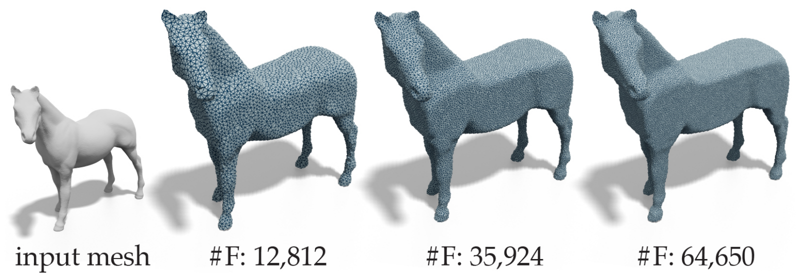


Figure 5.27: Although exhibiting similar cubenesses, our stylization is still not invariant to different resolutions.

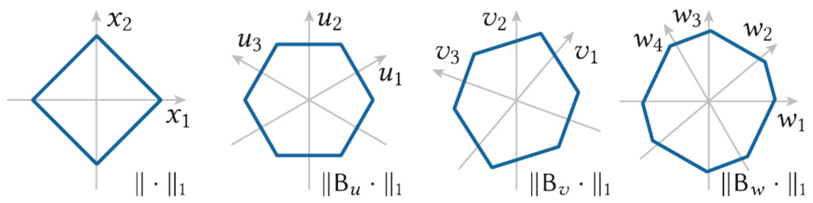


Figure 5.28: By specifying different coordinate transformations B inside the ℓ^1-norm, we can encourage polyhedral style.

ferent resolutions (see Fig. 5.27). Extending to quadrilateral meshes and NURBS surfaces could benefit existing modeling or engineering design softwares. Generalizing to volumetric meshes could have a better volume preservation. Exploring different deformation energies and ℓ^p-norm could lead to novel stylization tools for non-realistic modeling. Beyond generating stylized shapes, the mathematical expression of the cubic geometry could offer insights toward understanding more intricate styles. For instance, *Cubism* has been considered as a revolutionized artistic style for paintings and sculptures. Cubism has appeared since the early 20th century. Since then, several attempts have tried to describe [Henderson, 1983] and generate Cubist art [Wang et al., 2011; Corker-Marin et al., 2018], but more efforts are still required to offer scientific explanations to a wide variety of Cubist art. Our cubic stylization only focuses on a specific style. We hope this could inspire future attempts to capture different sculpting styles such as those presented in African art, or even a generic approach to create different styles in an unified framework.

5.6 Appendix

5.6.1 Polyhedral Generalization

Simply applying a coordinate transformation $B : \mathbb{R}^n \to \mathbb{R}^m$ inside the ℓ^1-norm can encourage polyhedral results (see Fig. 5.28 and 5.25). The ℓ^1-norm of a vector is defined as the summation of its magnitudes along each basis vector. Thus applying a coordinate transformation inside the ℓ^1-norm changes its bahavior because the basis vectors are different.

Following the notation in Eq. (5.1), polyhedron energy can be written as

$$\underset{\widetilde{\mathsf{V}},\{\mathsf{R}_i\}}{\text{minimize}} \sum_{i\in V}\sum_{j\in\mathcal{N}(i)} \frac{w_{ij}}{2}\|\mathsf{R}_i\mathsf{D}_{ij} - \widetilde{\mathsf{D}}_{ij}\|_F^2 + \lambda a_i \|\mathsf{B}\mathsf{R}_i\mathsf{n}_i\|_1.$$

In our case, B is a m-by-3 coordinate transformation matrix for shapes embedded in $\mathbb{R}^3$. Again by setting $\mathsf{z} = \mathsf{R}_i\mathsf{n}_i$ we can reach almost the same optimization procedures, except the Eq. (5.5) now becomes (we ignore the iteration superscript for clarity)

$$\mathsf{z}^{k+1} \leftarrow \underset{\mathsf{z}}{\arg\min}\ \lambda a_i\|\mathsf{B}\mathsf{z}\|_1 + \frac{\rho}{2}\|\mathsf{R}_i\mathsf{n}_i - \mathsf{z} + \mathsf{u}\|_2^2. \tag{5.10}$$

Similar to common techniques for solving the *Basis Pursuit* problem, we introduce a variable $\mathsf{t} \succeq \|\mathsf{B}\mathsf{z}\|_1$ to transform Eq. (5.10) into a small quadratic program subject to equality constraints

$$\begin{aligned}\underset{\mathsf{z},\mathsf{t}}{\text{minimize}}\quad & \begin{bmatrix}\mathsf{z}^\top & \mathsf{t}^\top\end{bmatrix}\begin{bmatrix}\rho/2\cdot \mathsf{I}_3 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\mathsf{z}\\ \mathsf{t}\end{bmatrix}\\ & + \begin{bmatrix}-\rho(\mathsf{R}_i\mathsf{n}_i+\mathsf{u})^\top & \lambda a_i 1_m^\top\end{bmatrix}\begin{bmatrix}\mathsf{z}\\ \mathsf{t}\end{bmatrix}\\ \text{subject to}\quad & \begin{bmatrix}\mathsf{B} & -\mathsf{I}_m\\ -\mathsf{B} & -\mathsf{I}_m\end{bmatrix}\begin{bmatrix}\mathsf{z}\\ \mathsf{t}\end{bmatrix} \preceq 0,\end{aligned}$$

where I_x and 1_x denote the identity matrix with size x and a column vector of 1 with size x respectively. We then solve this efficiently using CVXGEN [Mattingley and Boyd, 2012]. Note that the results in Fig. 5.24 and Fig. 5.25 use $m = 4$.

Chapter 6

Normal-Driven Spherical Shape Analogies

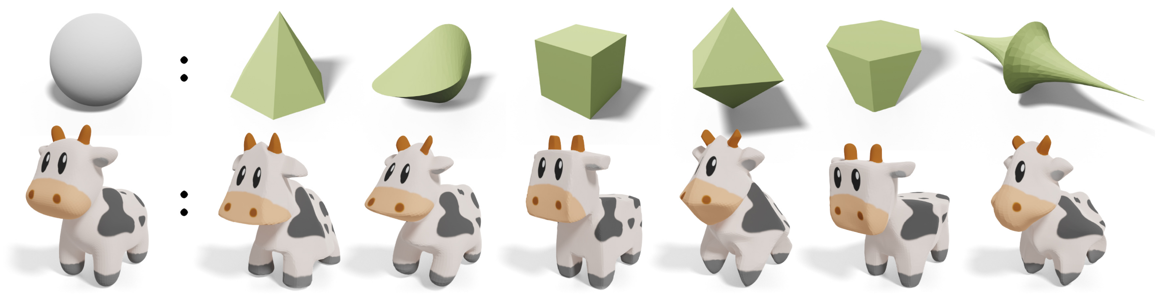


Figure 6.1: Our *normal-driven spherical shape analogy* stylizes an input 3D shape (bottom left) by studying how the surface normal of a style shape (green) relates to the surface normal of a sphere (gray).

This paper introduces a new method to stylize 3D geometry. The key observation is that the surface normal is an effective instrument to capture different geometric styles. Centered around this observation, we cast stylization as a shape analogy problem, where the analogy relationship is defined on the surface normal. This formulation can deform a 3D shape into different styles within a single framework. One can plug-and-play different target styles by providing an exemplar shape or an energy-based style description (e.g., developable surfaces). Our surface stylization methodology enables Normal Captures as a geometric counterpart to material captures (MatCaps) used in rendering, and the prototypical concept of Spherical Shape Analogies as a geometric counterpart to image analogies in image processing.

6.1 Introduction

Analogies of the form $A : A' :: B : B'$ is a reasoning process that conveys A *is to* A' *as* B *is to* B'. This formulation has become a core technique for creating artistic 2D digital content, such as image analogies [Hertzmann et al., 2001] in Photoshop [Adobe Inc., 2021] for image stylization and the Lit Sphere [Sloan et al., 2001] (a.k.a. MatCap) in ZBrush [Pixologic Inc., 2020] for non-photorealistic renderings. However, leveraging analogies to stylize 3D geometry is still at a preliminary stage because defining the analogy relationship on surface meshes requires dealing with irregular discretizations, curved metrics, and different topologies.

In this paper, we introduce a step towards more general 3D shape analogies, named *spherical shape analogies*. We consider a specific case where A is a unit sphere. This restriction enables us to operate on an input mesh B with arbitrary topologies, boundaries, and geometric complexity. While not fully general, because A is restricted to be a sphere, we demonstrate that this formulation can immediately achieve different geometric styles within a single framework. In Fig. 6.1, we show that by providing different target style shapes A' to the algorithm, we can turn the input shape B into different styles. In addition to stylization, our method can encompass many existing applications, such as developable surface approximation and PolyCube deformation.

One key observation in our spherical shape analogies is that the surface normal is an effective instrument to capture geometric styles. Thus, we define the analogy relationship based on normals: we optimize a stylized shape B' such that the relationship between the surface normals of B and B' is the same as the relationship between the surface normals of A and A'

We realize this by casting stylization as a simple and effective normal-driven shape optimization problem which aims at deforming the input shape towards a set of desired normals. However, such an optimization problem is often difficult due to the nonlinearity of unit normals. We draw inspiration from previous works and apply a change of variables to accelerate the computation: instead of directly optimizing the vertex positions, we optimize a set of rotations that rotate the normals of the input mesh to the set of desired normals. Our simple formulation with the change of variables results in a generic stylization algorithm that runs at interactive rates.

6.2 Related Work

Our work shares similar motivations to computer-assisted image stylization pioneered by Haeberli [1990]. But since our outputs are stylized 3D geometries, we focus the discussion on geometric stylization and geometric deformation methods.

Analogy-based Geometric Stylization

Many generative models have been proposed for creating stylized 3D objects, such as collage art [Gal et al., 2007; Theobalt et al., 2007], manga style [Shen et al., 2012], cubic style [Liu and Jacobson, 2019a], and neuronal homunculus [Reinert et al., 2012]. However, these methods are tailor-made for a specific style.

Analogy $A : A' :: B : B'$ is a powerful formulation for achieving different stylization results within a single framework. This formulation has inspired several design tools for images [Hertzmann et al., 2001], non-photorealistic renderings [Sloan et al., 2001; Fiser et al., 2016], and curves [Hertzmann et al., 2002]. Beyond 2D data, the idea of analogy has also been used for transferring 3D geometric details from one shape to another. We omit the discussion on methods that are not based on analogies, such as mesh cloning [Zhou et al., 2006; Takayama et al., 2011] and geometric learning [Liu et al., 2020; Hertz et al., 2020; Wang et al., 2020; Chen et al., 2021; Li and Zhang, 2021], and focus on analogy-based techniques. Ma et al. [2014] propose a method for 3D style transfer based on patch-based assembly. However, their method cannot handle free-form deformations and requires the source and the exemplar shape to share a similar structure in order to compute high-quality correspondences. Bhat et al. [2004] propose a voxel-based texture synthesis method for transferring geometric details encoded in a volumetric grid. Berkiten et al. [2017] use metric learning for details represented as displacement maps. These methods are designed for high-frequency details (e.g., wrinkles on the surface). In contrast, our spherical shape analogies focuses on larger scale free-form deformations. Albeit limited — in our analogies A is restricted to the unit sphere — our method enables a first step in this exciting direction.

Surface Normals in Shape Deformation

A key insight of our spherical shape analogies is to leverage surface normals to capture geometric styles. The surface normal is a fundamental geometric quantity and is ubiquitous in geometry processing. A representative example is in the *PolyCube* deformation [Tarini et al., 2004] where the goal is to optimize surface normals to be axis-aligned. Gregson et al. [2011] and Zhao et al. [2017] use the closest rotation from the surface normal to an axis-aligned direction to drive the PolyCube deformation. Huang et al. [2014] and Fu et al. [2016] propose to minimize energies defined on normals to create PolyCube shapes. In architectural geometry design, surface normals are a main ingredient for characterizing polygon meshes with planar faces. The methods proposed by Deng et al. [2011] and Poranne et al. [2013] utilize normals to formulate a *distance-from-plane* constraint to encourage planarity. Tang et al. [2014] use the dot product between a face normal and its adjacent edge vectors to determine whether the vertices of a polygon are coplanar.

Characterizing whether a mesh can be flattened to 2D without stretching or shearing, a.k.a. *developability*, also relies on surface normals. Stein et al. [2018a] characterize discrete developability based on the 1-ring face normals, and propose an algorithm to compute piecewise developable surfaces. Sellán et al. [2020] reformulate the developable energy into a convex semidefinite program for finding piecewise developable heightfields. In addition to these examples, deforming shapes into the cubic style [Liu and Jacobson, 2019a; Fumero et al., 2020], constructing shape abstractions [Alexa, 2021], surface parameterization [Zhao et al., 2020], and interactive mesh editing [Yu et al., 2004; Sorkine et al., 2004] are all related

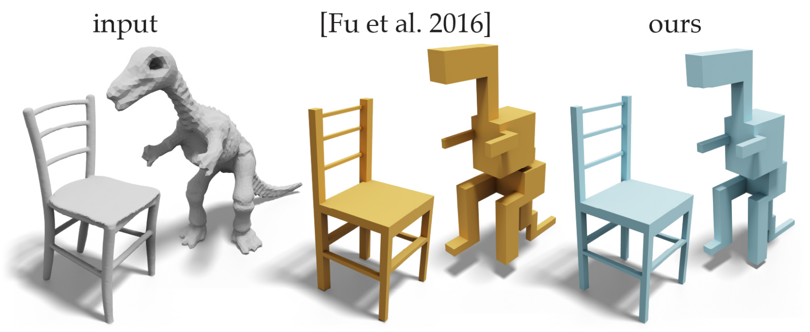


Figure 6.2: Our method can be used to create PolyCube shapes (blue) and obtain comparable results to [Fu et al., 2016] (yellow).

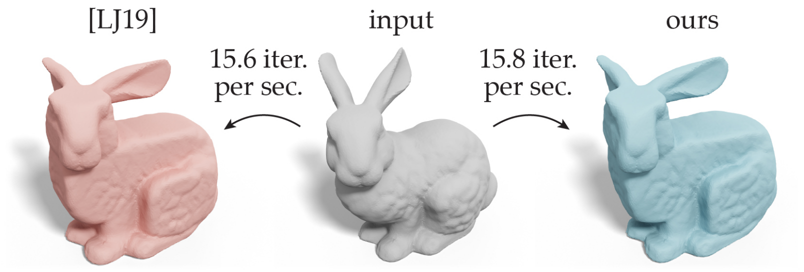


Figure 6.3: Although more general for creating different geometric styles (e.g., Fig. 6.1), our normal driven editing can also be applied to cubic stylization [Liu and Jacobson, 2019a], achieving comparable performance (blue) to the previous method (red).

to surface normals. Many more examples can be found in the design of geometric filters, such as the Guided filter [Zhang et al., 2015b], the Shock filter [Prada and Kazhdan, 2015], the Bilateral normal filter [Zheng et al., 2011], and Total Variation mesh denoising [Zhang et al., 2015a].

Our method can be adapted to these normal-based deformations. Compared to the PolyCube method [Fu et al., 2016], we achieve comparable quality (see Fig. 6.2), but we can further generalize to polytopes (see Fig. 6.18). Compared to [Liu and Jacobson, 2019a] in cubic stylization (see Fig. 6.3), we can achieve similar performance, but we can further generalize to many styles other than the cubic style (see Fig. 6.1). In developable surface approximation, in contrast to the method by Sellán et al. [2020], our method can be applied to surface triangle meshes (see Fig. 6.4) and is significantly faster than the method by Stein et al. [2018a] (see Fig. 6.5).

Shape Deformation

Our geometric stylization method can also be perceived as a type of shape deformation method. We share technical similarities with methods that deform a shape while addressing given modeling constraints. A common choice is to minimize the *as-rigid-as-possible* (ARAP) energy [Sorkine and Alexa, 2007; Igarashi et al., 2005b; Chao et al., 2010] while sat-

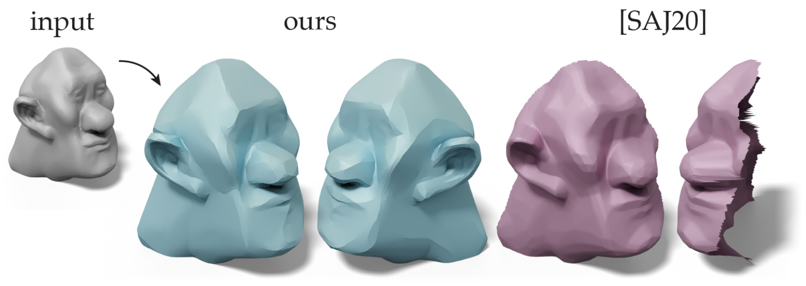


Figure 6.4: Sellán et al. [2020] propose a technique to make 2D heightfields developable (purple). In contrast, our method can create developable approximations for surface meshes in 3D (blue).

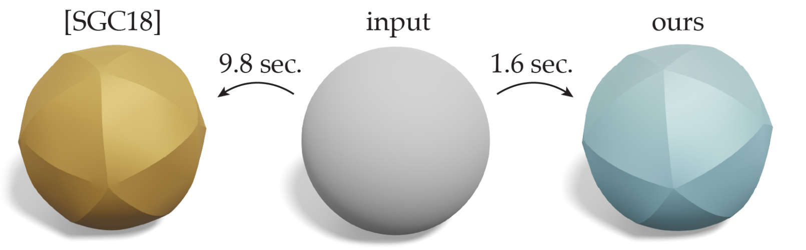


Figure 6.5: Compared to the method proposed by Stein et al. [2018a] for creating developable approximations (left), our method can create visually comparable results (right) with significant speed-ups.

isfying the constraints. This ARAP energy measures the rigidity of local surface patches and favors detail-preserving smooth deformations. In the case where locally rigid deformations are too constrained, the conformal energy [Crane et al., 2011; Vaxman et al., 2015] which preserves angles is commonly used. In contrast to ARAP, the conformal energy often triggers larger deformations as it allows both local uniform scaling and rigid transformations. In addition to mesh deformations, similar energies have also been used for parameterization [Liu et al., 2008], shape optimization [Bouaziz et al., 2012], and simulating mass-spring systems [Liu et al., 2013b]. The ARAP and conformal energies are also commonly used as regularization terms in mesh optimization problems, such as reconstruction [Zollhöfer et al., 2014], surface registration [Huang et al., 2008; Yoshiyasu et al., 2014], PolyCube construction [Huang et al., 2014], and surface stylization [Liu and Jacobson, 2019a]. Their popularity comes from the property that they favor smooth deformations and are amenable to fast optimizations. For the same reasons, we also use these as our regularization energies for interactive modeling tasks (see Fig. 6.11).

6.3 Spherical Shape Analogies

Our main idea is to use surface normals to capture the style of 3D objects: if two shapes share a similar normal "profile", we consider them to exhibit the same geometric style.

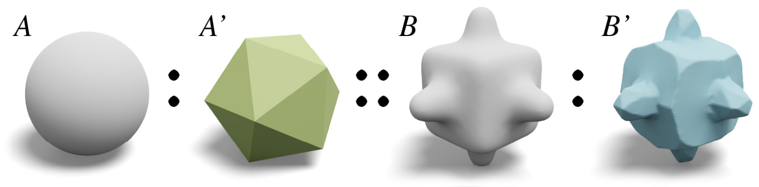


Figure 6.6: We generate an output shape B' that relates to the input B in the same way as how the surface normal of a given primitive A' relates to the surface normal of a sphere A.

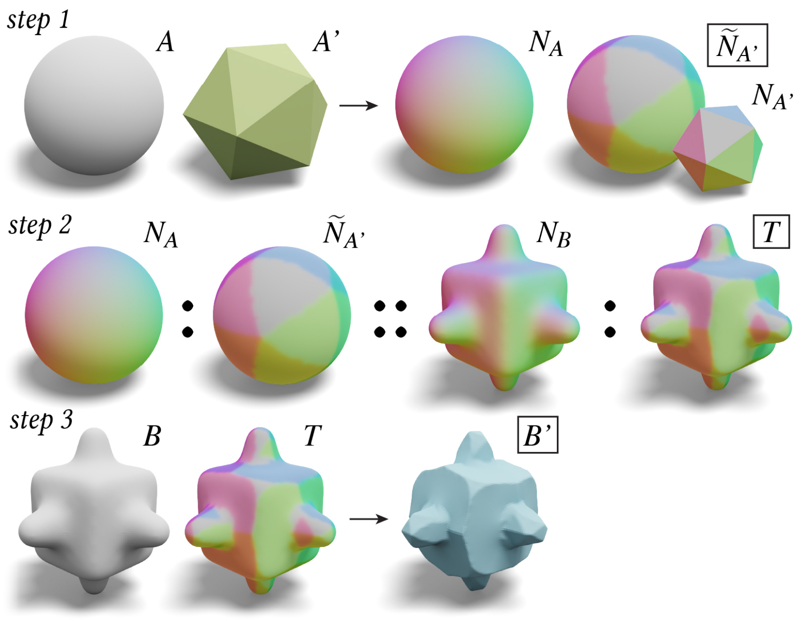


Figure 6.7: Our algorithm defines the analogous relation based on the surface normals. We first map the normals of the style shape $N_{A'}$ to a unit sphere to obtain $\widetilde{N}_{A'}$ (top row), transfer the relationship between N_A and $\widetilde{N}_{A'}$ to the input shape to obtain the target normals T (middle row), then optimize the input shape B so that the actual output normals are aligned with the target normal T (bottom row).

Centered around this observation, as discussed in Sec. 6.1, we propose an analogy-based stylization method to translate the relationship between the normals of A', A to create a stylized output shape B' (see Fig. 6.6). Throughout the paper, we use green color to denote the target style shape A', gray color to denote the input shape B, and blue color to denote the output stylized shape B'.

Our algorithm consists of three simple steps, described in Fig. 6.7: (1) we map the surface normal of A' to a unit sphere A in order to compute target normals on a sphere $\widetilde{N}_{A'}$, (2) we construct analogous target normals T that relate to N_B the same way $\widetilde{N}_{A'}$ relate to N_A, (3) we take B, T as inputs and generate the stylized shape B' whose normals approximate T via optimization.

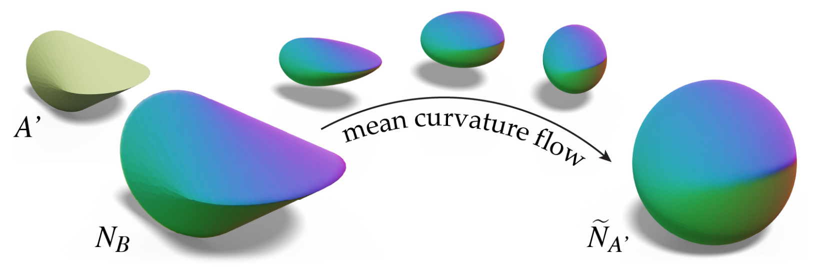


Figure 6.8: Given a style shape A', we run conformalized mean curvature flow [Kazhdan et al., 2012] to map the normals of style shape $N_{A'}$ to a sphere as $\widetilde{N}_{A'}$.

6.3.1 Generating $\widetilde{N}_{A'}$

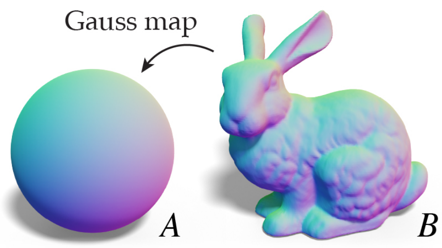
Depending on the provided style shape A' or user preferences, we consider three ways to get a set of target normals on a sphere $\widetilde{N}_{A'}$.

1. *Closest normals.* The simplest case is when the style shape A' is a simple convex shape with only few distinct face normals (e.g., icosahedron). We compute $\widetilde{N}_{A'}$ simply via snapping the normals of the sphere N_A to the nearest normal in the style shape $N_{A'}$.

2. *Spherical parameterization.* For a generic genus-0 shape (e.g., smooth or concave), we compute its parameterization to a sphere using, for example, conformalized mean curvature flow [Kazhdan et al., 2012]. Then $\widetilde{N}_{A'}$ can be computed from the spherical parameterization.

3. *Normal Capture.* If one desires more control, one can manually specify $\widetilde{N}_{A'}$, in the spirit of how *MatCap* (material capture [Sloan et al., 2001]) is used in rendering. We can then skip the first step in Fig. 6.7 and move on to the second step using the user-provided $\widetilde{N}_{A'}$.

6.3.2 Generating T



Generating target normals T on the input shape B using analogy requires correspondences between A and B. We compute the map using the *Gauss map*, leveraging the fact that our A is always a unit sphere (see the inset, where we use colors to visualize the correspondences). Specifically, the unit normal vector of each element (e.g., vertex or face) on the input shape B can be equivalently interpreted as a point on the unit sphere A. Thus, we can easily map signals from A back to B. Once the correspondences are obtained via the normals of input shape N_B, we can trivially compute T by "pasting" $\widetilde{N}_{A'}$ on top of B.

6.3.3 Generating B'

After obtaining a set of target normals $T = \{\mathsf{t}_k\}$ for each vertex k, our goal is to obtain a deformed output shape B' whose surface normals approximate T. Let V be a matrix

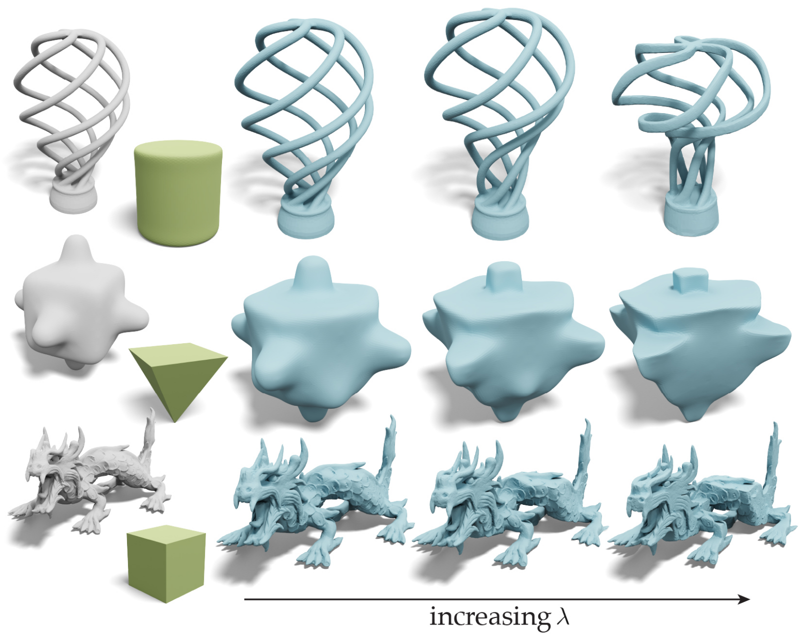


Figure 6.9: The λ parameter in Eq. (6.3) controls the balance between preserving the original shape and satisfying the desired style. We show different stylization results with increasing λ. ©Spiral Light Bulb (top) by benglish under CC BY-SA.

of vertex locations with size $|\mathsf{V}|$-by-3 and F be the face list with size $|\mathsf{F}|$-by-3 of the input shape B. Our output shape B' is a deformed version of the input shape and we use V' to denote the $|\mathsf{V}|$-by-3 matrix of the deformed vertex locations. We formulate the normal-driven deformation as an energy optimization in the following form:

$$\min_{\mathsf{V}'} \sum_{k \in \mathsf{V}} E_R(\mathsf{v}_k, \mathsf{v}'_k) + \lambda a_k \|\hat{n}_k(\mathsf{V}') - \mathsf{t}_k\|_2^2, \tag{6.1}$$

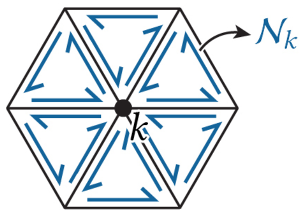
where E_R denotes a regularization energy to preserve the details of the input mesh, and the second part measures the squared distance from the output unit surface normal $\hat{\mathsf{n}}_k(\mathsf{V}')$ to the target output normal t_k at vertex k. We use a_k to denote the Voronoi area of the vertex k, λ is a weighting parameter to control the balance between the two terms, and v_k (v'_k) is the input (output) location of vertex k. In Fig. 6.9, we can observe that using a small λ, the method preserves the input shape B. Using a bigger λ, the method favors deforming the shape more into the style of A'. The choice of E_R depends on the user's intent. One can apply different regularizations to obtain different results. For the purposes of this exposition, we introduce our optimization based on ARAP regularization in Sec. 6.3.4. We discuss how to extend to other regularizations in Sec. 6.4.1.

6.3.4 Normal-Driven Optimization with ARAP

We use $\mathsf{e}_{ij} := \mathsf{v}_j - \mathsf{v}_i \in \mathbb{R}^3$ to denote the edge vector between vertices i, j on the original mesh, and $\mathsf{e}'_{ij} := \mathsf{v}'_j - \mathsf{v}'_i$ for the edge vectors on the deformed mesh. We can rewrite the energy as

$$\min_{\mathsf{V}',\mathsf{R}} \sum_{k\in V} \underbrace{\sum_{\{i,j\}\in\mathcal{N}_k} w_{ij}\|\mathsf{R}_k\mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2}_{E_{\text{ARAP}}} + \lambda a_k \|\hat{n}_k(\mathsf{V}') - \mathsf{t}_k\|_2^2, \tag{6.2}$$

We use $\mathcal{N}_k$ to denote the edge vectors of the *spokes and rims* at vertex k (see the inset) [Chao et al., 2010], $\mathsf{R}_k \in SO(3)$ to denote a 3-by-3 rotation matrix defined k, and w_{ij} is the cotangent weight of edge i, j [Pinkall and Polthier, 1993]. However, this energy is difficult to optimize because the term $\hat{n}_k(\mathsf{V}')$ is non-linear in V'.



We adapt the observation made in [Liu and Jacobson, 2019a] that the space of unit vectors can be captured by rotations. Thus, we can perform a change of variables by replacing $\hat{n}_k(\mathsf{V}')$ with the *rotated* unit normal of the input mesh $\mathsf{R}_k\hat{\mathsf{n}}_k$ as

$$\boxed{\min_{\mathsf{V}',\mathsf{R}} \sum_{k\in V} \sum_{i,j\in\mathcal{N}_k} w_{ij}\|\mathsf{R}_k\mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2 + \lambda a_k \|\mathsf{R}_k\hat{\mathsf{n}}_k - \mathsf{t}_k\|_2^2,} \tag{6.3}$$

where $\hat{\mathsf{n}}_k$ is the kth unit vertex normal of the input mesh computed via area-weighted average of face normals, which is constant throughout the optimization. This $\mathsf{R}_k\hat{\mathsf{n}}_k$ can be perceived as an approximation of the area-weighted vertex normals of the output mesh $\hat{n}_k(\mathsf{V}')$. In Fig. 6.10, we visualize the difference between the output normals $\hat{n}_k(\mathsf{V}')$ and the rotated input normals $\mathsf{R}_k\hat{\mathsf{n}}_k$. We can notice that $\mathsf{R}_k\hat{\mathsf{n}}_k$ is a decent approximation of the output vertex normals computed via area-weighted average. We can observe that error tends to concentrate at high-curvature regions because discrete vertex normals are less accurate along on those regions and the ARAP regularization encourages smooth deformation. This change of variables allows us to solve for the R_ks in parallel and make this energy quadratic in V'. In addition, the fact that R_k is shared across the ARAP term and the normal term enables us to jointly consider both the regularization and the normal terms when obtaining the deformed vertex locations V'.

We minimize this energy via the local/global strategy [Sorkine and Alexa, 2007], where the local step involves solving a set of small *Orthogonal Procrustes* problems and the global step amounts to a linear solve. For the sake of reproducibility, we reiterate the local-global steps for our energy in App. 6.7.1 and 6.7.2. Non-linear methods, such as Newton's method, could be applied to our scenario. However, Newton's method is far slower than the local-global optimization since a single iteration of Newton's method could be more expensive

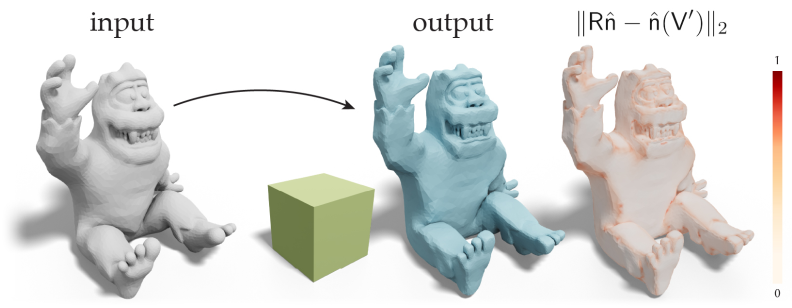

input
output
$\|\mathsf{R}\hat{\mathsf{n}} - \hat{\mathsf{n}}(\mathsf{V}')\|_2$
1
0

Figure 6.10: Empirically, we show that rotated input normals $\mathsf{R}_k\hat{\mathsf{n}}_k$ is a good approximation of the area-weighted output vertex normals $\hat{n}_k(\mathsf{V}')$. We can observe that the error mostly occurs on the high-curvature regions (right). ©Proto Paradigm under CC BY.

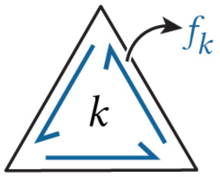
than 100 iterations of the local-global iterations (see [Liu et al., 2017d]). Thus, it is less suitable for our interactive applications. Further accelerating our solver using other optimization methods (e.g., [Kovalsky et al., 2016; Peng et al., 2018; Zhu et al., 2018]) should be possible, but is left as future work.

6.4 Extensions & Analysis

In this section, we introduce extensions to different regularizations and show how to handle cases where target normals $T(B')$ are a function of output geometry.

6.4.1 Different Regularizations

In addition to E_{ARAP}, the normal-driven optimization supports different regularization energies for different modeling intents. One could use ARAP when the goal is to produce a smooth deformation that preserves surface details. If one wants to produce a non-smooth deformation (e.g., sharp creases) while preserving local rigidity, one could instead use a *face-only* ARAP energy E_{FARAP} discussed in [Zhao and Gortler, 2016; Levi and Gotsman, 2015] which consists of only the membrane term. If one is interested in preserving the textures and allowing local scaling, one could use an *as-conformal-as-possible* energy E_{ACAP} [Bouaziz et al., 2012].



Face-only ARAP. The core idea is to remove the bending term from ARAP and only measure the membrane term [Terzopoulos et al., 1987], so that two adjacent triangles can bend freely. We achieve this by applying the idea from [Zhao and Gortler, 2016; Levi and Gotsman, 2015] which only measures the ARAP energy over the three edge vectors of a face f_k (see the inset), instead of the spokes and rims $\mathcal{N}_k$. Precisely, we can write this

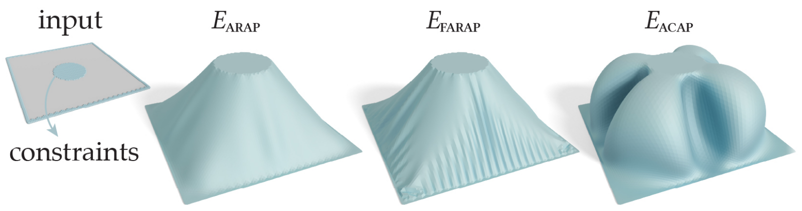


Figure 6.11: Different regularizations favor different behaviors. Given a sheet (gray), we pull up the center part (central blue dots) and shrink the boundary (blue dots on the boundary), then we minimize each regularization energy to determine the unconstrained vertices. We can observe that E_{ARAP} favors rigid and smooth interpolation, E_{FARAP} favors sharp bending between triangles, and E_{ACAP} favors to preserve angles while allowing local scaling.

"face-only" ARAP regularization E_{FARAP} as

$$E_{\text{FARAP}}(\mathsf{V}', \mathsf{R}) = \sum_{k \in F} \sum_{\{i,j\} \in f_k} w_{ij} \|\mathsf{R}_k \mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2, \tag{6.4}$$

As-conformal-as-possible. If the goal is to create novel geometric details, it is crucial to allow non-rigid deformations. However, an arbitrary deformation may lead to undesirable behaviors, such as badly shaped triangles. Thus constraining the angle preservation, a.k.a. conformality, a suitable regularization. Specifically, we use the ACAP energy E_{ACAP} in [Bouaziz et al., 2012] as our regularization

$$E_{\text{ACAP}}(\mathsf{V}', \mathsf{R}, \mathsf{s}) = \sum_{k \in F} \sum_{i,j \in \mathcal{N}_k} w_{ij} \|s_k \mathsf{R}_k \mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2, \tag{6.5}$$

where s_k is a scalar representing the scaling of local patch. One can compute the optimal s_k analytically via the method by Schönemann and Carroll [1970] (see App. 6.7.3).

Deploying these regularizations $E_{\text{FARAP}}, E_{\text{ACAP}}$ requires only a few changes in the optimization steps. Deploying E_{FARAP} only involves changing the incidence matrix. Deploying E_{ACAP} only requires adding one more line of code in the local step to solve an *isotropic orthogonal Procrustes* problem [Schönemann and Carroll, 1970]. We detail such changes in App. 6.7.3. In Fig. 6.11, we apply the same deformation to a sheet but with different regularizations. We can see that different regularizations favor drastically different solutions.

Our framework allows one to easily plug-and-play different regularizations. Specifically, we use E_{ARAP} for applications that favor smooth deformation (e.g., Fig. 6.13), E_{FARAP} for creating sharp creases (Fig. 6.18, 6.20), and E_{ACAP} when one wants to manipulate geometric details such as in Fig. 6.22.

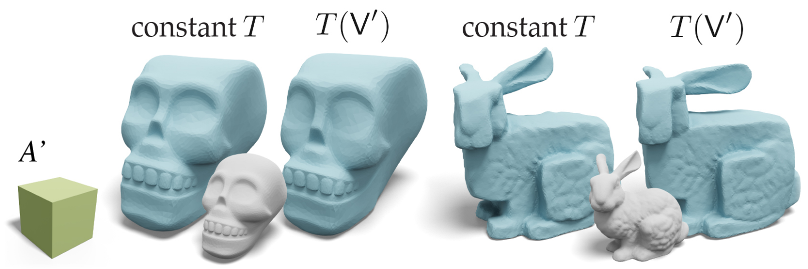


Figure 6.12: Setting the target normal T as a constant or treating it as a function of the output mesh $T(B')$ leads to different local minima. In many cases (left pair), both options lead to similar looking results. But setting T as a constant may result in an undesirable local minimum in some cases (right), such as the ears of the bunny.

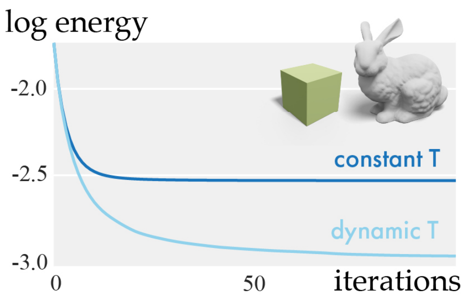
Algorithm 5: Normal-driven optimization

```
Input  : A triangle mesh $\mathsf{V}, \mathsf{F}$ and a weight $\lambda$
Output: Deformed vertex positions $\mathsf{V}'$
1. compute $\widetilde{N}_{A'}$                                   // step 1, Sec. 6.3.1
2. compute $T$ from $\hat{n}(\mathsf{V})$                          // step 2, Sec. 6.3.2
3. $\hat{\mathsf{n}} \leftarrow \hat{n}(\mathsf{V})$               // compute input surface normals
4. $\mathsf{Q}, \mathsf{K} \leftarrow precompute(\mathsf{V}, \mathsf{F})$   // see Sec. 6.7.1
5. $\mathsf{V}' \leftarrow \mathsf{V}$
6. while not converge do
7.     $\mathsf{R} \leftarrow local\text{-}step\ (\mathsf{V}', \hat{\mathsf{n}}, T, \lambda)$   // Sec. 6.7.1
8.     $\mathsf{V}' \leftarrow global\text{-}step\ (\mathsf{R}, \mathsf{Q}, \mathsf{K})$      // Sec. 6.7.2
9.     compute $T$ from $\hat{n}(\mathsf{V}')$                     // (optional) for dynamic $T$
```

6.4.2 Dynamic Target Normals

Our method converges to a local minimum. Empirically, we observe that fixing the target normal T throughout the optimization may work well perceptually in many cases (see the left pair in Fig. 6.12). However, a fixed T may lead to an undesirable local minimum due to a sub-optimal assignment of T (see the right pair in Fig. 6.12). Inspired by Projective Dynamics [Bouaziz et al., 2014], a simple solution to avoid such local minima is to treat T as a function of B' ($N_{B'}$ specifically), and update T at every iteration. We summarize the pseudo code in Alg. 5. If T is a constant throughout the optimization, one can simply skip the optional step at line 9.

In terms of convergence, in the case where T is constant, the convergence behaves the same as the original ARAP [Sorkine and Alexa, 2007], where the energy decreases monotonically. In the case where T is dependent to B', we do not guarantee a monotonic decrease in energy, but the optimization still converges in our experiments. In the inset, we vi-



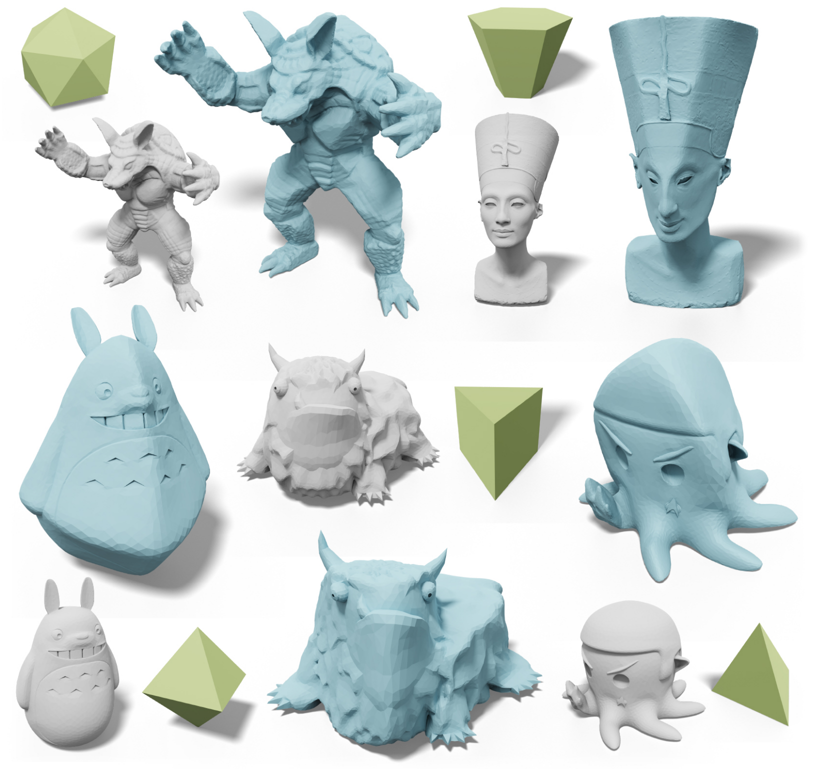
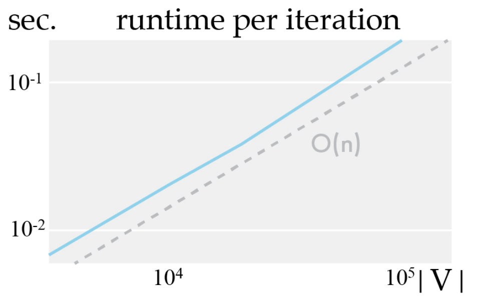


Figure 6.13: Given an input shape (gray), our approach can transfer the style of a primitive shape (green) to obtain a stylized output shape (blue). ©Johannes (bottom left), Joseph Larson (bottom middle), and Angelo Tartanian (bottom left) under CC BY. The Nefertiti mesh (top right) was scanned by Nora Al-Badri and Jan Nikolai Nelles from the Nefertiti bust.

sualize the convergence plot for the examples in Fig. 6.12.

We implement our algorithm in C++ using Eigen [Guennebaud et al., 2010] and evaluate our method on a MacBook Pro with an Intel i5 2.3GHz processor. Our method runs at 24 iterations per second for a mesh with around 20k vertices. We report a complete picture of our runtime in the inset. The local step is the computation bottleneck for meshes with less than 20k vertices, but further acceleration can be achieved via the method by Zhang et al. [2021]. Typically, within the first 10 iterations, our method can achieve a visually similar result compared to the converged solution. This property enables us to build an interactive tool for users to experiment with different style shapes A' or artistic controls.



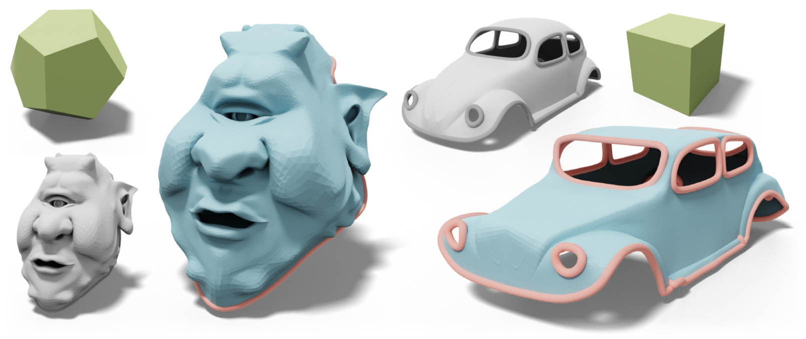


Figure 6.14: Even for input shapes with boundaries (gray), our method is still applicable to transfer the style of primitive shapes (green) to obtain the stylized output shape (blue).

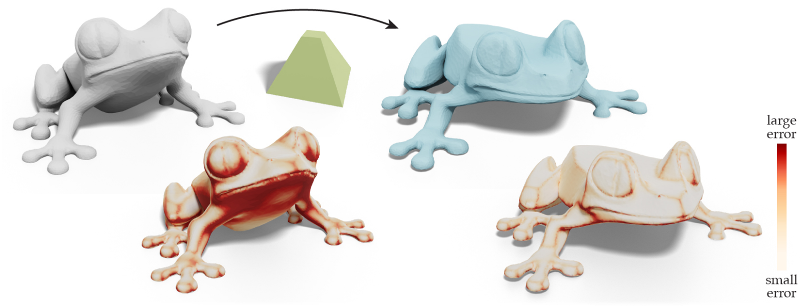


Figure 6.15: We visualize the difference between the mesh normals and the normals of the style shape. Our normal-driven optimization effectively reduce the difference to the target normals. ©Morena Protti under CC BY.

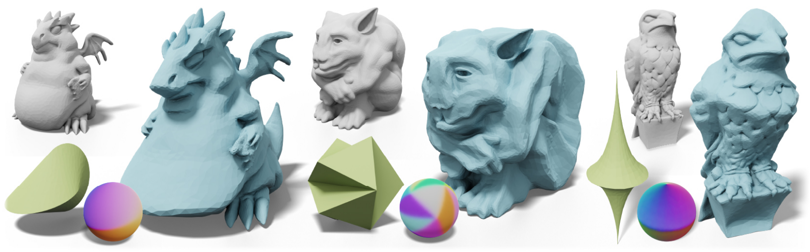


Figure 6.16: When the input shape is smooth or non-convex, we use the mean curvature flow (see Fig. 6.8) to obtain target normals to proceed the optimization. We deform the input shapes (gray) to exhibit the style of an oloid (left, green), a Jessen's icosahedron (middle, green), and a tractricoid (right, green), respectively. ©Splotchy Ink (left), fong182 (middle), and Colin Freeman (right) under CC BY.

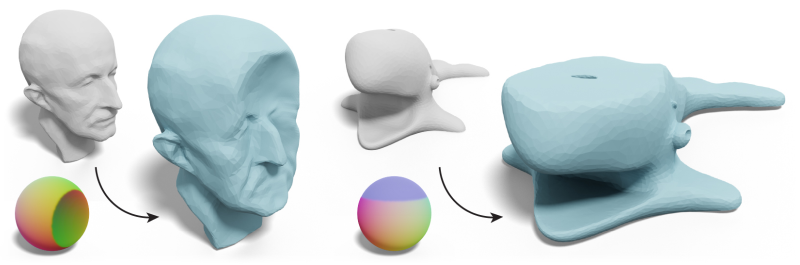


Figure 6.17: One can manually specify the target normals on a sphere (Normal Captures) for full control, and deform the input shape (gray) to the style (blue) prescribed by the colored sphere. ©MakerBot (right) under CC BY.

6.5 Applications

The major benefit of our analogy-based stylization method is that one can plug-and-play different style shapes to obtain different results. When one provides convex primitives with few distinct face normals, we can simply use the method discussed in Sec. 6.3.1 to turn an input shape into the style of the primitive (see Fig. 6.13, 6.14). In Fig. 6.15, we also quantitatively show that our method can effectively reduce the difference between the mesh normals and the normals of a primitive. If the provided style shape is smooth or non-convex, where the simple closest normal may fail to capture the style, one could use a spherical parameterization described in Sec. 6.3.1 to achieve the stylization. If desiring more user controls, one could "paint" the desired surface normals on a unit sphere (see Sec. 6.3.1), and then transfer the style of the painted normals directly to the input (see Fig. 6.17)

6.5.1 PolyCube Deformation

If one is interested in PolyCube maps [Tarini et al., 2004], we can adapt normal driven editing to create PolyCube maps, following the observation in [Zhao et al., 2017]. Specifically, we need to use a cube as a style shape and move the pre-computation step in Alg. 5 to the optimization loop. Moving the pre-computation in the loop would no longer preserve the original details, which is desirable for creating PolyCube shapes. This modification may also lead to badly shaped triangle. When these faces appear, a quick solution is to move the vertex towards the 1-ring average by a small amount to improve triangle quality. For the sake of comparison, we use the same PolyCube segmentation as in [Fu et al., 2016] and show that we can achieve comparable results in Fig. 6.2. We can further generalize the PolyCube map to other polygonal boxes by specifying non-cube normals (see Fig. 6.18).

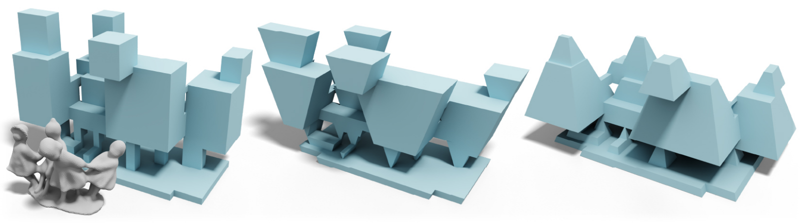


Figure 6.18: By using different sets of normals, we can generalize the PolyCube method (left) to create polygonal boxed maps.

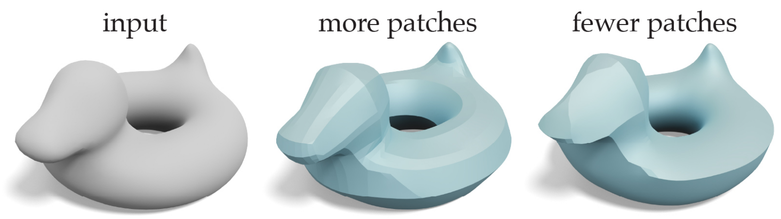


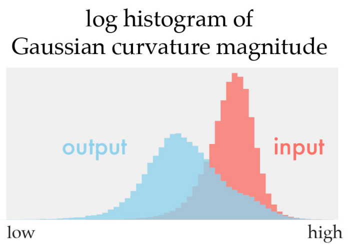
Figure 6.19: Stein et al. [Stein et al., 2018a] control the patches on the developable surfaces via remeshing the input. We, instead, can control the amount of creases (middle, right) by tuning a single parameter (see App. 6.7.5).

6.5.2 Developable Surface Approximation

So far we have only considered an explicit shape or a set of painted normals as our style shape. Here we further extend our method to support an energy that describes a certain style. In particular, we consider the case when the target normal T is computed via an optimization

$$T = \arg\min_{T} f(B'), \tag{6.6}$$

and, similar to the case where T is dependent on B', we update T at every iteration in the local/global solve.



We evaluate this extension via setting f to be the discrete developability energy proposed in [Stein et al., 2018a], with details provided in App. 6.7.5. Compared to the original method, our approach contains a regularization term in addition to the developable energy, thus our optimization requires no remeshing and results in the faster optimization (see Fig. 6.5). In Fig. 6.19, we further show that our framework enables one to control the number of creases in the results. With our framework one can interactively create a variety of piece-wise developable shapes (see Fig. 6.20). In Fig. 6.21, we evaluate our results by visualizing the discrete Gaussian curvature before and after running our developable flow. We can observe that the Gaussian curvature which concentrates along the creases and results in a piece-wise developable sur-

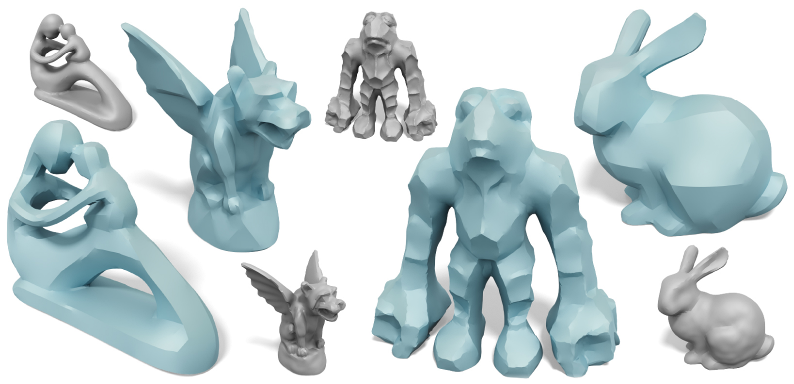


Figure 6.20: Our normal driven editing can be used to create piece-wise developable surfaces (blue). Our method requires no remeshing and is fast enough for interactive modeling. ©cerberus333 (third) under CC BY-NC.

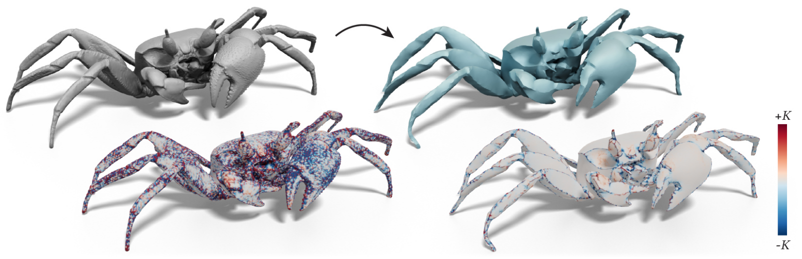


Figure 6.21: We use our normal driven editing to deform the input shape (gray) into a piece-wise developable approximation (blue). In the bottom row, we visualize the Gaussian curvature concentrates on the creases after the deformation, leading to a piece-wise developable shape. ©Oliver Laric under CC BY-NC-SA.

face. In the inset, we quantitatively demonstrate that our method effectively increases the developability of the mesh in Fig. 6.21.

6.6 Limitations & Future Work

Our method draws inspiration from Projective Dynamics [Bouaziz et al., 2014] to handle the case where target normals T are a function of output shape B' (e.g., Fig. 6.12, 6.20). Although fast and suitable for our intended interactive applications, our method often struggles to converge to a highly accurate solution. Extending our optimization to, for example, Newton's method would be desirable for applications that desire highly accurate solutions.

Our approach is restricted to a sphere as our reference shape A, and uses the Gauss map to determine the correspondences between A and the input B. As the Gauss map purely relies on surface normals to determine the map, the resulting correspondences to the unit sphere is ignorant to area distortion. This characteristic is

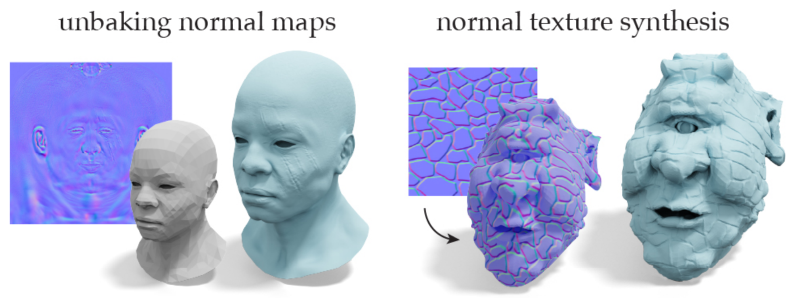


Figure 6.22: If one is interested in creating high-frequency geometric textures, we recommend target normals via texture synthesis and then optimizing the geometry via normal-driven optimization. We demonstrate an example of "unbaking" normal maps (left) and an example of geometric texture synthesis (right).

beneficial for handling input shapes B that are very different (e.g., different genus) from a sphere because in these cases it is challenging to obtain a map with low area distortion. However, the price we have to pay is that we cannot support structured and high-frequency patterns (e.g., geometric texture synthesis). Thus, if one is interested in stylizing shapes with detailed textures, we suggest to first synthesize target normals on the surface directly [Wei et al., 2009] then perform the normal-driven optimization (Sec. 6.3.4). In Fig. 6.22 we demonstrate this alternative by unbaking an existing normal map for manufacturing purposes (see the inset) and synthesizing normal textures from an image.

Our method currently supports manifold triangle meshes. Extending to non-manifold meshes, polygon meshes, volumetric meshes, and point clouds could be beneficial for handling real-world geometric data. Not every shape or normal capture sphere is valid to serve as a style shape of our algorithm (e.g., a normal capture sphere with inside-out normals). Discovering the validity of a style shape is important to understand the behavior of these novel modeling methods. Removing the assumption about the source shape being a sphere could lead to a more general analogy-based shape editing. Based on the observation that surface normals are a promising geometric quantity to capture the style of a shape. Developing a better categorization of styles based on normals or exploring learning-based techniques on normals (instead of vertices) could lead to novel stylization methods.

6.7 Appendix

6.7.1 Local Step with E_{ARAP}

Given a fixed V', we obtain the optimal rotation for each vertex k by solving the following minimization problem

$$\mathsf{R}_k = \underset{\mathsf{R}_k \in SO(3)}{\arg\min} \sum_{i,j \in \mathcal{N}_k} w_{ij} \|\mathsf{R}_k \mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2 + \lambda a_k \|\mathsf{R}_k \hat{\mathsf{n}}_k - \mathsf{t}_k\|_2^2$$

The above optimization is an instance of the *orthogonal Procrustes* which finds the best rotation matrix R_k to map a set of vectors $(\mathsf{e}_{ij}, \hat{\mathsf{n}}_k)$ to another set of vectors $(\mathsf{e}'_{ij}, \mathsf{t}_k)$. We can re-write it into a more compact expression as:

$$\mathsf{R}_k^\star = \underset{\mathsf{R}_k \in SO(3)}{\arg\max}\ \mathrm{Tr}(\mathsf{R}_k \mathsf{X}_k) \tag{6.7}$$

$$\mathsf{X}_k = \begin{bmatrix} \mathsf{E}_k & \hat{\mathsf{n}}_k \end{bmatrix} \begin{bmatrix} \mathsf{W}_k & \\ & \lambda a_k \end{bmatrix} \begin{bmatrix} \mathsf{E}'^{\top}_k \\ \mathsf{t}_k^{\top} \end{bmatrix}. \tag{6.8}$$

where W_k is a $|\mathcal{N}_k|$-by-$|\mathcal{N}_k|$ diagonal matrix of the cotangent weights w_{ij}, E_k and E'_k are 3-by-$|\mathcal{N}_k|$ matrices concatenating the edge vectors of the face one-ring at the rest and deformed states, respectively. One can then derive the optimal R_k from the SVD of $\mathsf{X}_k = \mathcal{U}_k \sum_k \mathcal{V}_k^\top$

$$\mathsf{R}_k = \mathcal{V}_k \mathcal{U}_k^\top, \tag{6.9}$$

up to changing the sign of the column of $\mathcal{U}_k$ so that $\det(\mathsf{R}_k) > 0$.

6.7.2 Global Step with E_{ARAP}

The global step updates the deformed vertex positions V' from a fixed set of rotations R obtained via the local step. This boils down to solving the following problem

$$\mathsf{V}'^\star = \underset{\mathsf{V}'}{\arg\min} \sum_{k \in V} \sum_{i,j \in \mathcal{N}_k} w_{ij} \|\mathsf{R}_k \mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2$$

We can expand this energy as

$$\begin{aligned} &\sum_{k \in V} \sum_{i,j \in \mathcal{N}_k} w_{ij} \|\mathsf{R}_k \mathsf{e}_{ij} - \mathsf{e}'_{ij}\|_2^2 \\ &\quad = \sum_{k \in V} \sum_{i,j \in f_k} w_{ij} \mathsf{e}'^{\top}_{ij} \mathsf{e}'_{ij} - 2 w_{ij} \mathsf{e}'^{\top}_{ij} \mathsf{R}_k \mathsf{e}_{ij} + \text{constant} \end{aligned}$$

It is often convenient to express the summation in terms of matrices. We introduce a directed incidence matrix A_k with size $|V|$-by-$|\mathcal{N}_k|$ to represent the edge vectors in $\mathcal{N}_k$ as $\mathsf{V}^\top \mathsf{A}_k$, and we use M_k to represent a $|\mathcal{N}_k|$-by-$|\mathcal{N}_k|$ diagonal matrix of the weights w_{ij}. Then we can re-write the energy in terms of matrices as

$$\begin{aligned}
&\sum_{k\in V} \operatorname{Tr}(\mathsf{M}_k\mathsf{A}_k^\top\mathsf{V}'\mathsf{V}'^\top\mathsf{A}_k) - 2\operatorname{Tr}(\mathsf{M}_k\mathsf{A}_k^\top\mathsf{V}'\mathsf{R}_k\mathsf{V}^\top\mathsf{A}_k)\\
&= \sum_{k\in V} \operatorname{Tr}(\mathsf{V}'^\top\mathsf{A}_k\mathsf{M}_k\mathsf{A}_k^\top\mathsf{V}') - 2\operatorname{Tr}(\mathsf{R}_k\mathsf{V}^\top\mathsf{A}_k\mathsf{M}_k\mathsf{A}_k^\top\mathsf{V}')\\
&= \operatorname{Tr}\left(\mathsf{V}'^\top\left(\textstyle\sum_k \mathsf{A}_k\mathsf{M}_k\mathsf{A}_k^\top\right)\mathsf{V}'\right) - 2\operatorname{Tr}\left(\left(\textstyle\sum_k \mathsf{R}_k\mathsf{V}^\top\mathsf{A}_k\mathsf{M}_k\mathsf{A}_k^\top\right)\mathsf{V}'\right)\\
&= \operatorname{Tr}(\mathsf{V}'^\top\mathsf{Q}\mathsf{V}') - 2\operatorname{Tr}(\mathsf{R}\mathsf{K}\mathsf{V}'),
\end{aligned}\tag{6.10}$$

where $\mathsf{R} = \{\mathsf{R}_k\}$ is the concatenation of all the rotations, Q is a $|V|$-by-$|V|$ symmetric matrix, and K is a $|9V|$-by-$|3V|$ matrix stacking the constant terms which can be computed during the precomputation. We can then find the optimal V' by solving a linear system

$$\mathsf{Q}\mathsf{V}' = \mathsf{K}^\top\mathsf{R}^\top$$

As we know from [Sorkine and Alexa, 2007], Q is the cotangent Laplacian [Pinkall and Polthier, 1993]. We can pre-factorize Q to speed up runtime performance. With these pieces in hand, we can minimize our energy Eq. (6.3) by iteratively performing the local and the global steps (see Alg. 5).

6.7.3 Generalization to E_{FARAP} and E_{ACAP}

Changing the regularization from E_{ARAP} to the membrane-only regularization E_{FARAP} (Eq. (6.4)) requires re-defining R on each face and changing the set of edge vectors to the three edge vectors of a triangle. These changes would lead us to replace the $\mathsf{E}_k, \mathsf{E'}_k$ in the local step Eq. (6.7) to the three edge vectors of a face, and a_k to the face area. In the global step, one only needs to update the incidence matrix A_k in Eq. (6.10) to a $|V|$-by-$|f_k|$ matrix containing the three edge vectors information.

Deploying the as-conformal-as-possible regularization E_{ACAP} (Eq. (6.5)) changes the local step to solve an instance of the isotropic orthogonal Procrustes problem, where an analytical solution has been derived in [Schönemann and Carroll, 1970]. In short, one can obtain the optimal rotation as in Eq. (6.9), and compute the optimal scaling s_k analytically as

$$s_k = \frac{\operatorname{Tr}(\mathsf{W}_k\mathsf{E'}_k^\top\mathsf{R}_k\mathsf{E}_k) + \lambda a_k\hat{\mathsf{n}}_k^\top\mathsf{t}_k}{\operatorname{Tr}(\mathsf{W}_k\mathsf{E}_k^\top\mathsf{E}_k) + \lambda a_k\hat{\mathsf{n}}_k^\top\mathsf{t}_k}.$$

When assembling the matrices for the global step, using E_{ACAP} would require replacing R_k with $s_k\mathsf{R}_k$.

6.7.4 Projective Dynamics for Dynamic Target Normals

We draw inspiration from *projective dynamics* [Bouaziz et al., 2014] to handle cases where the target normal T is a function of output geometry B'. Let us first define

$$T = \arg\min E_N(\mathsf{V}')$$

as a minimizer of an energy E_N defined on the output shape. In our cases, E_N could be the distance to the closet normals or the developable energy [Stein et al., 2018a]. With this definition, we re-write Eq. (6.3) as

$$\begin{aligned} &\min_{\mathsf{V}',\mathsf{R}} \sum_{k \in V} \sum_{i,j \in \mathcal{N}_k} w_{ij} \|\mathsf{R}_k \mathsf{e}_{ij} - \mathsf{e'}_{ij}\|_2^2 + \lambda a_k \|\mathsf{R}_k \hat{\mathsf{n}}_k - \mathsf{t}_k\|_2^2, \\ &\text{subject to } T = \arg\min E_N(\mathsf{V}') \end{aligned}$$

This reformulation allows us to directly deploy the projective dynamics solver by first projecting $T = \{\mathsf{t}_k\}$ to the "constraint" E_N, fixing t_k, and solving the original problem as Eq. (6.3) via the local/global solver to get V' at the next iteration. We then iterate this procedure (see Alg. 5) until convergence. This expression enables us to plug-and-play different E_N for different modeling objectives.

6.7.5 Normal Driven Developable Surfaces

Our normal-driven editing can be used to create developable surfaces by specifying a set of target normals that are developable. Stein et al. [Stein et al., 2018a] propose a characterization of discrete developability based on face normals of a vertex one-ring. In short, if all the one-ring face normals correspond to a common plane or two planes, then this local one-ring is piecewise developable.

With this characterization, we can easily get a set of "developable" face normals by (1) visiting all the one-ring faces of a vertex, (2) performing a small principle component analysis on the face normals for each one-ring, and (3) projecting the normals to one or two common planes by zeroing out the components correspond to the smallest eigenvalues. By using a different threshold to decide whether to zero out the smallest or the smallest two components, we can control the amount of creases in the developable approximation (see Fig. 6.19). As each face will receive three (possibly) different developable normals from the previous procedure, we simply average them to get the target face normals. We perform this developable normal computation at each iteration in parallel, which corresponds to the Line 9 of Alg. 5.

Chapter 7

Spectral Coarsening of Geometric Operators

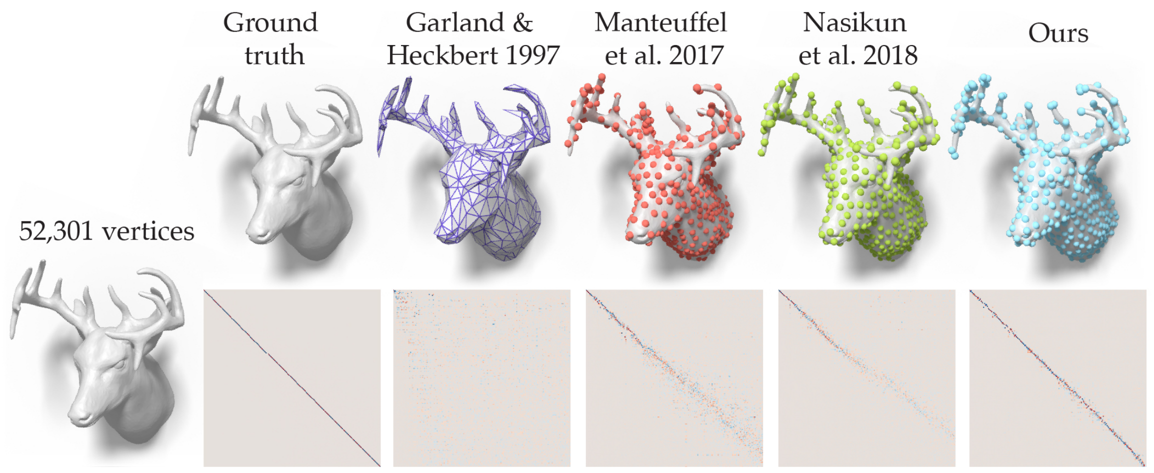
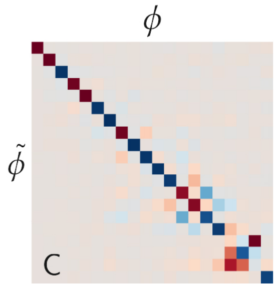


Figure 7.1: There are many ways to coarsen a 52,301×52,301 sparse anisotropic Laplace matrix down to a sparse 500×500 matrix: simplify the mesh [Garland and Heckbert, 1997] and rediscretize; apply algebraic multigrid coarsening [Manteuffel et al., 2017]; or approximate using radial-basis functions [Nasikun et al., 2018]. We introduce a way to measure how well the coarse operator maintains the original operator's eigenvectors (bottom row). The visualization shows deviation from a diagonal matrix indicating poor eigenvector preservation. In response, we introduce an optimization to coarsen geometric operators while preserving eigenvectors and maintaining sparsity and positive semi-definiteness.

We introduce a novel approach to measure the behavior of a geometric operator before and after coarsening. By comparing eigenvectors of the input operator and its coarsened counterpart, we can quantitatively and visually analyze how well the spectral properties of the operator are maintained. Using this measure, we show that standard mesh simplification and algebraic coarsening techniques fail to maintain spectral properties. In response, we introduce a novel approach for *spectral coarsening*. We show that it is possible to significantly reduce the sampling density of an operator derived from a 3D shape without affecting the low-frequency eigenvectors. By marrying techniques developed within the algebraic multigrid and the functional maps literatures, we successfully coarsen a variety of isotropic and anisotropic operators while maintaining sparsity and positive semi-definiteness. We demonstrate the utility of this approach for applications including operator-sensitive sampling, shape matching, and graph pooling for convolutional neural networks.

7.1 Introduction

Geometry processing relies heavily on building matrices to represent linear operators defined on geometric domains. While typically sparse, these matrices are often too large to work with efficiently when defined over high resolution representations. A common solution is to simplify or coarsen the domain. However, matrices built from coarse representations often do not behave the same way as their fine counterparts leading to inaccurate results and artifacts when resolution is restored. Quantifying and categorizing *how* this behavior is different is not straightforward and most often coarsening is achieved through operator-oblivious remeshing. The common appearance-based or geometric metrics employed by remeshers, such as the classical quadratic error metric [Garland and Heckbert, 1997] can have very little correlation with maintenance of operator behavior.



We propose a novel way to compare the spectral properties of a discrete operator before and after coarsening, and to guide the coarsening to preserve them. Our method is motivated by the recent success of spectral methods in shape analysis and processing tasks, such as shape comparison and non-rigid shape matching, symmetry detection, and vector field design to name a few. These methods exploit eigenfunctions of various operators, including the Laplace-Beltrami operator, whose eigenfunctions can be seen as a generalization of the Fourier basis to curved surfaces. Thus, spectral methods expand the powerful tools from Fourier analysis to more general domains such as shapes, represented as triangle meshes in 3D. We propose to measure how well the eigenvectors (and by extension eigenvalues) of a matrix $\mathsf{L} \in \mathbb{R}^{n\times n}$ on the high-resolution domain are maintained by its coarsened counterpart $\widetilde{\mathsf{L}} \in \mathbb{R}^{m\times m}$ $(m < n)$ by computing a dense matrix $\mathsf{C}^{k\times k}$, defined as the inner product of the first k eigenvectors $\Phi \in \mathbb{R}^{n\times k}$ and $\widetilde{\Phi} \in \mathbb{R}^{m\times k}$ of L and $\widetilde{\mathsf{L}}$ respectively:

$$\mathsf{C} = \widetilde{\Phi}^\top \widetilde{\mathsf{M}} \mathsf{P} \Phi, \tag{7.1}$$

where $\widetilde{\mathsf{M}} \in \mathbb{R}^{m\times m}$ defines a mass-matrix on the coarse domain and $\mathsf{P} \in \mathbb{R}^{m\times n}$ is a restriction operator from fine to coarse. The closer C resembles the identity matrix the more the eigenvectors of the two operators before and after coarsening are similar.

We show through a variety of examples that existing geometric and algebraic coarsening methods fail to varying degrees to preserve the eigenvectors and the eigenvalues of common operators used in geometry processing (see Fig. 7.1 and Fig. 7.2).

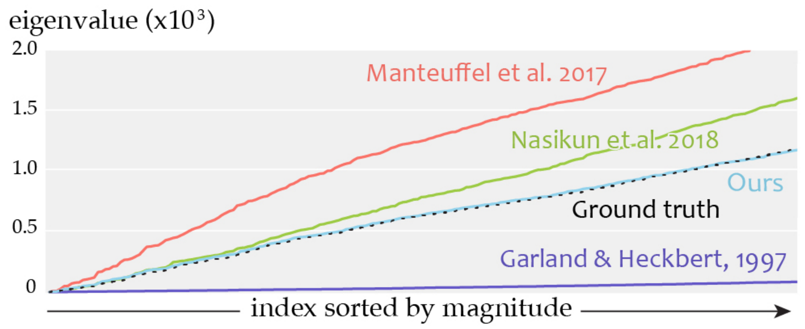


Figure 7.2: Our coarsening directly preserves eigenvectors so eigenvalues are also implicitly preserved: eigenvalue plot of Fig. 7.1.

In response, we propose a novel coarsening method that achieves much better preservation under this new metric. We present an optimization strategy to coarsen an input positive semi-definite matrix in a way that better maintains its eigenvectors (see Fig. 7.1, right) while preserving matrix sparsity and semi-definiteness. Our optimization is designed for operators occurring in geometry processing and computer graphics, but does not rely on access to a geometric mesh: our input is the matrix L, and an area measure M on the fine domain, allowing us to deal with non-uniform sampling. The output coarsened operator $\widetilde{\mathsf{L}}$ and an area measure $\widetilde{\mathsf{M}}$ on the coarse domain are defined for a subset of the input elements chosen carefully to respect anisotropy and irregular mass distribution defined by the input operator. The coarsened operator is optimized via a novel formulation of coarsening as a sparse semi-definite programming optimization based on the operator commutativity diagram.

We demonstrate the effectiveness of our method at categorizing the failure of existing methods to maintain eigenvectors on a number of different examples of geometric domains including triangle meshes, volumetric tetrahedral meshes and point clouds. In direct comparisons, we show examples of successful spectral coarsening for isotropic and anisotropic operators. Finally, we provide evidence that spectral coarsening can improve downstream applications such as shape matching, graph pooling for graph convolutional neural networks, and data-driven mesh sampling.

7.2 Related Work

Mesh Simplification and Hierarchical Representation The use of multi-resolution shape representations based on mesh simplification has been extensively studied in computer graphics, with most prominent early examples including mesh decimation and optimization approaches [Schroeder et al., 1992; Hoppe et al., 1993] and their multiresolution variants e.g., *progressive meshes* [Hoppe, 1996; Popović and Hoppe, 1997] (see [Cignoni et al., 1998a] for an overview and comparison of a wide range of mesh simplification methods). Among these classical techniques, perhaps the best-known and most widely used approach

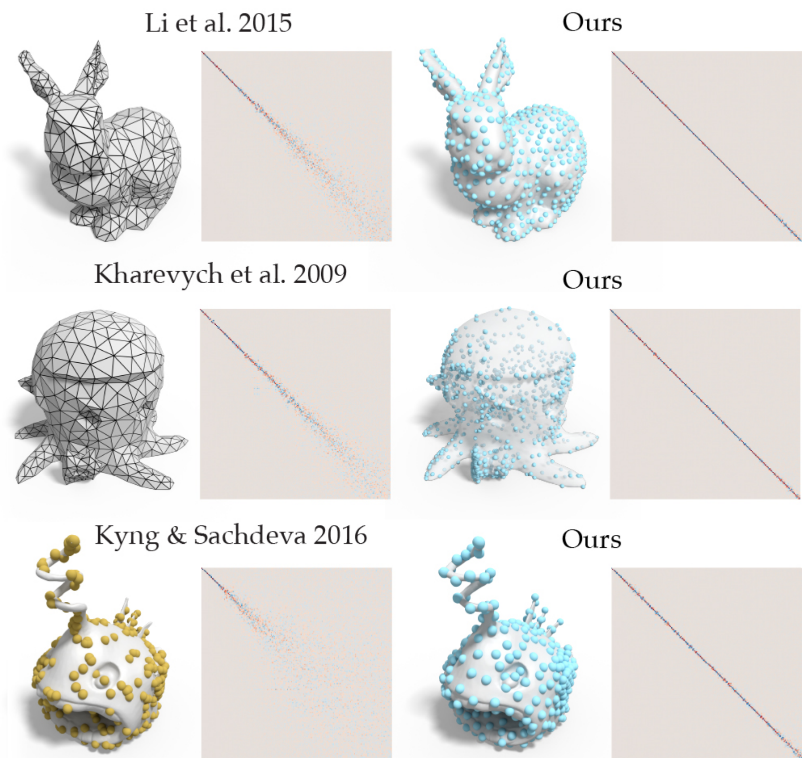


Figure 7.3: As methods of Li et al. [2015]; Kharevych et al. [2009]; Kyng and Sachdeva [2016] are not designed for preserving spectral properties, they only preserve very low frequency eigenvectors (top-left corner of matrix images), but fails for subsequent modes.

is based on the *quadratic error metrics* introduced in [Garland and Heckbert, 1997] and extended significantly in follow-up works to incorporate texture and appearance attributes (e.g., [Garland and Heckbert, 1998; Hoppe, 1999] to name a few). Other, more recent approaches have also included variational shape approximation [Cohen-Steiner et al., 2004] and wavelet-based methods especially prominent in shape compression [Schroder, 1996; Peyré and Mallat, 2005], as well as more flexible multi-resolution approaches such as those based on hybrid meshes [Guskov et al., 2002] among myriad others. Although mesh simplification is a very well-studied problem, the vast majority of approximation techniques is geared towards preservation of shape *appearance* most often formulated via the preservation of local geometric features. Li et al. [2015] conduct a *frequency-adaptive* mesh simplification to better preserve the acoustic transfer of a shape by appending a modal displacement as an extra channel during progressive meshes. In Fig. 7.3, we show that this method fails to preserve all low frequency eigenvectors (since it is designed for a single frequency). Our measure helps to reveal the accuracy of preserving *spectral* quantities, and to demonstrate that existing techniques often fail to achieve this objective.

Numerical Coarsening in Simulation Coarsening the geometry of an elasticity simulation mesh without adjusting the material parameters (e.g., Young's modulus) leads to *nu-*

merical stiffening. Kharevych et al. [2009] recognize this and propose a method to independently adjust the per-tetrahedron elasticity tensor of a coarse mesh to agree with the six smallest deformation modes of a fine-mesh inhomogeneous material object (see Fig. 7.3 for comparison). Chen et al. [2015] extend this idea via a data-driven lookup table. Chen et al. [2018] consider numerical coarsening for regular-grid domains, where matrix-valued basis functions on the coarse domain are optimized to again agree with the six smallest deformation modes of a fine mesh through a global quadratic optimization. To better capture vibrations, Chen et al. [2017a] coarsen regular-grids of homogeneous materials until their low frequency vibration modes exceeding a Hausdorff distance threshold. The ratio of the first eigenvalue before and after coarsening is then used to rescale the coarse materials Young's modulus. In contrast to these methods, our proposed optimization is not restricted to regular grids or limited by adjusting physical parameters directly.

Algebraic Multigrid Traditional multigrid methods coarsen the mesh of the geometric domain recursively to create an efficient iterative solver for large linear systems [Briggs et al., 2000]. For isotropic operators, each geometric level smooths away error at the corresponding frequency level [Burt and Adelson, 1983]. *Algebraic* multigrid (AMG) does not see or store geometric levels, but instead defines a hierarchy of system matrices that attempt to smooth away error according to the input matrix's spectrum [Xu and Zikatanov, 2017]. AMG has been successfully applied for anisotropic problems such as cloth simulation [Tamstorf et al., 2015]. Without access to underlying geometry, AMG methods treat the input sparse matrix as a graph with edges corresponding to non-zeros and build a coarser graph for each level by removing nodes and adding edges according to an *algebraic distance* determined by the input matrix. AMG like all multigrid hierarchies are typically measured according to their solver convergence rates [Xu and Zikatanov, 2017]. While eigenvector preservation is beneficial to AMG, an efficient solver must also avoid adding too many new edges during coarsening (i.e., [Livne and Brandt, 2012; Kahl and Rottmann, 2018]). Meanwhile, to remain competitive with other blackbox solvers, AMG methods also strive to achieve very fast hierarchy construction [Xu and Zikatanov, 2017]. Our analysis shows how state-of-the-art AMG coarsening methods such as [Manteuffel et al., 2017] which is designed for fast convergence fails to preserve eigenvectors and eigenvalues (see Fig. 7.1 and Fig. 7.2). Our optimization formulation in Sec. 7.3.1 and Sec. 7.3.2 is inspired by the "root node" selection and Galerkin projection approaches found in the AMG literature [Stuben, 2000; Bell, 2008; Manteuffel et al., 2017].

Spectrum Preservation In contrast to geometry-based mesh simplification very few methods have been proposed targeting preservation of spectral properties. Öztireli and colleagues [Öztireli et al., 2010] introduced a technique for spectral sampling on surfaces. In a similar spirit to our approach, their method aims to compute samples on a surface that

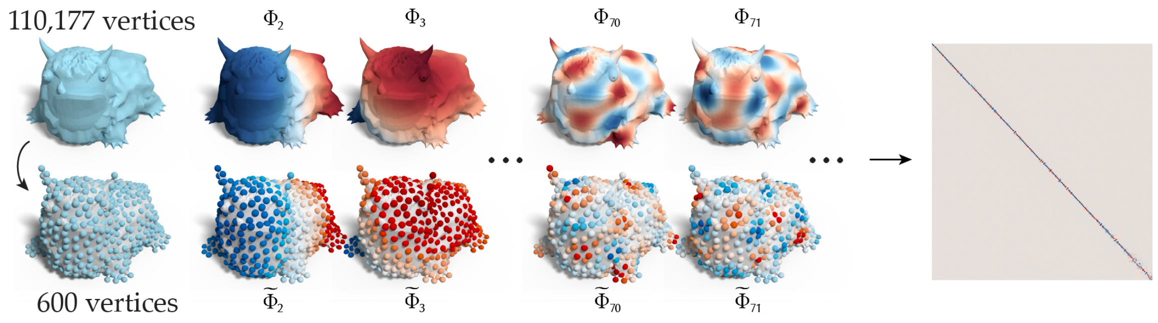


Figure 7.4: When eigenvectors are equivalent (up to sign) before and after coarsening the operator, the matrix C (right) resembles the identity matrix.

can approximate the Laplacian spectrum of the original shape. This method targets only isotropic sampling and is not well-suited to more diverse operators such as the anisotropic Laplace-Beltrami operator handled by our approach. More fundamentally, our goal is to construct a coarse representation that preserves an entire *operator*, and allows, for example, to compute eigenfunctions and eigenvalues in the coarse domain, which is not addressed by a purely sampling-based strategy. More recently, an efficient approach for approximating the Laplace-Beltrami eigenfunctions has been introduced in [Nasikun et al., 2018], based on a combination of fast Poisson sampling and an adapted coarsening strategy. While very efficient, as we show below, this method unfortunately fails to preserve even medium frequencies, especially in the presence of high-curvature shape features or more diverse, including anisotropic Laplacian, operators.

We note briefly that spectrum preservation and optimization has also been considered in the context of sound synthesis, including [Bharaj et al., 2015], and more algebraically for efficient solutions of Laplacian linear systems [Kyng and Sachdeva, 2016]. In Fig. 7.3, we show that the method of Kyng and Sachdeva [2016] only preserves very low frequency eigenvectors. Our approach is geared towards operators defined on non-uniform triangle meshes and does not have limitations of the approach of Kyng and Sachdeva [2016] which only works on Laplacians where all weights are positive.

7.3 Method

The input to our method is a n-by-n sparse, positive semi-definite matrix $\mathsf{L} \in \mathbb{R}^{n \times n}$. We can perceive L as the Hessian of an energy derived from a geometric domain with n vertices and the sparsity pattern is determined by the connectivity of a mesh or local neighbor relationship. For example, L may be the discrete cotangent Laplacian, the Hessian of the discrete Dirichlet energy. However, we do not require direct access to the geometric domain or its spatial embedding. We also take as input a non-negative diagonal weighting or mass matrix $\mathsf{M} \in \mathbb{R}^{n \times n}$ (i.e., defining an inner-product on vectors from the input domain). The main parameter of our method is the positive number $m < n$ which determines the size of our coarsened output.

Our method outputs a sparse, positive semi-definite matrix $\widetilde{\mathsf{L}} \in \mathbb{R}^{m\times m}$ that attempts to maintain the low-frequency eigenvalues and eigenvectors of the input matrix L (see Fig. 7.4).

Algorithm 6: Spectral Coarsening given L, M and m

```
1. $\mathsf{P}, \mathsf{K} \leftarrow$ *combinatorial coarsening*$(\mathsf{L}, \mathsf{M}, m)$;
2. $\widetilde{\mathsf{L}}, \widetilde{\mathsf{M}} \leftarrow$ *operator optimization*$(\mathsf{L}, \mathsf{M}, \mathsf{P}, \mathsf{K})$;
```

We propose coarsening in two steps (see Alg. 2). First we treat the input matrix L as encoding a graph and select a subset of m "root" nodes, assigning all others to clusters based on a novel graph-distance. This clustering step defines a restriction operator (P in Eq. (7.1)) and a cluster-assignment operator K that determines the sparsity pattern of our output matrix $\widetilde{\mathsf{L}}$. In the second stage, we optimize the non-zero values of $\widetilde{\mathsf{L}}$.

7.3.1 Combinatorial coarsening

Given an input operator $\mathsf{L} \in \mathbb{R}^{n\times n}$ and corresponding mass-matrix $\mathsf{M} \in \mathbb{R}^{n\times n}$, the goal of this stage is to construct two sparse binary matrices $\mathsf{K}, \mathsf{P} \in \{0,1\}^{m\times n}$ (see Fig. 7.3.1). Acting as a cluster-assignment operator, K has exactly one 1 per column, so that $\mathsf{K}_{ij} = 1$ indicates that element j on the input domain is assigned to element i on the coarsened domain. Complementarily, acting as a restriction or subset-selection operator, P has exactly one 1 per row and no more than one 1 per column, so that $\mathsf{P}_{ij} = 1$ indicates that element j on the input domain is selected as element i in the coarsened domain to *represent* its corresponding cluster. Following the terminology from the algebraic multigrid literature, we refer to this selected element as the "root node" of the cluster [Manteuffel et al., 2017]. In our figures, we visualize P by drawing large dots on the selected nodes and K by different color segments.

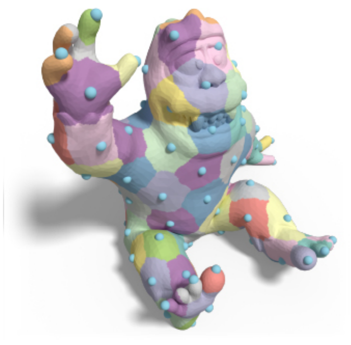


Figure 7.5: Blue dots and colored regions indicate "root nodes" and clusters selected by P and K respectively.

Consider the graph with n nodes implied by interpreting non-zeros of L as undirected edges. Our node-clustering and root-node selection should respect how quickly information at one node *diffuses* to neighboring nodes according to L and how much mass or weight is associated with each node according to M. Although a variety of algebraic distances have been proposed [Ruge and Stüben, 1987; Chen and Safro, 2011; Olson et al., 2010; Livne and Brandt, 2012], they are not directly applicable to our geometric tasks because they are not aware of different sizes M of finite elements (see Fig. 7.6).

According to this diffusion perspective, the edge-distance of the edge between nodes i

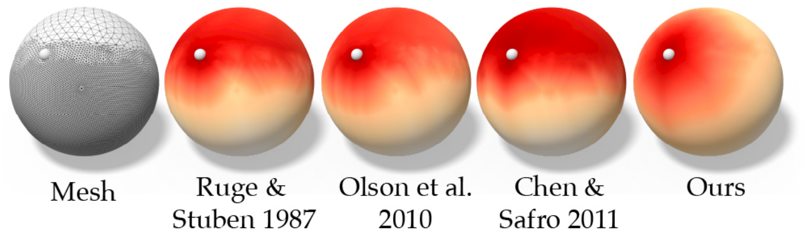


Figure 7.6: We visualize the graph shortest path distance from the source point (gray) to all the other points, where the strength of connections between adjacent points is defined using different operator-dependent strength measures. In an isotropic problem, our resulting "distance" is more robust to different element sizes and grows more uniformly in all directions (right).

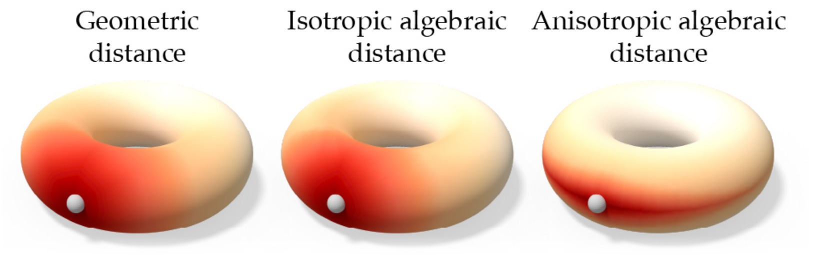


Figure 7.7: We visualize the graph shortest path distance from the source point (gray) to all the other points. Our operator-dependent distance can handle both isotropic and anisotropic problems, whereas standard geometry-based measure (e.g. edge length) is limited to isotropic problems.

and j should be inversely correlated with $-\mathsf{L}_{ij}$ and positively correlated with $(\mathsf{M}_{ii} + \mathsf{M}_{jj})$. Given the *units* of L and M in terms of powers of length p and q respectively (e.g., the discrete cotangent Laplacian for a triangle mesh has units p=0, the barycentric mass matrix has units q=2), then we adjust these correlations so that our edge-distance has units of length. Putting these relations together and avoiding negative lengths due to positive off-diagonal entries in L, we define the edge-distance between connected nodes as:

$$\mathsf{D}_{ij} = \max\left(\frac{(\mathsf{M}_{ii} + \mathsf{M}_{jj})^{(p+1)/q}}{-\mathsf{L}_{ij}}, 0\right).$$

Compared to Euclidean or geodesic distance, shortest-path distances using this edge-distance respects the anisotropy of L (see Fig. 7.8, Fig. 7.7). Compared to state-of-the-art algebraic distances, our distance accounts for irregular mass distribution, e.g., due to irregular meshing (see Fig. 7.6).

Given this (symmetric) matrix of edge-distances, we compute the k-mediods clustering [Struyf et al., 1997] of the graph nodes according to shortest path distances (computed efficiently using the modified Bellman-Ford method and Lloyd aggregation method of Bell [2008]). We initialize this iterative optimization with a random set of k root nodes. Unlike

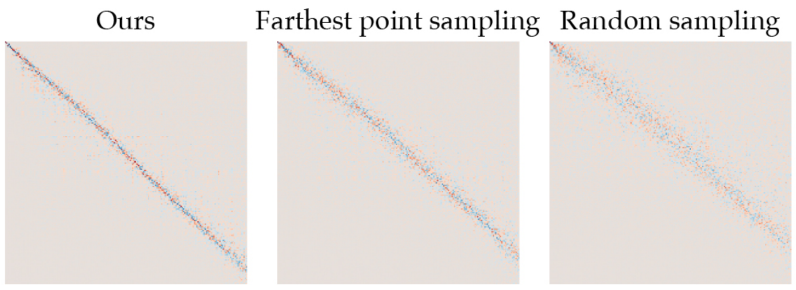


Figure 7.9: Coarsening with our operator-aware distance (left) results in better eigenfunction preservation compared to the farthest point sampling (middle) and the random sampling (right) on an anisotropic operator.

k-means where the *mean* of each cluster is not restricted to the set of input points in space, k-mediods chooses the cluster root as the mediod-node of the cluster (i.e., the node with minimal total distance to all other nodes in the cluster). All other nodes are then re-assigned to their closest root. This process is iterated until convergence. Cluster assignments and cluster roots are stored as K and P accordingly. Comparing to the farthest point sampling and the random sampling, our approach results in a better eigenfunction preservation for anisotropic operators (Fig. 7.9).

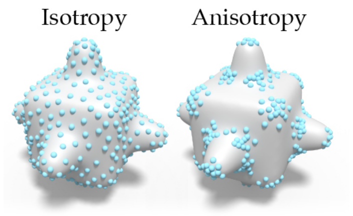


Figure 7.8: Our coarsening is aware of the anisotropy of the underlying operator, resulting in a different set of selected root nodes.

We construct a sparsity pattern for $\widetilde{\mathsf{L}}$ so that $\widetilde{\mathsf{L}}_{ij}$ may be non-zero if the cluster j is in the three-ring neighborhood of cluster i as determined by cluster-cluster adjacency. If we let $\mathsf{S}_\mathsf{L} \in \{0,1\}^{n\times n}$ be a binary matrix containing a 1 if and only if the corresponding element of L is non-zero, then we can compute the "cluster adjacency" matrix $\widetilde{\mathsf{A}} = \mathsf{K}\mathsf{S}_\mathsf{L}\mathsf{K}^\top \in \{0,1\}^{m\times m}$ so that $\widetilde{\mathsf{A}}_{ij} = 1$ if and only if the clusters i and j contain some elements u and v such that $\mathsf{L}_{uv} \neq 0$. Using this adjacency matrix, we create a sparse restriction matrix with wider connectivity $\mathsf{S}_\mathsf{G} = \mathsf{K}^\top\widetilde{\mathsf{A}} \in \{0,1\}^{n\times m}$. Finally, our predetermined sparsity pattern for $\widetilde{\mathsf{L}}$ is defined to be that of $\mathsf{S}_{\widetilde{\mathsf{L}}} = \mathsf{S}_\mathsf{G}{}^\top\mathsf{S}_\mathsf{L}\mathsf{S}_\mathsf{G} = \widetilde{\mathsf{A}}^3 \in \{0,1\}^{m\times m}$. We found that using the cluster three-ring sparsity is a reasonable trade-off between infill density and performance of the optimized operator. Assuming the cluster graph is 2-manifold with average valence 6, the three-ring sparsity implies that $\widetilde{\mathsf{L}}$ will have 37 non-zeros per row/column on average, independent to m and n. In practice, our cluster graph is nearly 2-manifold. The $\widetilde{\mathsf{L}}$ in Fig. 7.3.1, for instance, has approximately 39 non-zeros per row/column.

7.3.2 Operator optimization

Given a clustering, root node selection and the desired sparsity pattern, our second step is to compute a coarsened matrix $\widetilde{\mathsf{L}}$ that maintains the eigenvectors of the input matrix L as much as possible. Since L and $\widetilde{\mathsf{L}}$ are of different sizes, their corresponding eigenvectors are also of different lengths. To compare them in a meaningful way we will use the functional map matrix C defined in Eq. (7.1) implied by the restriction operator P (note that: prolongation from coarse to fine is generally ill-defined). This also requires a mass-matrix on the coarsened domain, which we compute by lumping cluster masses: $\widetilde{\mathsf{M}} = \mathsf{K}\mathsf{M}\mathsf{K}^\top$. The first k eigenvectors for the input operator and yet-unknown coarsened operator may be computed as solutions to the generalized eigenvalue problems $\mathsf{L}\Phi = \Lambda\mathsf{M}\Phi$ and $\widetilde{\mathsf{L}}\widetilde{\Phi} = \widetilde{\Lambda}\widetilde{\mathsf{M}}\widetilde{\Phi}$, where $\Lambda, \widetilde{\Lambda}$ are eigenvalue matrices.

Knowing that the proximity of the functional map matrix C to an identity matrix encodes eigenvector preservation, it might be tempting to try to enforce $\|\mathsf{C} - \mathsf{I}\|_F$ directly. This however is problematic because it does not handle sign flips or multiplicity (see Fig. 7.10). More importantly, recall that in our setting C is not a free variable, but rather a non-linear function (via eigen decomposition) of the unknown sparse matrix $\widetilde{\mathsf{L}}$.

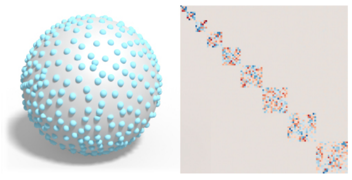


Figure 7.10: The optimized C should be block diagonal when the operator has algebraic multiplicity.

Instead, we propose to minimize the failure to realize the commutativity diagram of a functional map. Ideally, for any function on the input domain $\mathbf{f} \in \mathbb{R}^n$ applying the input operator $\mathsf{M}^{-1}\mathsf{L}$ and then the restriction matrix P is equivalent to applying P then $\widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}$, resulting in the same function $\widetilde{\mathbf{f}} \in \mathbb{R}^m$ on the coarsened domain:

$$\begin{tikzcd} \mathbf{f} \arrow[r, "\mathsf{M}^{-1}\mathsf{L}"] \arrow[d, "\mathsf{P}"'] & \bullet \arrow[d, "\mathsf{P}"] \\ \bullet \arrow[r, "\widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}"'] & \widetilde{\mathbf{f}} \end{tikzcd} \tag{7.2}$$

This leads to a straightforward energy that minimizes the difference between the two paths in the commutativity diagram for all possible functions $\mathbf{f}$:

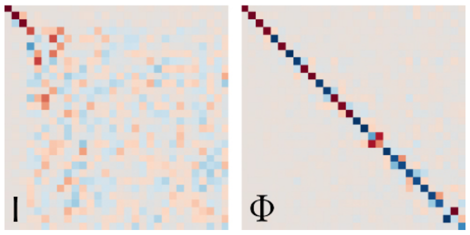
$$E(\widetilde{\mathsf{L}}) = \|\mathsf{P}\mathsf{M}^{-1}\mathsf{L}\mathsf{I} - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{P}\mathsf{I}\|^2_{\widetilde{\mathsf{M}}}, \tag{7.3}$$

where $\mathsf{I} \in \mathbb{R}^{n\times n}$ is the identity matrix (included didactically for the discussion that follows) and

$$\|\mathsf{X}\|^2_{\widetilde{\mathsf{M}}} = \mathrm{tr}(\mathsf{X}^\top\widetilde{\mathsf{M}}\mathsf{X})$$

computes the Frobenius inner-product defined by $\widetilde{\mathsf{M}}$.

By using I as the spanning matrix, we treat all functions equally in an L2 sense. Inspired by the functional maps literature, we can instead compute this energy over only lower frequency functions spanned by the first k eigenvectors $\Phi \in \mathbb{R}^{n\times k}$ of the operator L. Since high frequency functions naturally cannot live on a coarsened domain, this parameter k allows the optimization to focus on functions that matter. Consequently, preservation of low frequency eigenvectors dramatically improves (see inset).



Substituting Φ for I in Eq. (7.3), we now consider minimizing this reduced energy E_k over all possible *sparse* positive semi-definite (PSD) matrices $\widetilde{\mathsf{L}}$:

$$\underset{\widetilde{\mathsf{L}} \doteq \mathsf{S}_{\widetilde{\mathsf{L}}}}{\text{minimize}} \ \underbrace{\frac{1}{2}\|\mathsf{P}\mathsf{M}^{-1}\mathsf{L}\Phi - \widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{P}\Phi\|^2_{\widetilde{\mathsf{M}}}}_{E_k(\widetilde{\mathsf{L}})} \tag{7.4}$$

$$\text{subject to} \ \ \widetilde{\mathsf{L}} \text{ is positive semi-definite} \tag{7.5}$$

$$\text{and} \ \ \widetilde{\mathsf{L}}\mathsf{P}\Phi_0 = 0 \tag{7.6}$$

where we use $\mathsf{X} \doteq \mathsf{Y}$ to denote that X has the same sparsity pattern as Y. The final linear-equality constraint in Eq. (7.6) ensures that the eigen-vectors Φ_0 corresponding to zero eigenvalues are exactly preserved ($\mathsf{P}\mathsf{L}\Phi_0 = \mathsf{P}\mathsf{M}\Phi_0 0 = 0$). Note that while it might seem that Eq. (7.4) is only meant to preserve the eigen*vectors*, a straightforward calculation (See App. 7.6.3) shows that it promotes the preservation of eigen*values* as well.

This optimization problem is convex [Boyd and Vandenberghe, 2004], but the sparsity constraint makes it challenging to solve efficiently. Most efficient semi-definite programming (SDP) solvers (e.g., Mosek, cvxopt, Gurobi) only implement *dense* PSD constraints. The academic community has studied SDPs over sparse matrices, yet solutions are not immediately applicable (e.g., those based on chordal sparsity [Vandenberghe and Andersen, 2015; Zheng et al., 2017]) or practically efficient (e.g., [Andersen et al., 2010]). Even projecting a sparse matrix X on to the set of PSD matrices with the same sparsity pattern is a difficult sub-problem (the so-called sparse matrix nearness problem, e.g., [Sun and Vandenberghe, 2015]), so that proximal methods such as ADMM lose their attractiveness.

If we drop the PSD constraint, the result is a simple quadratic optimization with linear constraints and can be solved directly. While this produces solutions with very low objective values E_k, the eigenvector preservation is sporadic and negative eigenvalues appear (see Fig. 7.11). Conversely, attempting to replace the PSD constraint with the *stricter* but more amenable diagonal dominance linear inequality constraint (i.e., $\widetilde{\mathsf{L}}_{ii} \geq \sum_{j\neq i} \widetilde{\mathsf{L}}_{ji}$) produces a worse objective value and poor eigenvector preservation.

Instead, we propose introducing an auxiliary sparse matrix variable $\mathsf{G} \in \mathbb{R}^{n\times m}$ and restricting the coarsened operator to be created by using G as an interpolation operator:

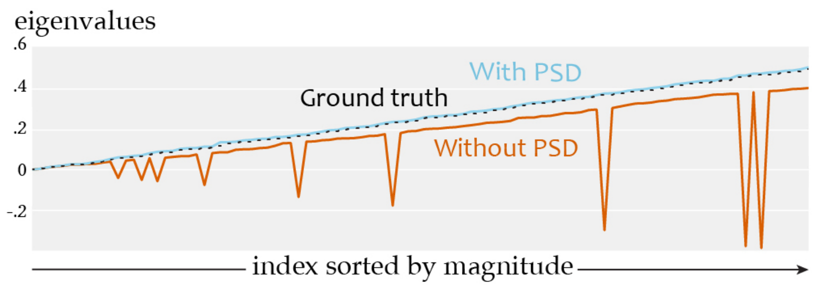


Figure 7.11: Dropping the PSD constraint leads to a simple quadratic optimization problem which can be solved directly, but it produces a non-PSD $\widetilde{\mathsf{L}}$ that contains negative eigenvalues.

$\widetilde{\mathsf{L}} := \mathsf{G}^\top \mathsf{L} \mathsf{G}$. Substituting this into Eq. (7.4), we optimize

$$\begin{aligned} \underset{\mathsf{G} \doteq \mathsf{S}_\mathsf{G}}{\text{minimize}} \quad & \underbrace{\frac{1}{2} \| \mathsf{P} \mathsf{M}^{-1} \mathsf{L} \Phi - \widetilde{\mathsf{M}}^{-1} \mathsf{G}^\top \mathsf{L} \mathsf{G} \mathsf{P} \Phi \|^2_{\widetilde{\mathsf{M}}}}_{E_k(\mathsf{G})}, \\ \text{subject to} \quad & \mathsf{G} \mathsf{P} \Phi_0 = \Phi_0 \end{aligned} \tag{7.7}$$

where the sparsity of $\widetilde{\mathsf{L}}$ is maintained by requiring sparsity of G. The null-space constraint remains linear because $\mathsf{G}\mathsf{P}\Phi_0 = \Phi_0 \Rightarrow \mathsf{G}^\top \mathsf{L} \mathsf{G} \mathsf{P} \Phi_0 = 0$ implies that $\widetilde{\mathsf{L}}$ contains the null-space of L. The converse will not necessarily be true, but is unlikely to happen because this would represent inefficient minimization of the objective. In practice, we never found that spurious null-spaces occurred. While we get to remove the PSD constraint ($\mathsf{L} \succeq 0$ implies $\mathsf{G}^\top \mathsf{L} \mathsf{G} \succeq 0$), the price we have paid is that the energy is no longer quadratic in the unknowns, but quartic.

Therefore in lieu of convex programming, we optimize this energy over the non-zeros of G using a gradient-based algorithm with a fixed step size γ. Specifically, we use NADAM [Dozat, 2016] optimizer which is a variant of gradient descent that combines momentum and Nesterov's acceleration. For completeness, we provide the *sparse* matrix-valued gradient $\partial E_k / \partial \mathsf{G}$ in App. 7.6.1. The sparse linear equality constraints are handled with the orthogonal projection in App. 7.6.2). We summarize our optimization in pseudocode Alg. 6. We stop the optimization if it stalls (i.e., does not decrease the objective after 10 iterations) and use a fixed step size $\gamma = 0.02$. This rather straightforward application of a gradient-based optimization to maintaining the commutativity diagram in Eq. (7.2) performs quite well for a variety of domains and operators.

7.4 Evaluation & Validation

Our input is a matrix L which can be derived from a variety of geometric data types. In Fig. 7.12 we show that our method can preserve the property of the Laplace operators defined on triangle meshes [Pinkall and Polthier, 1993; MacNeal, 1949; Desbrun et al., 1999], point clouds [Belkin et al., 2009], graphs, and tetrahedral meshes [Sharf et al., 2007]. We

Algorithm 7: Operator optimization using NADAM

```
1. $\mathsf{G} \leftarrow \mathsf{K}^\top$;                                   // initialization
2. while not stalled do
3.     $\partial E_k/\partial \mathsf{G} \leftarrow$ sparse gradient $(\mathsf{G})$;
4.     $\Delta\mathsf{G} \leftarrow$ NADAM$(\partial E_k/\partial \mathsf{G})$;
5.     $\mathsf{G} \leftarrow \mathsf{G} - \gamma\,\Delta\mathsf{G}$;
6.     $\mathsf{G} \leftarrow$ orthogonal projection $(\mathsf{G}, \Phi_0)$;   // see App. 7.6.2
```

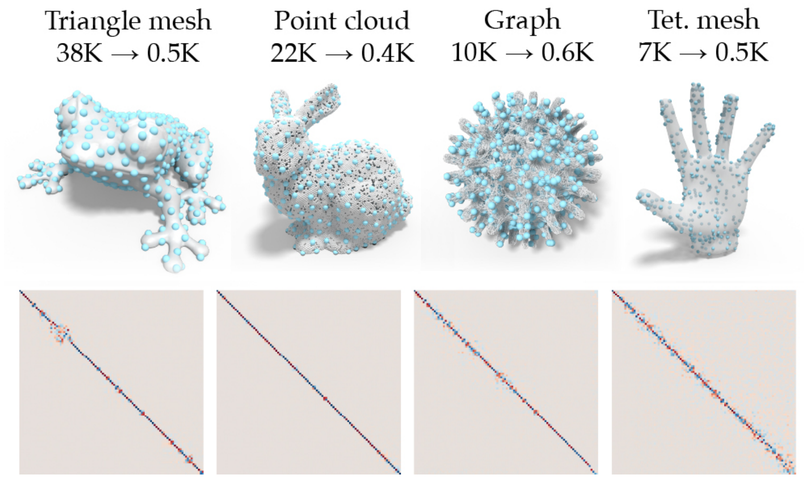


Figure 7.12: Our algebraic formulation is directly applicable to different data types, such as triangle meshes, point clouds, graphs, and tetrahedral meshes.

also evaluate our method on a variety of operators, including the offset surface Laplacian [Corman et al., 2017], the Hessian of the Ricci energy [Jin et al., 2008], anisotropic Laplacian [Andreux et al., 2014], and the intrinsic Delaunay Laplacian [Fisher et al., 2006] (see Fig. 7.13).

We further evaluate how the coarsening generalizes beyond the optimized eigenfunctions. In Fig. 7.4, we coarsen the shape using the first 100 eigenfunctions and visualize the 200×200 functional map image. This shows a strong diagonal for the upper 100×100 block and a slowly blurring off-diagonal for the bottom block, demonstrating a graceful generalization beyond the optimized eigenfunctions.

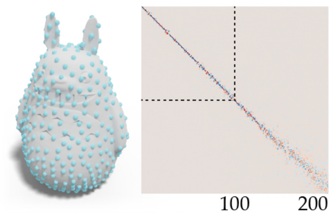


Figure 7.14: We optimize for the first 100 eigenfunctions and visualize the 200×200 functional map, demonstrating a graceful generalization beyond the optimized eigenfunctions.

Our algebraic approach takes the operator as the input, instead of the mesh, thus the output quality is robust to noise or sharp features (see Fig. 7.15). In addition, we can apply our method recursively to the output operator to construct a multilevel hierarchy (see Fig. 7.16).

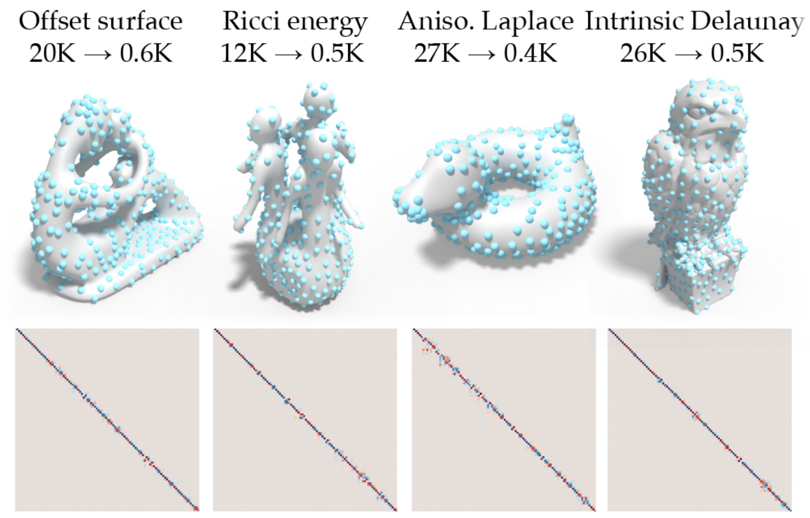


Figure 7.13: Our method preserves the eigenfunctions of the offset surface Laplacian, the Hessian of the Ricci energy, the anisotropic Laplace, and the the intrinsic Delaunay Laplacian.

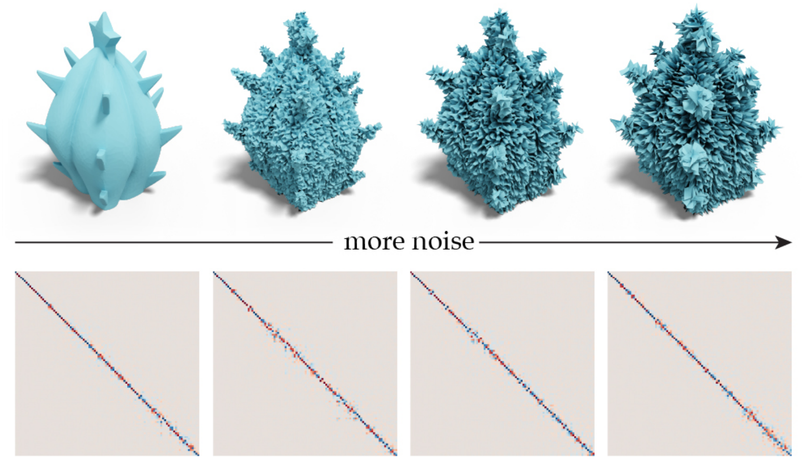


Figure 7.15: Our coarsening takes the operator as the input, thus the output quality is robust to noise and sharp geometric features.

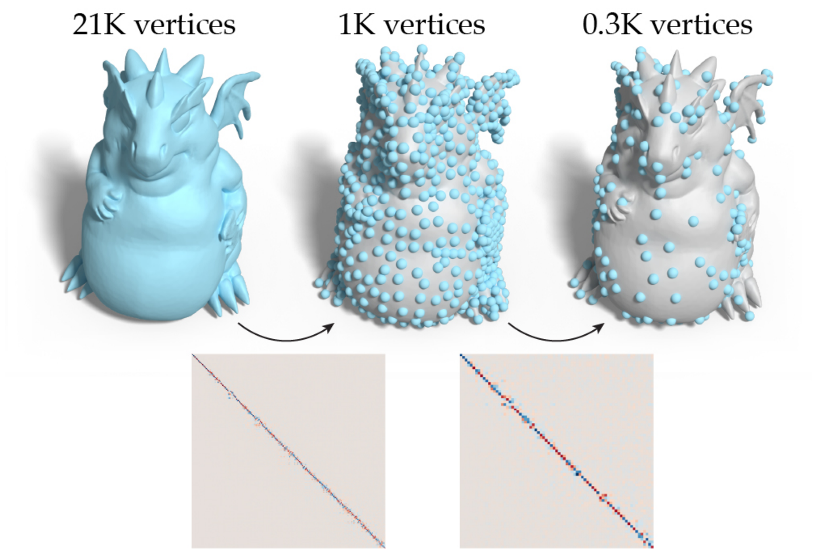


Figure 7.16: We apply our approach recursively to construct a multilevel hierarchy: from 21,000 rows through 1,000 rows to finally 300 rows.

7.4.1 Comparisons

Existing coarsening methods are usually not designed for preserving the spectral property of operators. Geometry-based mesh decimation (i.e., *QSlim* [Garland and Heckbert, 1997]) is formulated to preserve the appearance of the geometry, and results in poor performance in preserving the operator (see Fig. 7.1). As an iterative solver, algebraic multigrid, i.e., root-node method [Manteuffel et al., 2017], optimizes the convergence rate and does not preserve the spectral properties either. Recently, Nasikun et al. [2018] propose approximating the isotropic Laplacian based on constructing locally supported basis functions. However, this approach falls short in preserving the spectral properties of shapes with high-curvature thin structures and anisotropic operators (see Fig. 7.17, Fig. 7.18). In contrast, our proposed method can effectively preserve the eigenfunctions for both isotropic and anisotropic operators.

In addition, a simple derivation (App. 7.6.3) can show that minimizing the proposed energy implies eigenvalue preservation (see Fig. 7.2 and Fig. 7.19).

In Fig. 7.20, we show that our method handles anisotropy in the input operator better than existing methods. This example also demonstrates how our method gracefully degrades as anisotropy increases. Extreme anisotropy (far right column) eventually causes our method to struggle to maintain eigenvectors.

7.4.2 Implementation

In general, let k be the number of eigenvectors/eigenvalues in use, we recommend to use the number of root nodes $m > k \times 2$. In Fig. 7.21 we show that if m is too small, the degrees

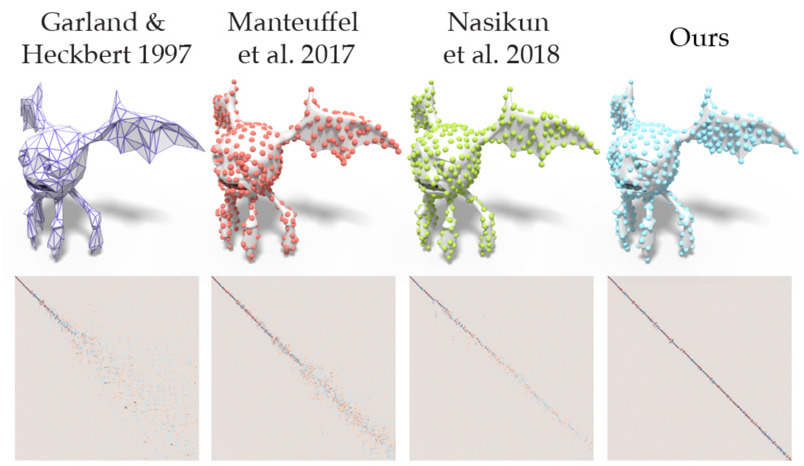


Figure 7.17: We simplify the Laplacian from $n = 30,000$ to $m = 500$. Our coarsening preserve the first 200 eigenfunctions better than the QSlim [Garland and Heckbert, 1997], the root-node algebraic multigrid [Manteuffel et al., 2017], and the fast approximation [Nasikun et al., 2018].

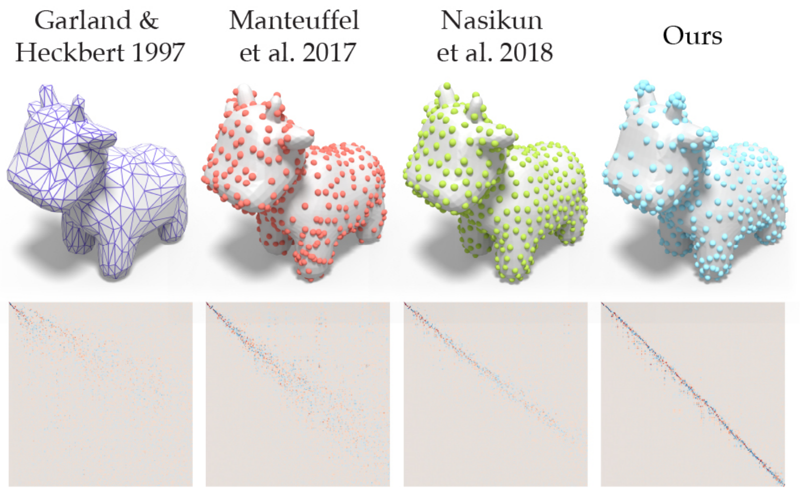


Figure 7.18: We simplify the anisotropic Laplacian [Andreux et al., 2014] (with parameter 70) from $n = 25,000$ to $m = 600$. Our approach can preserve eigenfunctions of anisotropic operators better than the existing approaches.

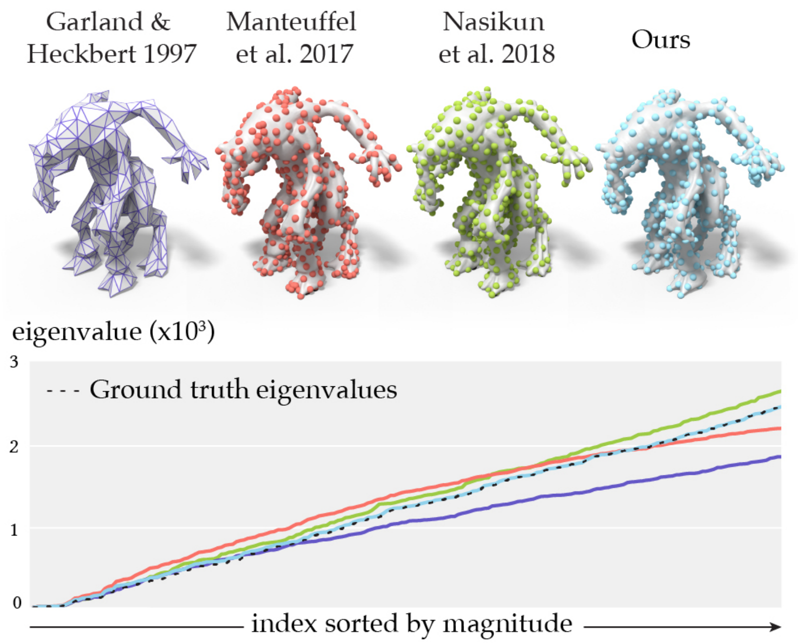


Figure 7.19: We compare the performance of preserving eigenvalues with different simplification methods. As optimizing our proposed energy implies eigenvalue preservation, we show that the eigenvalues of the simplified operator is well-aligned with the original eigenvalues.

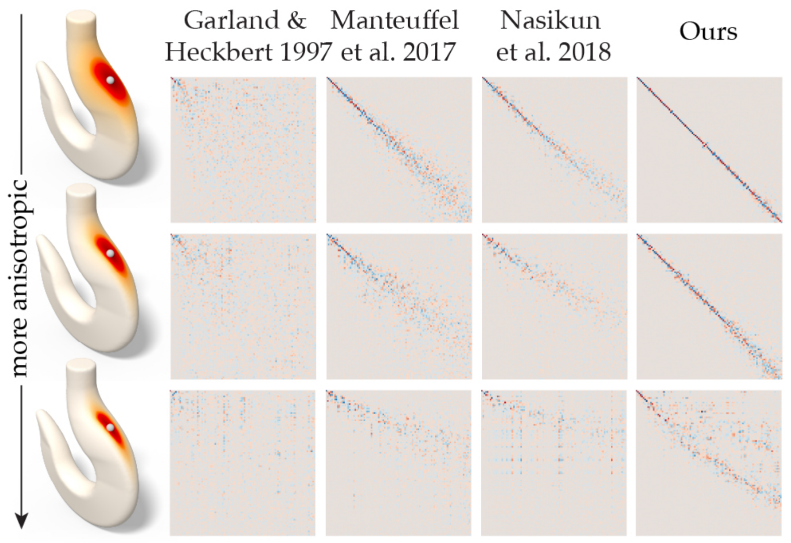


Figure 7.20: We increase the anisotropy parameter of [Andreux et al., 2014] (60, 120, 180) while simplifying an operator from 21,000 rows down to 500. Our approach handles anisotropy better than existing approaches but still struggles to preserve extreme anisotropic operators.

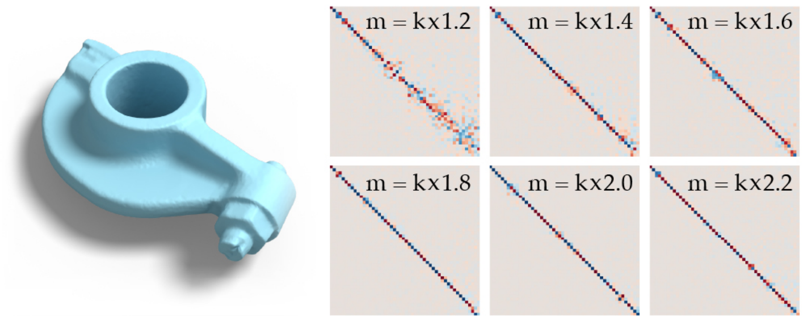


Figure 7.21: Let k be the number of eigenvectors we want to preserve, experimentally we observed that $m > k \times 2$ leads to desired results.

of freedom are insufficient to capture the eigenfunctions with higher frequencies.

Our serial C++ implementation is built on top of LIBIGL [Jacobson et al., 2018] and SPECTRA [Qiu, 2018]. We test our implementation on a Linux workstation with an Intel Xeon 3.5GHz CPU, 64GB of RAM, and an NVIDIA GeForce GTX 1080 GPU. We evaluate our runtime using the mesh from Fig. 7.4 in three different cases: (1) varying the size of input operators n, (2) varying the size of output operators m, and (3) varying the number of eigenvectors in use k. All experiments converge in 100-300 iterations. We report our runtime in Fig. 8.18. We obtain 3D shapes mainly from *Thingi10K* [Zhou and Jacobson, 2016] and clean them with the method of [Hu et al., 2018].

7.4.3 Difference-Driven Coarsening

We also validate our combinatorial coarsening by applying it to the shape difference operator [Rustamov et al., 2013] which provides an informative representation of how two shapes differ from each other. As a positive definite operator it fits naturally into our framework. Moreover, since the difference is captured via functional maps, it does not require two shapes to have the same triangulation. We therefore take a pair of shapes with a known functional map between them, compute the shape difference operator and apply our combinatorial coarsening, while trying to best preserve this computed operator. Intuitively, we expect the samples to be informed by the shape difference and thus capture the areas of distortion between the shapes (see App. 7.6.4 for more detail). As shown in Fig. 7.23, our coarsening indeed leads to samples in areas where the intrinsic distortion happens, thus validating the ability of our approach to capture and reveal the characteristics of the input operator.

We can further take the element-wise maximum from a collection of shape difference operators to obtain a data-driven coarsening informed by many shape differences (see Fig. 7.24).

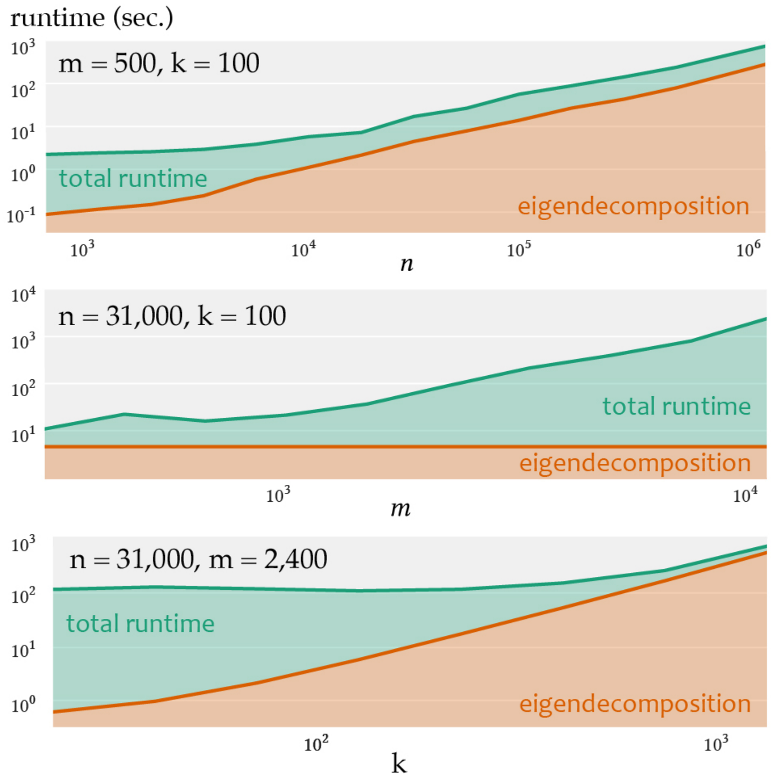


Figure 7.22: Our runtime shows that our approach is more suitable for aggressive coarsening (middle). For large input meshes and many eigenvectors in use (top, bottom), computing eigendecomposition is the bottleneck.

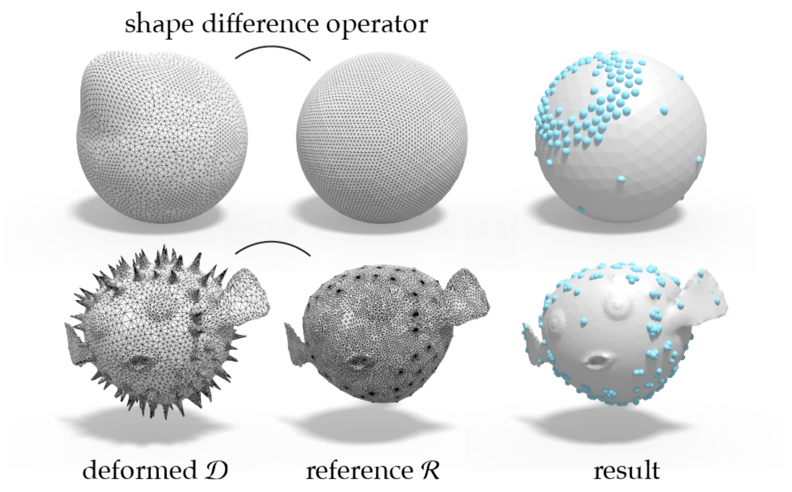


Figure 7.23: Given a reference shape $\mathcal{R}$ and its deformed version $\mathcal{D}$, we combine the shape difference operator with our coarsening to compute a set of samples that capture the areas of highest distortion between the shapes.

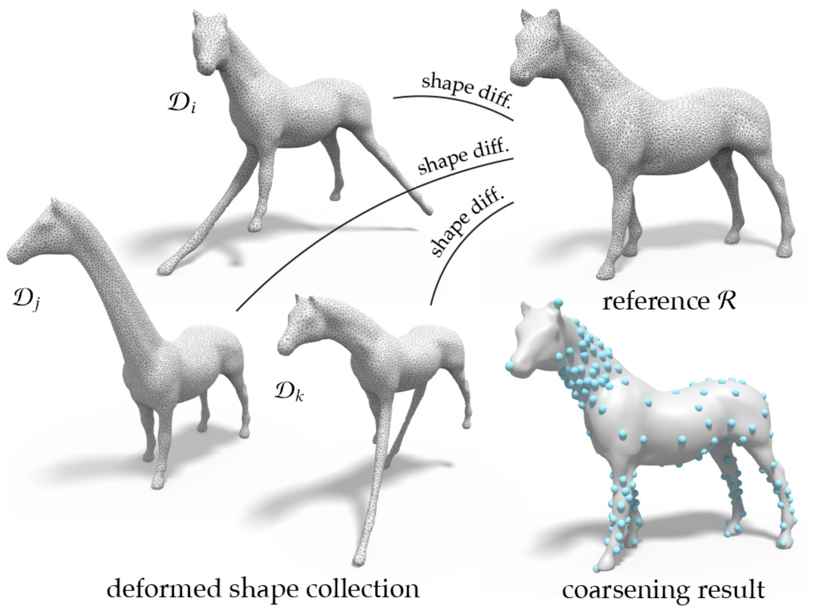


Figure 7.24: By combining the shape difference operators from the reference shape $\mathcal{R}$ to a collection of deformed shapes $\mathcal{D}$, algebraic coarsening can simplify a mesh based on the "union" of all the deformations.

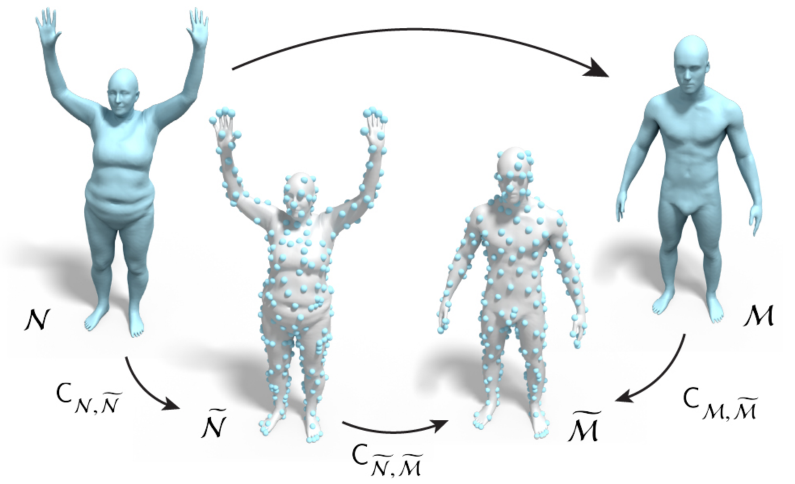

$\mathcal{N}$
$\mathcal{M}$
$\mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}$
$\widetilde{\mathcal{N}}$
$\mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$
$\widetilde{\mathcal{M}}$
$\mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}$

Figure 7.25: Efficient matching computes the map $\mathsf{C}_{\mathcal{N},\mathcal{M}}$ between the original shapes by (1) applying our proposed coarsening to the shape pair and obtain $\mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}$, $\mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}$, (2) compute shape correspondences $\mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$ in the reduced space, and (3) solve a linear system based on the commutative diagram.

7.4.4 Efficient Shape Correspondence

A key problem in shape analysis is computing correspondences between pairs of non-rigid shapes. In this application we show how our coarsening can significantly speed up existing shape matching methods while also leading to comparable or even higher accuracy. For this we use a recent iterative method based on the notion of *Product Manifold Filter* (PMF), which has shown excellent results in different shape matching applications [Vestner et al., 2017a]. This method, however, suffers from high computational complexity, since it is based on solving a linear assignment problem, $O(n^3)$, at each iteration. Moreover, it requires the pair of shapes to have the same number of vertices. As a result, in practice before applying PMF shapes are typically subsampled to a coarser resolution and the result is then propagated back to the original meshes. For example in [Litany et al., 2017], the authors used the standard QSlim [Garland and Heckbert, 1997] to simplify the meshes before matching them using PMF. Unfortunately, since standard appearance-based simplification methods can severely distort the spectral properties this can cause problems for spectral methods such as [Vestner et al., 2017a] both during matching between coarse domains and while propagating back to the dense ones. Instead our spectral-based coarsening, while not resulting in a mesh provides all the necessary information to apply a spectral technique via the eigen-pairs of the coarse operator, and moreover provides an accurate way to propagate the information back to the original shapes.

More concretely, we aim to find correspondences between the coarsened shapes $\widetilde{\mathcal{N}}, \widetilde{\mathcal{M}}$ and to propagate the result back to the original domains $\mathcal{N}, \mathcal{M}$ by following a commutative diagram (see Fig. 7.25). When all correspondences are encoded as functional maps this diagam can be written in matrix form as:

$$\mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}\mathsf{C}_{\mathcal{N},\mathcal{M}} = \mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}\mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}, \tag{7.8}$$

where $\mathsf{C}_{\mathcal{X},\mathcal{Y}}$ denotes the functional map from $\mathcal{X}$ to $\mathcal{Y}$. Using Eq. 7.8, the functional map $\mathsf{C}_{\mathcal{N},\mathcal{M}}$ can be computed by solving a simple least squares problem, via a single linear solve. Our main observation is that if the original function space is preserved during the coarsening, less error will be introduced when moving across domains.

We tested this approach by evaluating a combination of our coarsening with [Vestner et al., 2017a] and compared it to several baselines on a challenging non-rigid non-isometric dataset containing shapes from the SHREC 2007 contest [Giorgi et al., 2007], and evaluated the results using the landmarks and evaluation protocol from [Kim et al., 2011] (please see the details on both the exact parameters and the evaluation in the Appendix). Figure 7.26 shows the accuracy of several methods, both that directly operate on the dense meshes [Kim et al., 2011; Nogneng and Ovsjanikov, 2017] as well as using kernel matching [Vestner et al., 2017a] with QSlim and with our coarsening. The results in Figure 7.26 show that our approach produces maps with comparable quality or superior quality to existing meth-

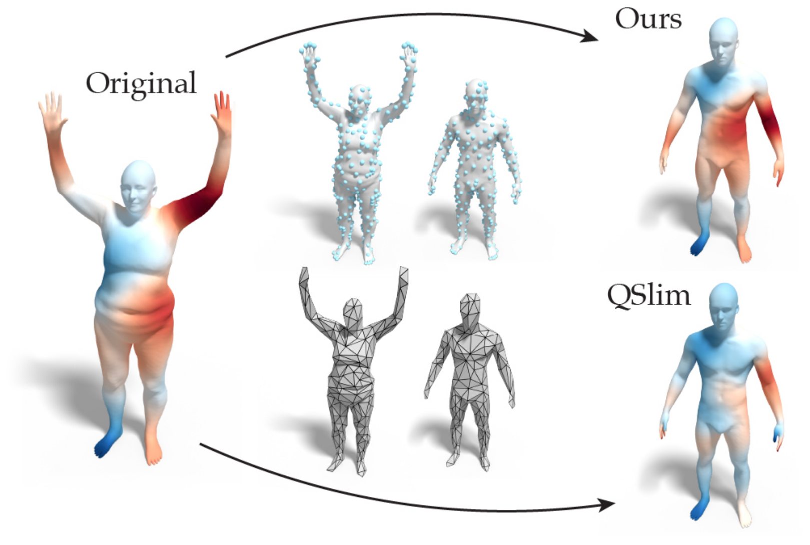


Figure 7.26: Using our coarsening (top) to infer functional maps between the original pair from the coarse pair introduces less error than using the appearance-based mesh simplification (bottom), QSlim [Garland and Heckbert, 1997].

ods on these non-isometric shape pairs, and results in significant improvement compared to coarsening the shapes with QSlim. At the same time, in Table 7.1 we report the runtime of different methods, which shows that our approach leads to a significant speed-up compared to existing techniques, and enables an efficient and accurate PMF-based matching method (see Fig. 7.27) with significantly speedup.

7.4.5 Graph Pooling

Convolutional neural networks [LeCun et al., 1998] have led to breakthroughs in image, video, and sound recognition tasks. The success of CNNs has inspired a growth of interest in generalizing CNNs to graphs and curved surfaces [Bronstein et al., 2017]. The fundamental components of a graph CNN are the *pooling* and the *convolution*. Our root node representation P, K defines a way of performing pooling on graphs. Meanwhile, our out-

Table 7.1: We report the total time, pre-computation time + runtime, for computing a 60-by-60 functional map on a shape pair with 14,000 vertices each. Our pre-computation time will be amortized by the number of pairs because we apply coarsening on each shape independently, and the number of combinations is quadratic in the number of shapes. Our runtime is orders of magnitude faster because we only need to perform shape matching in the coarsened domain (i.e., 300 vertices).

[*Nogneng 17*]+*ICP*	[*Nogneng 17*]	[*Kim 11*]	[*Vestner 17*]+*ours*
32.4 *sec*	4.6 *sec*	90.6 *sec*	10.8+**0.3** *sec*

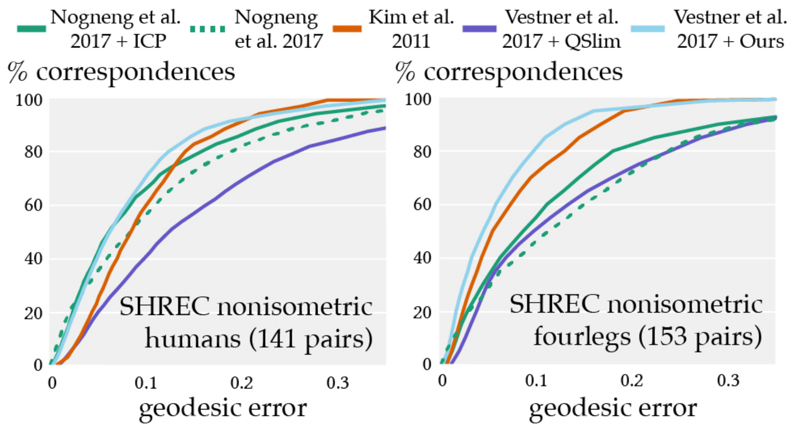


Figure 7.27: Our efficient shape correspondence with kernel matching [Vestner et al., 2017a] achieves comparable matching quality on many non-isometric shape pairs from SHREC [Giorgi et al., 2007] dataset with methods that directly operator on dense meshes [Kim et al., 2011; Nogneng and Ovsjanikov, 2017].

put $\widetilde{\mathsf{L}}$ facilitates graph convolution on the coarsened graph due to the convolution theorem [Arfken and Weber, 1999].

To evaluate the performance of graph pooling, we construct several mesh EMNIST datasets where each mesh EMNIST digit is stored as a real-value function on a triangle mesh. Each mesh EMNIST dataset is constructed by overlaying a triangle mesh with the original EMNIST letters [Cohen et al., 2017]. We compare our graph pooling with the graph pooling IN [Defferrard et al., 2016] by evaluating the classification performance. For the sake of fair comparisons, we use the same graph Laplacian, the same architecture, and the smae hyperparameters. The only difference is the graph pooling module. In addition to EMNIST, we evaluate the performance on the fashion-MNIST dataset [Xiao et al., 2017] under the same settings. In Fig. 7.28, we show that our graph pooling results in better training and testing performance. We provide implementation details in App. 7.6.6.

7.5 Limitations & Future Work

Reconstructing a valid mesh from our output coarsened operator would enable more downstream applications. Incorporating a fast eigen-approximation or removing the use of eigen decomposition would further scale the spectral coarsening. Moreover, exploring sparse SDP methods (e.g. [Sun and Vandenberghe, 2015]) could improve our operator optimization. Jointly optimizing the sparsity and the operator entries may lead to even better solutions. Further restricting the sparsity pattern of the coarsened operator while maintaining the performance would aid to the construction of a deeper multilevel representation, which could aid in developing a hierarchical graph representation for graph neural networks. A scalable coarsening with a deeper multilevel representation could promote a

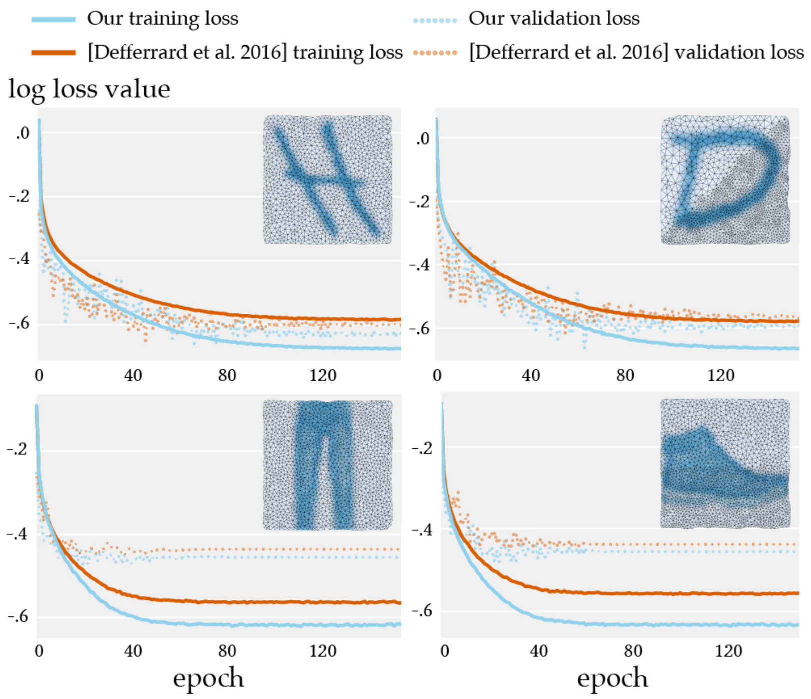


Figure 7.28: Graph pooling using our coarsening performs better than the pooling presented in [Defferrard et al., 2016] on classifying the mesh EMNIST (top row) and the mesh fashion-MNIST (bottom row) datasets.

multigrid solver for geometry processing applications.

7.6 Appendix

7.6.1 Derivative with Respect to Sparse G

To use a gradient-based solver in Alg. 6 to solve the optimization problem in Eq. (7.7) we need derivatives with respect to the non-zeros in G (the sparsity of $\mathsf{S_G}$). We start with the dense gradient:

$$\frac{\partial E_k}{\partial \mathsf{G}} = \frac{\partial}{\partial \mathsf{G}} \frac{1}{2} \|\mathsf{P}\mathsf{M}^{-1}\mathsf{L}\Phi - \widetilde{\mathsf{M}}^{-1}\mathsf{G}^\top \mathsf{L}\mathsf{G}\mathsf{P}\Phi\|^2_{\widetilde{\mathsf{M}}}.$$

We start the derivation by introducing two constant variables A, B to simplify the expression

$$\frac{\partial E_k}{\partial \mathsf{G}} = \frac{\partial}{\partial \mathsf{G}} \frac{1}{2} \|\mathsf{A} - \widetilde{\mathsf{M}}^{-1}\mathsf{G}^\top \mathsf{L}\mathsf{G}\mathsf{B}\|^2_{\widetilde{\mathsf{M}}}$$
$$\mathsf{A} = \mathsf{P}\mathsf{M}^{-1}\mathsf{L}\Phi, \ \mathsf{B} = \mathsf{P}\Phi.$$

Using the fact that $\mathsf{L}, \mathsf{M}, \mathsf{M}_c$ are symmetric matrices and the rules in matrix trace derivative, we expand the equation as follows.

$$\begin{aligned}
\frac{\partial E_k}{\partial \mathsf{G}} &= \frac{\partial}{\partial \mathsf{G}} \frac{1}{2} \|\mathsf{A} - \widetilde{\mathsf{M}}^{-1}\mathsf{G}^\top \mathsf{L} \mathsf{G} \mathsf{B}\|^2_{\widetilde{\mathsf{M}}} \\
&= \frac{\partial}{\partial \mathsf{G}} \frac{1}{2} \operatorname{tr}\left((\mathsf{A}^\top - \mathsf{B}^\top \mathsf{G}^\top \mathsf{L} \mathsf{G} \widetilde{\mathsf{M}}^{-1}) \widetilde{\mathsf{M}} (\mathsf{A} - \widetilde{\mathsf{M}}^{-1} \mathsf{G}^\top \mathsf{L} \mathsf{G} \mathsf{B})\right) \\
&= -\frac{\partial}{\partial \mathsf{G}} \Big(\operatorname{tr}\left(\mathsf{B}^\top \mathsf{G}^\top \mathsf{L} \mathsf{G} \mathsf{A}\right) + \frac{1}{2} \operatorname{tr}\left(\mathsf{B}^\top \mathsf{G}^\top \mathsf{L} \mathsf{G} \widetilde{\mathsf{M}}^{-1} \mathsf{G}^\top \mathsf{L} \mathsf{G} \mathsf{B}\right) \Big) \\
&= -\left(\mathsf{L}\mathsf{G}\mathsf{A}\mathsf{B}^\top + \mathsf{L}\mathsf{G}\mathsf{B}\mathsf{A}^\top\right) \\
&\qquad + \left(\mathsf{L}\mathsf{G}\widetilde{\mathsf{M}}^{-1}\mathsf{G}^\top \mathsf{L}\mathsf{G}\mathsf{B}\mathsf{B}^\top + \mathsf{L}\mathsf{G}\mathsf{B}\mathsf{B}^\top \mathsf{G}^\top \mathsf{L}\mathsf{G}\widetilde{\mathsf{M}}^{-1}\right) \\
&= \mathsf{L}\mathsf{G}(-\mathsf{A}\mathsf{B}^\top - \mathsf{B}\mathsf{A}^\top + \widetilde{\mathsf{M}}^{-1}\mathsf{G}^\top \mathsf{L}\mathsf{G}\mathsf{B}\mathsf{B}^\top + \mathsf{B}\mathsf{B}^\top \mathsf{G}^\top \mathsf{L}\mathsf{G}\widetilde{\mathsf{M}}^{-1})
\end{aligned}$$

Computing the $\partial E_k / \partial \mathsf{G}$ subject to the sparsity S_G can be naively achieved by first computing the dense gradient $\partial E_k / \partial \mathsf{G}$ and then project to the sparsity constraint through an element-wise product with the sparsity S_G. However, the naive computation would waste a large amount of computational resources on computing gradient values that do not satisfy the sparsity. We incorporate the sparsity and compute gradients only for the non-zero entries as

$$\begin{aligned}
&\left(\frac{\partial E_k}{\partial \mathsf{G}}\right)_{ij} = Y_{i*} Z_{*j}, \quad i, j \in \mathsf{S}_\mathsf{G} \\
&Y = \mathsf{L}\mathsf{G} \\
&Z = -\mathsf{A}\mathsf{B}^\top - \mathsf{B}\mathsf{A}^\top + \widetilde{\mathsf{M}}^{-1}\mathsf{G}^\top \mathsf{L}\mathsf{G}\mathsf{B}\mathsf{B}^\top + \mathsf{B}\mathsf{B}^\top \mathsf{G}^\top \mathsf{L}\mathsf{G}\widetilde{\mathsf{M}}^{-1},
\end{aligned}$$

where Y_{i*}, Z_{*j} denote the ith row of Y and the jth column of Z.

7.6.2 Sparse Orthogonal Projection

Let $\mathsf{g} \in \mathbb{R}^z$ be the vector of non-zeros in G, so that $\text{vec}(\mathsf{G}) = \mathsf{Z}\mathsf{g}$, where $\mathsf{Z} \in \{0,1\}^{mn \times z}$ scatter matrix. Given some G_1 that does not satisfy our constraints, we would like to find its closest projection onto the matrices that do satisfy the constraints. In other words, we aim at solving:

$$\begin{aligned}
&\underset{\mathsf{G} \doteq \mathsf{S}_\mathsf{G}}{\text{minimize}} \ \|\mathsf{G} - \mathsf{G}_1\| \\
&\text{subject to } \mathsf{G}\mathsf{P}\Phi_0 - \Phi_0 = 0.
\end{aligned}$$

Using properties of the vectorization and the Kronecker product, we can now write this in terms of vectorization:

$$\begin{aligned}&\underset{\mathsf{g}}{\text{minimize}}\ \|\mathsf{g}-\mathsf{g}_1\|\\&\text{subject to } ((\mathsf{P}\Phi_0)^\top \otimes \mathsf{id}_m)\mathsf{Z}\mathsf{g}-\text{vec}(\Phi_0)=0.\end{aligned}$$

$$\begin{aligned}\mathsf{g}&=\mathsf{g}_1-\mathsf{A}^\top(\mathsf{A}\mathsf{A}^\top)^{-1}\mathsf{b}\\\mathsf{A}&=((\mathsf{P}\Phi_0)^\top\otimes\mathsf{id}_m)\mathsf{Z}\\\mathsf{b}&=((\mathsf{P}\Phi_0)^\top\otimes\mathsf{id}_m)\mathsf{Z}\mathsf{g}_1-\text{vec}(\Phi_0).\end{aligned}$$

This can be simplified to an element-wise division when Φ_0 is a single vector.

7.6.3 Eigenvalue Preservation

Minimizing the commutativity energy Eq. (7.4) implies

$$\mathsf{P}\mathsf{M}^{-1}\mathsf{L}\Phi_i=\widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{P}\Phi_i. \tag{7.9}$$

As Φ_i comes from solving the generalized eigenvalue problem, for every Φ_i we must have: $\mathsf{L}\Phi_i=\lambda_i\mathsf{M}\Phi_i$. Therefore, Eq. (7.9) implies $\lambda_i\mathsf{P}\mathsf{M}^{-1}\Phi_i=\widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}\mathsf{P}\Phi_i$, which means that $\mathsf{P}\Phi_i$ must be an eigenvector of $\widetilde{\mathsf{M}}^{-1}\widetilde{\mathsf{L}}$ corresponding to *the same eigenvalue* λ_i.

7.6.4 Modified Shape Difference

Rustamov et al. [2013] capture geometric distortions by tracking the inner products of real-valued functions induced by transporting these functions from one shape to another one via a functional map [Ovsjanikov et al., 2012]. This formulation allows us to compare shapes with different triangulations and encode the shape difference between two shapes using a single matrix. Given a functional map C between a reference shape $\mathcal{R}$ and a deformed shape $\mathcal{D}$, the area-based shape difference operator A can be written as ([Rustamov et al., 2013] option 2)

$$\mathsf{A}_{\mathcal{R},\mathcal{D}}=\mathsf{C}^\top\mathsf{C},$$

where C is the functional map from functions on $\mathcal{R}$ to functions on $\mathcal{D}$. The operator $\mathsf{A}_{\mathcal{R},\mathcal{D}}$ encodes the difference between $\mathcal{R}$ and $\mathcal{D}$. Its eigenvectors corresponding to eigenvalues larger than one encode area-expanding regions; its eigenvectors corresponding to eigenvalues smaller than one encode area-shrinking regions.

Motivated by the goal of producing denser samples in the parts undergoing shape changes, regardless of whether area is expanding or shrinking, our modified shape difference operator $\tilde{\mathsf{A}}$ has the form

$$\tilde{\mathsf{A}}_{\mathcal{R},\mathcal{D}} = (\mathsf{id} - \mathsf{A}_{\mathcal{R},\mathcal{D}})^\top (\mathsf{id} - \mathsf{A}_{\mathcal{R},\mathcal{D}}).$$

This formulation treats area expanding and shrinking equally, the eigenvectors of eigenvalues larger than zero capture where the shapes differ.

Note that this $\tilde{\mathsf{A}}$ is a size k-by-k matrix where k is the number of basis vectors in use. We map $\tilde{\mathsf{A}}$ back to the original domain by

$$\tilde{\mathsf{A}}^{\text{orig}}_{\mathcal{R},\mathcal{D}} = \Phi_{\mathcal{R}} \tilde{\mathsf{A}}_{\mathcal{R},\mathcal{D}} \Phi_{\mathcal{R}}^\top.$$

Although the $\tilde{\mathsf{A}}^{\text{orig}}_{\mathcal{R},\mathcal{D}}$ is dense and many components do not correspond to any edge in the triangle mesh, the non-zero components corresponding to actual edges contain information induced by the operator. Therefore by extracting the inverse of the off-diagonal components of $\tilde{\mathsf{A}}^{\text{orig}}_{\mathcal{R},\mathcal{D}}$ that correspond to actual edges as the $-\mathsf{L}_{ij}$, we can obtain a coarsening result induced by shape differences.

7.6.5 Efficient Shape Correspondence

We obtain dense functional maps from coarse ones by solving

$$\mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}} \mathsf{C}_{\mathcal{N},\mathcal{M}} = \mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}} \mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}, \tag{7.10}$$

where $\mathsf{C}_{\mathcal{N},\mathcal{M}}, \mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}, \mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$ are functional maps represented in the Laplace basis. $\mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$ is the functional map of functions stored in the hat basis. To distinguish $\mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$ from the others, we use $\mathsf{T}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$ to represent the map in the hat basis. Eq. (7.10) can be re-written as

$$\Phi_{\widetilde{\mathcal{M}}} \mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}} \mathsf{C}_{\mathcal{N},\mathcal{M}} = \mathsf{T}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}} \Phi_{\widetilde{\mathcal{N}}} \mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}},$$

where Φ are eigenvectors of the Laplace-Beltrami operator. Then we can solve the dense map by, for example, the MATLAB backslash.

$$\mathsf{C}_{\mathcal{N},\mathcal{M}} = (\Phi_{\widetilde{\mathcal{M}}} \mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}) \setminus (\mathsf{T}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}} \Phi_{\widetilde{\mathcal{N}}} \mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}),$$

Due to limited computational power, we often use truncated eigenvectors and functional maps. To avoid having the truncation error destroy the map inference, we use rectangular wide functional maps for both $\mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}, \mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}$ to obtain a smaller squared $\mathsf{C}_{\mathcal{N},\mathcal{M}}$. For instance, the experiments in Fig. 7.27 use size 120-by-200 for both $\mathsf{C}_{\mathcal{M},\widetilde{\mathcal{M}}}, \mathsf{C}_{\mathcal{N},\widetilde{\mathcal{N}}}$, and we only use the top left 120-by-120 block of $\mathsf{C}_{\mathcal{N},\mathcal{M}}$.

To compute $\mathsf{C}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$ (or $\mathsf{T}_{\widetilde{\mathcal{N}},\widetilde{\mathcal{M}}}$), we normalize the shape to have surface area 2,500 for numerical reasons, coarsen the shapes down to 500 vertices, and use the kernel matching [Vestner et al., 2017a] for finding bijective correspondences. We use $\alpha = 0.01$ (the same notation as [Vestner et al., 2017a]) to weight the pointwise descriptor, specifically the wave kernel signature [Aubry et al., 2011]; we use time parameter 0.01 $\times$ surface area for the heat kernel pairwise descriptors; we use 7 landmarks for shape pairs in the SHREC dataset.

7.6.6 Graph Pooling Implementation Detail

We use the LeNet-5 network architecture, the same as the one used in [Defferrard et al., 2016], to test our graph pooling on the mesh EMNIST [Cohen et al., 2017] and mesh fashion-MNIST [Xiao et al., 2017] datasets. Specifically, the network has 32, 64 feature maps at the two convolutional layers respectively, and a fully connected layer attached after the second convolutional layer with size 512. We use dropout probability 0.5, regularization weight 5e-4, initial learning rate 0.05, learning rate decay 0.95, batch size 100, and train for 150 epochs using the SGD optimizer with momentum 0.9. The graph filters have a support of 25, and each average pooling reduces the mesh size to roughly 1/8 of the size before pooling.

Our mesh is generated by the Poisson disk sampling followed by the Delaunay triangulation and a planar flipping optimization implemented in MeshLab [Cignoni et al., 2008]. We also perform local midpoint upsampling to construct meshes with non-uniform discretizations. Then EMNIST letters are "pasted" to the triangle mesh using linear interpolation.

Chapter 8

Surface Multigrid via Intrinsic Prolongation

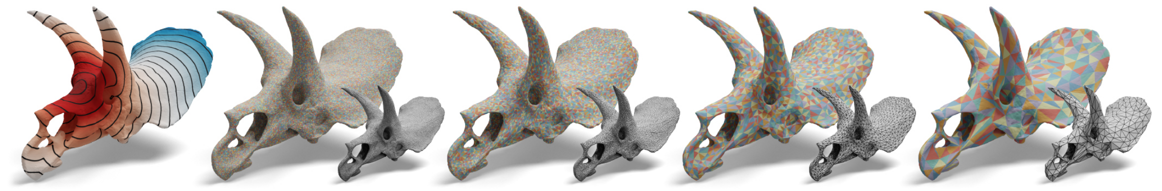


Figure 8.1: We introduce a novel geometric multigrid solver for curved surfaces. Our key ingredient is an *intrinsic* prolongation operator computed via parameterizing the high resolution shape via its coarsened counterpart, visualized using colored triangles. By recursively applying this self-parameterization, we obtain a hierarchy (from left to right) for our multigrid method (e.g., to solve heat geodesics [Crane et al., 2017], far left). ©model by Benoît Rogez under CC BY-NC.

This paper introduces a novel geometric multigrid solver for unstructured curved surfaces. Multigrid methods are highly efficient iterative methods for solving systems of linear equations. Despite the success in solving problems defined on structured domains, generalizing multigrid to unstructured curved domains remains a challenging problem. The critical missing ingredient is a prolongation operator to transfer functions across different multigrid levels. We propose a novel method for computing the prolongation for triangulated surfaces based on intrinsic geometry, enabling an efficient geometric multigrid solver for curved surfaces. Our surface multigrid solver achieves better convergence than existing multigrid methods. Compared to direct solvers, our solver is orders of magnitude faster. We evaluate our method on many geometry processing applications and a wide variety of complex shapes with and without boundaries. By simply replacing the direct solver, we upgrade existing algorithms to interactive frame rates, and shift the computational bottleneck away from solving linear systems.

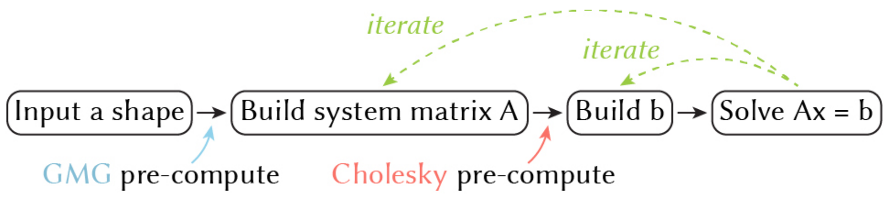


Figure 8.2: Many geometry processing algorithms that involve solving linear systems $\mathsf{A}x = \mathsf{u}^l$ often consist of four steps: (1) loading a geometry, (2) building the left-hand-side A, (3) building the right-hand-side u^l, and (4) solving the system $\mathsf{A}x = \mathsf{u}^l$. Direct solvers (e.g., Cholesky) perform pre-computation after building A, making it suitable for applications where only u^l is changing. Geometric multigrid methods perform pre-computation solely based on the geometry. Thus, even when the entire system A, u^l changes, geometric multigrid solvers can still leverage the same pre-computed hierarchy to solve the system efficiently.

8.1 Introduction

Linear solvers are the heart of many geometry processing algorithms. For positive semi-definite problems defined on surface meshes, direct solvers (e.g., Cholesky factorization) are commonplace. Unfortunately, direct solvers do not scale and often become the bottleneck for problems on high-resolution surface meshes. Especially for applications where the linear system changes at every iteration (e.g., simulation), direct solvers require an expensive re-factorization.

For problems on structured domains (e.g., 2D/3D regular grids), an excellent alternative is *geometric multigrid* methods. Geometric multigrid solvers perform pre-computation solely based on the geometry without knowing the linear system of interest (see Fig. 8.2). This enables multigrid methods to solve the system efficiently in linear time even when the system changes at each time step. Multigrid solvers already become non-trivial for unstructured grids (e.g., arbitrary triangle meshes in 2D or tetrahedral meshes in 3D), the added complexity of immersing triangle meshes in 3D has left a "black-box" multigrid solver for curved surfaces elusive until now.

In this paper, we propose a *Galerkin geometric multigrid* solver for manifold surface meshes with or without boundaries. Our key ingredient is a method for computing the *prolongation* operator based on the intrinsic geometry. Our multigrid solver achieves a better convergence rate compared to alternative multigrid methods. Replacing direct solvers with our black-box surface multigrid solver leads to orders of magnitude speed-up. We show that our method is effective in a variety of applications ranging from data smoothing to shell simulation, with linear systems of different sparsity patterns and density. Our contributions turn existing algorithms into interactive applications (e.g., Fig. 8.19) and shift the bottleneck away from solving linear systems (e.g., Fig. 8.23).

8.2 Related Works

Multigrid methods [Brandt, 1977] have earned a reputation as one of the fastest numerical solvers for solving linear systems. On structured domains (e.g., 2D/3D grid), multigrid is very well-studied both theoretically [Trottenberg et al., 2000; Hackbusch, 2013] and practically [Brandt and Livne, 2011]. In graphics, multigrid has been an attractive solution for interactive and large-scale applications on structured domains, most prominently for image processing [Kazhdan and Hoppe, 2008; Krishnan and Szeliski, 2011] and simulating fluids on large grids [McAdams et al., 2010; Aanjaneya et al., 2019; Lai et al., 2020]. Even for problems where the original representation is unstructured, an auxiliary background grid can be introduced for multigrid to perform efficient computation and transfer the solution back to the unstructured representation. For example, one can run multigrid on a background hexahedral mesh to simulate elastic deformations [Zhu et al., 2010; Dick et al., 2011] and character skinning [McAdams et al., 2011]. Chuang et al. [2009] deploy multigrid on a background voxel grid to solve Poisson problems defined on the surface mesh. To reduce the complexity of using structured representations, adaptive multigrid methods are developed for subsurface scattering [Haber et al., 2005], smoke simulation [Setaluri et al., 2014], and other graphics applications [Kazhdan and Hoppe, 2019].

Unstructured Euclidean Domains Directly deploying multigrid to unstructured "grids" in Euclidean domains (e.g., 2D triangle meshes and 3D tetrahedral meshes) has also been an important problem for decades. The main difficulties lie in how to construct the multigrid hierarchy and how to transfer signals back-and-forth across different grid levels. In graphics, unstructured multigrid for 2D triangle meshes is widely applied to cloth simulation where the design pattern is prescribed by a 2D boundary curve. In the methods proposed in [Oh et al., 2008; Jeon et al., 2013; Wang et al., 2018b], the authors generate the hierarchy in a coarse-to-fine manner by triangulating the 2D design pattern and then recursively subdividing it to get finer resolutions. Wang et al. [2018b] generate the multigrid hierarchy from fine-to-coarse by clustering vertices on the fine mesh and re-triangulating the 2D domain. When it comes to 3D tetrahedral meshes, multigrid is commonly used to simulate deformable objects. Georgii and Westermann [2006] build the hierarchy with different tetrahedralizations of the same 3D domain. Otaduy et al. [2007] repetitively compute the offset surface of the boundary mesh, decimate the offset surface, and tetrahedralize the interior to obtain the hierarchy. Sacht et al. [2015] follow a similar technique but with more elaborate and tighter fitting offsets. Adams and Demmel [1999] recursively remove the maximum independent set of vertices and tetrahedralize the interior. These unstructured multigrid methods for the Euclidean domains rely on the fact that every level in the hierarchy is a triangulation or tetrahedralization of the same space. Thus, they can easily define linear prolongation using barycentric coordinates on triangles or tetrahedra. Recently, Xian et al. [2019] propose a "mesh-free" alternative for tetrahedral meshes. They propose to use

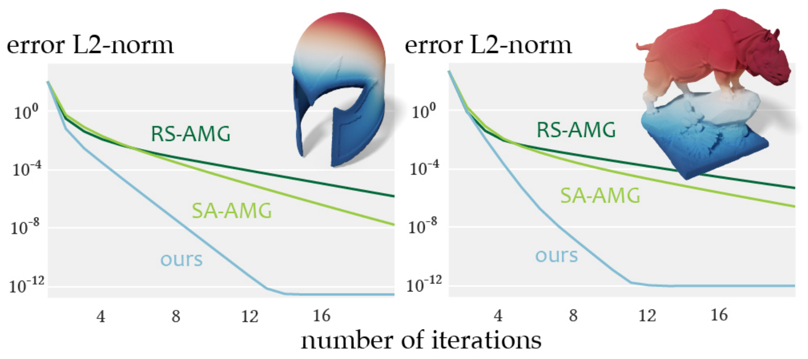


Figure 8.3: When solving a Poisson problem defined on the surface mesh (top right), we demonstrate that our multigrid based on the intrinsic prolongation (blue) converges faster than the algebraic multigrid methods (green), including the classic Ruge-Stüben (RS-AMG) [Ruge and Stüben, 1987] and the Smoothed Aggregation algebraic multigrid (SA-AMG) [Vanek et al., 1996]. Note that we use an off-the-shelf implementation from PyAMG [Olson and Schroder, 2018] with their default multigrid hyperparameters. ©models by 3DWP (right) under CC BY-SA.

farthest point sampling to get "meshes" at coarser levels, and define the prolongation using piecewise constant interpolation which only requires the closest point query.

Algebraic Multigrid A popular alternative to deal with unstructured meshes is to use *algebraic multigrid* [Brandt et al., 1985] which builds the hierarchy by treating the linear system matrix as a weighted graph and coarsening it. This approach makes no assumptions on the structure of the geometry. Thus, it is directly applicable to any unstructured domain. For this reason, algebraic methods are deployed to mesh deformation [Shi et al., 2006], cloth simulation [Tamstorf et al., 2015], and other graphics applications [Krishnan et al., 2013]. However, the cost of algebraic multigrid's generality is the need to re-build the hierarchy whenever the system matrix changes (see Fig. 8.2). Furthermore, defining the inter-grid transfer operators for algebraic methods is more challenging and leads to worse performance compared to our method (see Fig. 8.3).

Curved Surfaces When it comes to surface meshes, defining the prolongation operator becomes more challenging compared to the Euclidean case. This is because the vertices of a high-resolution surface mesh do not lie on its coarsened counterpart, thus the straightforward barycentric computation is not immediately applicable. In the special case of subdivision surfaces where the hierarchy is given, there exists efficient geometric multigrid [Green et al., 2002] and multilevel [de Goes et al., 2016] solvers that leverage the subdivision's regular refinement process to define the prolongation. For unstructured surface meshes, Ray and Lévy [2003] build the hierarchy based on the progressive meshes [Hoppe, 1996; Kobbelt et al., 1998] and define the prolongation operator using global texture coordinates. Ni et al. [2004] (and similarly Shi et al. [2009]) find a maximum independent set of

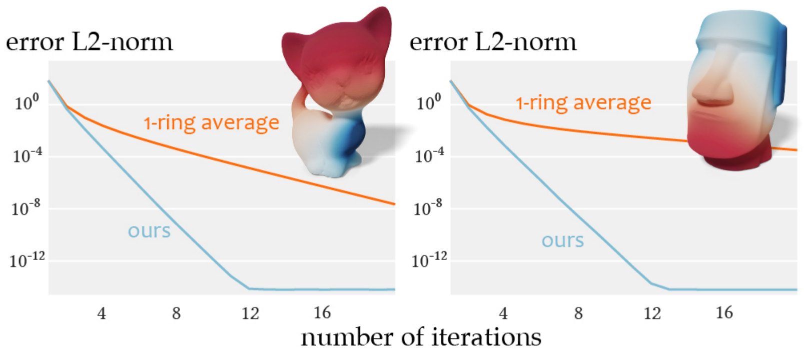


Figure 8.4: Compared to the prolongation based on the vertex 1-ring average [Aksoylu et al., 2005] (orange), our prolongation (blue) leads to a faster convergence rate when solving a Poisson problem on the surface where the top right colors are the solutions to the problem.

vertices to build a hierarchy and compute a prolongation operator based on a weighted average among one-ring neighboring vertices. Aksoylu et al. [2005] propose several methods for hierarchy construction based on removing the maximum independent set of vertices. Similarly, they also compute the prolongation by averaging among the one-ring neighbors. These approaches either need additional information (e.g., having subdivision connectivity or texture information) or use one-ring average (combinatorial information) to define the prolongation. But one-ring average often leads to a denser system because it requires on average 6 vertices to interpolate the result on a vertex, in contrast to 3 when using the barycentric interpolation. Performance wise, in Fig. 8.4 we show that our prolongation leads to a better convergence compared to the multigrid based on averaging one-ring vertices.

Our method computes the prolongation based on *intrinsic geometry*, in a similar spirit to [Sharp et al., 2019]. This enables us to define a linear prolongation simply using the barycentric coordinates, echoing the success of using barycentric coordinates in the Euclidean case. Furthermore, our approach allows one to plug-and-play different decimation strategies to construct multigrid hierarchies (see Fig. 8.13). This flexibility allows one to pick a well-suited decimation method for their tasks of interest.

Purely algebraic direct solvers (e.g., sparse Cholesky) are the *de facto* standard in geometry processing due to their reliability, scalability (as memory allows), and precision. Factorizations can be reused for changing right-hand sides (see Fig. 8.2), but trouble arises for the myriad applications where the system matrix also changes. Special classes of sparse changes can be efficiently executed: low-rank updates [Chen et al., 2008; Cheshmi et al., 2020] or partial re-factorizations [Herholz and Alexa, 2018; Herholz and Sorkine-Hornung, 2020]. However, many other scenarios trigger full numerical refactorization, such as changing parameters in a multiobjective optimization and Hessian updates for Newton's method Due to the overwhelming popularity of Cholesky factorization, we focus on it for many of our head-to-head comparisons.

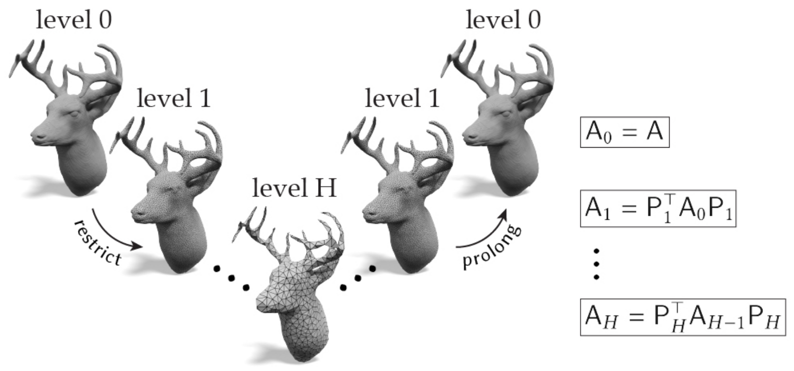


Figure 8.5: The multigrid V-cycle proceeds from the finest grid (level 0) to the coarsest grid (level H) and goes back up to the finest grid again. On each level (except for the coarsest level), we pre-relax the solution, *restrict* it to the coarser grid, compute the correction, *prolong* the correction back to the finer level, post-relax the correction, and then add the correction to the current solution. Our approach belongs to the family of *Galerkin multigrid* methods where we define the system matrix at a coarser level as $A_h = P_h^\top A_{h-1} P_h$. ©model by Takeshi Murata under CC BY-SA.

8.3 Multigrid Overview

Multigrid is a type of iterative solver that is scalable to solve large linear systems $Ax = u^l$. In this paper, we propose a novel geometric multigrid method for solving linear systems defined on curved surfaces, represented as irregular triangle meshes. We refer readers to excellent resources on multigrid [Trottenberg et al., 2000; Brandt and Livne, 2011], here we give only the essential components needed to understand our method. Note that we use "grid" or "mesh" interchangeably to denote the underlying geometry.

Multigrid methods solve a linear system in a hierarchical manner by employing two complementary processes: *relaxation* and *coarse-grid correction*. Relaxation involves applying classic iterative methods to correct the high-frequency error between the current solution and the exact solution of the system. Coarse-grid correction involves transferring the low-frequency error to a coarser mesh through *restriction*, solving a coarse-grid system of equations, then transferring the correction back to the finer mesh via *prolongation* (a.k.a. *interpolation*). This process of going from the fine grid to the coarse grid and then back to the fine grid is called the *V-cycle* (see Fig. 8.5). How to build the multigrid hierarchy and how to transfer information back and forth between grid levels are keys to determine the efficiency of a multigrid method.

Our method belongs to *geometric multigrid* based on the *Galerkin coarse grid approximation*. Geometric multigrid is a class of multigrid methods that builds the hierarchy purely based on the geometry, requiring no knowledge about the linear system. Galerkin coarse grid approximation builds the system matrix A_c on the coarsened mesh from the system

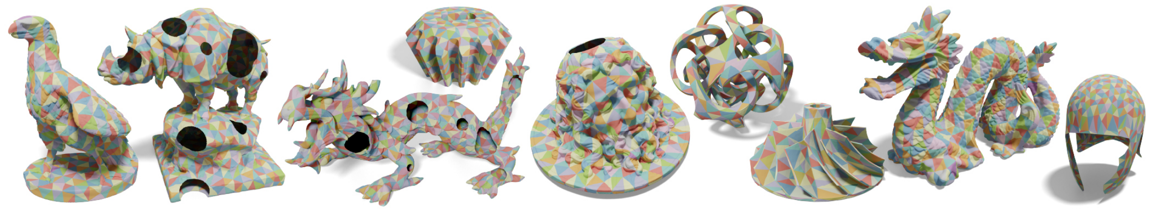


Figure 8.6: We visualize the bijective map computed using our method by coloring the high-resolution shape using the coarsened triangulation (as different colors). Our method is applicable to man-made objects, organic shapes, high-genus shapes, and meshes with boundaries. ©models by Oliver Laric (left 1, 5, 8) under CC BY-NC-SA and Landru (left 7) under CC BY.

matrix A on the original mesh as

$$\mathsf{A}_c = \mathsf{RAP}, \tag{8.1}$$

where R is the restriction operator to transfer signals from the fine mesh to the coarsened mesh and P is the prolongation operator to transfer the dual of signals (dot product between the signals with the fine basis functions) from coarse to fine. When A is symmetric, many methods often define $\mathsf{R} = \mathsf{P}^\top$. Thus, defining the prolongation operator P is extremely critical for Galerkin multigrid because it determines both the quality of the coarsened linear system $\mathsf{P}^\top\mathsf{AP}$ and the quality of the inter-grid transfer (restriction $\mathsf{P}^\top$ and prolongation P). In Alg. 9, we provide pseudo code of the Galerkin multigrid V-cycle where P plays a crucial role in the entire algorithm. An ideal prolongation must accurately interpolate smooth functions (low distortion) to ensure fast convergence. The prolongation also needs to be sparse to enhance the solver efficiency at coarser levels.

Defining a prolongation that satisfies these properties is well-studied on structured domains, but extending to unstructured curved surfaces remain a challenging problem. In this paper, we use successive self-parameterization to compute a prolongation operator P for curved surfaces based on the intrinsic geometry. Our novel joint flattening further reduces the distortion caused by the parameterization and our extension to meshes with boundaries broadens the applicability of our method to many complex geometries. When deploying our prolongation to the Galerkin multigrid framework, our method achieves better convergence than alternative multigrid methods for curved surfaces.

8.4 Intrinsic Prolongation

The central ingredient of Galerkin multigrid is the prolongation operator to interpolate signals from a coarsened mesh to its fine version. We compute the prolongation by maintaining an intrinsic parametrization, as opposed to extrinsic prolongation based on 3D spatial coordinates (cf. [Manson and Schaefer, 2011; Liu and Jacobson, 2019b]). Specifically, we

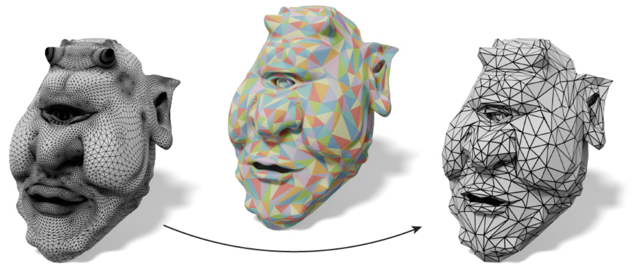


Figure 8.7: Given a high-resolution shape (left) and its coarsened counterpart (right), we compute a bijective map between the two so that for any given point on the fine mesh we can compute its corresponding barycentric coordinates on the coarsened mesh, and vice versa. We visualize the map by coloring the coarse triangulation on top of the high-resolution model.

parameterize the high-resolution mesh using the coarsened mesh to obtain a bijective map between the two (see Fig. 8.7). Given a point on the high-resolution mesh, we can obtain its corresponding barycentric coordinates on the low-resolution mesh, and vice versa. We can then assemble a linear prolongation operator based on the barycentric information.

We compute the bijective map using *successive self-parameterization*. The key idea is to successively build a bijective map for each decimation step and assemble the full map via composition. Our method for computing the successive parameterization is based on the framework of [Liu et al., 2020], which can be perceived as a combination of Lee et al. [1998] and Cohen et al. [2003]. The key differences of our method compared to [Liu et al., 2020] are a novel *joint flattening* method (see Sec. 8.4.2) to further reduce the distortion and a generalization to meshes with boundaries (see Sec. 8.4.3). For the sake of reproducibility, we reiterate the main ideas of successive self-parameterization here.

8.4.1 Successive Self-Parameterization

Let $\mathcal{M}^0$ be the input fine mesh with/without boundary. The mesh $\mathcal{M}^0$ is simplified into a series of meshes $\mathcal{M}^l$ with $0 \leq l \leq L$ through successive edge collapses. For each pair of meshes $\mathcal{M}^l, \mathcal{M}^{l+1}$, we use $f^l_{l+1} : \mathcal{M}^l \to \mathcal{M}^{l+1}$ to denote the bijective map between them. The main idea is to compute each f^l_{l+1} on-the-fly during the decimation process and composite all the maps between subsequent levels to obtain the final map $f^0_L : \mathcal{M}^0 \to \mathcal{M}^L$ as

$$f^0_L = f^L_{L+1} \circ \cdots \circ f^0_1. \tag{8.2}$$

Thus, the question boils down to the computation of the individual maps f^l_{l+1} before and after a single edge collapse.

For each edge collapse, the triangulation mostly remains the same except for the neighborhood of the collapsed edge. Thus, computing f^l_{l+1} only requires to figure out the map-

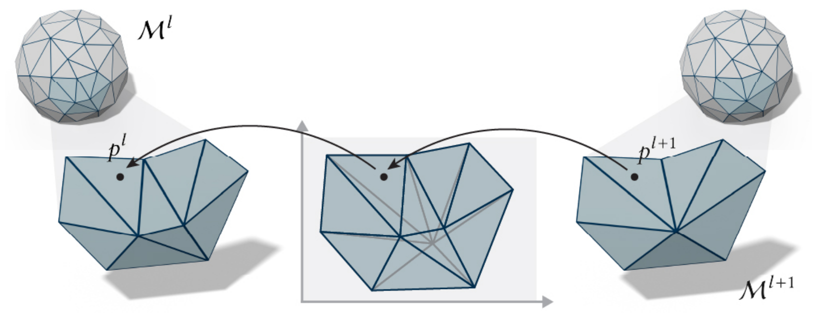


Figure 8.8: Since both the edge 1-ring before the collapse (left) and the vertex 1-ring after the collapse (right) are mapped to the same 2D domain with a consistent boundary curve (middle), we can easily use the shared UV space to map a point back and forth between $\mathcal{M}^l$ and $\mathcal{M}^{l+1}$.

ping within the edge 1-ring neighborhood. Let $\mathcal{N}^l(k)$ be the neighboring vertices of a vertex k (including vertex k itself) at level l and let $\mathcal{N}^l(i,j) = \mathcal{N}^l(i) \cup \mathcal{N}^l(j)$ denote the neighboring vertices of an edge i, j. The key observation is that the boundary vertices of $\mathcal{N}^l(i,j)$ before the collapse are the same as the boundary vertices of $\mathcal{N}^{l+1}(k)$ after the collapse, where k is the newly inserted vertex after collapsing edge i, j. Hence, we compute a shared UV-parameterization for the patches enclosed by $\mathcal{N}^l(i,j)$ and $\mathcal{N}^{l+1}(k)$ with the same boundary curve. Then, for any given point $p^l \in \mathcal{M}^l$ (represented in barycentric coordinates), we can utilize the shared UV parameterization to map p^l to its corresponding barycentric coordinates $p^{l+1} \in \mathcal{M}^{l+1}$ and vice-versa, as shown in Fig. 8.8.

8.4.2 Joint Flattening

The base method proposed by Liu et al. [2020] ensures boundary consistency by first flattening the edge 1-ring $\mathcal{N}^l(i,j)$ and setting the boundary vertices as hard constraints when flattening the vertex 1-ring $\mathcal{N}^{l+1}(k)$ after the collapse. Although this method can ensure boundary consistency, it always favors minimizing the distortion of $\mathcal{N}^l(i,j)$ and creates larger distortion when flattening $\mathcal{N}^{l+1}(k)$.

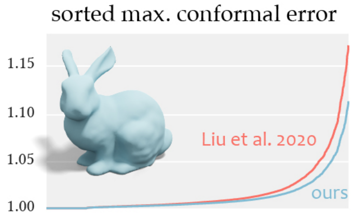


Figure 8.9: For each edge 1-ring on this bunny mesh, we collapse the edge and flatten the patch using the method of [Liu et al., 2020] and our joint flattening method. We visualize the sorted quasiconformal distortion among all the 1-rings and demonstrate that our method (blue) leads to less distortion.

We instead compute the shared UV-parameterization by *jointly* minimizing a distortion energy E defined on the edge 1-ring $\mathcal{N}^l(i,j)$ before the collapse and the vertex 1-ring $\mathcal{N}^{l+1}(k)$ after the collapse while ensuring boundary consistency. In Fig. 8.4.2, we demonstrate that our joint flattening results in a parameterization with less distortion compared to the method by Liu et al. [2020].

For notational convenience, we use V^l, F^l to denote the vertices and faces of the local patch within $\mathcal{N}^l(i,j)$ before the col-

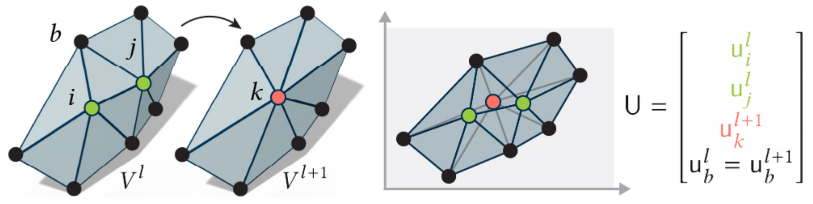

b
j
i
k
V^l
V^{l+1}
$\mathsf{U} = \begin{bmatrix} \mathsf{u}_i^l \\ \mathsf{u}_j^l \\ \mathsf{u}_k^{l+1} \\ \mathsf{u}_b^l = \mathsf{u}_b^{l+1} \end{bmatrix}$

Figure 8.10: Ensuring bijectivity between V^l and V^{l+1} requires their boundary vertices (black b) to have the same UV positions (right). We handle this constraint by introducing a joint variable U which contains shared degrees of freedom on the boundary.

lapse, and V^{l+1}, F^{l+1} to denote the vertices and faces of the local patch within $\mathcal{N}^{l+1}(k)$ after the collapse. We then write the joint energy optimization problem as

$$\underset{\mathsf{U}^l,\mathsf{U}^{l+1}}{\text{minimize}} \; E(V^l, F^l, \mathsf{U}^l) + E(V^{l+1}, F^{l+1}, \mathsf{U}^{l+1}) \tag{8.3}$$

$$\text{subject to } \mathsf{u}_b^l = \mathsf{u}_b^{l+1} \tag{8.4}$$

where we use $\mathsf{U}^l \in \mathbb{R}^{|V^l|\times 2}$ to represent the UV locations of V^l at level l and each $\mathsf{u}_i^l \in \mathbb{R}^2$ denotes a UV-vertex position. We also use $\mathsf{u}_b^l, \mathsf{u}_b^{l+1}$ to represent the boundary vertices of $\mathcal{N}^l(i,j)$ and $\mathcal{N}^{l+1}(k)$, respectively.

In order to handle the constraints, we introduce a joint variable $\mathsf{U} = \mathsf{U}^l \cup \mathsf{U}^{l+1}$ (see Fig. 8.10) to incorporate the equalities into the degrees of freedom and turn Eq. (8.3) into an unconstrained problem

$$\min_{\mathsf{U}} \; E(V^l, F^l, \mathsf{U}) + E(V^{l+1}, F^{l+1}, \mathsf{U}) \tag{8.5}$$

Introducing the joint variable U allows us to minimize the distortion energy for the patch before and after the collapse simultaneously.

8.4.3 Boundary Edges

The creation of the joint variable in Fig. 8.10 is only applicable when the collapsed edge lies fully in the interior of the triangle mesh. For boundary edges, different treatment is required to ensure the shared parameterization has a consistent boundary curve in order to preserve the bijectivity (cf. [Liu et al., 2017b]).

In the case where one of the two incident vertices lies on the boundary, we create a joint variable which snaps the UV position of the other (interior) vertex to the boundary (see Fig. 8.11). Note that we only perform this snapping operation in the parameterization domain, their corresponding vertices $\mathsf{v}_j, \mathsf{v}_k$ in $\mathbb{R}^3$ are still placed at the locations which minimize the decimation error metric (e.g., appearance preservation).

In the case where both edge vertices are on the boundary, we determine the joint vari-

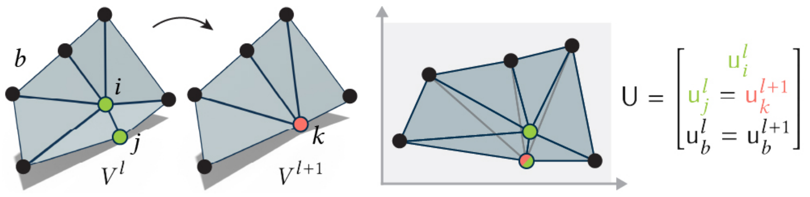

b
i
j
V^l
k
V^{l+1}
$U = \begin{bmatrix} u_i^l \\ u_j^l = u_k^{l+1} \\ u_b^l = u_b^{l+1} \end{bmatrix}$

Figure 8.11: When one of the edge vertices is on the boundary (vertex j in this case), we have to also constrain the vertex j and k to have the same UV location to ensure bijectivity, as shown in the joint variable U on the right.

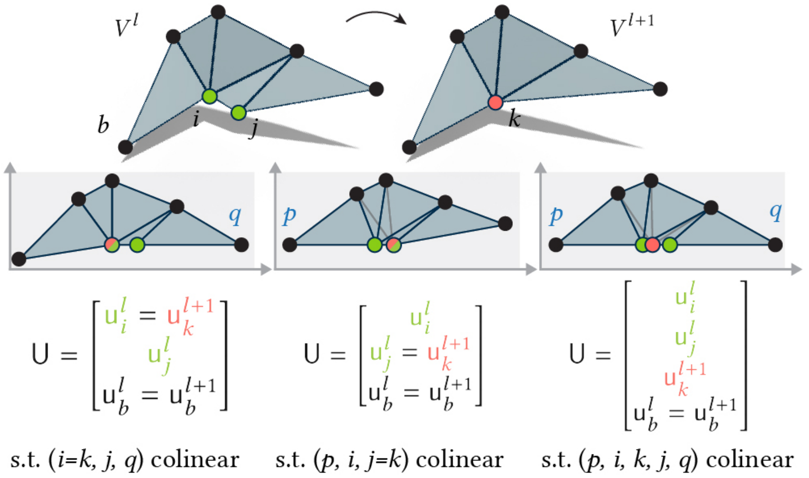

V^l
b
i
j
V^{l+1}
k
q
p
p
q
$U = \begin{bmatrix} u_i^l = u_k^{l+1} \\ u_j^l \\ u_b^l = u_b^{l+1} \end{bmatrix}$
$U = \begin{bmatrix} u_i^l \\ u_j^l = u_k^{l+1} \\ u_b^l = u_b^{l+1} \end{bmatrix}$
$U = \begin{bmatrix} u_i^l \\ u_j^l \\ u_k^{l+1} \\ u_b^l = u_b^{l+1} \end{bmatrix}$
s.t. (i=k, j, q) colinear
s.t. (p, i, j=k) colinear
s.t. (p, i, k, j, q) colinear

Figure 8.12: When the edge i, j is a boundary edge, we consider three cases: $u_i^l = u_k^{l+1}$ (left), $u_j^l = u_k^{l+1}$ (middle), and vertex i, j, k are colinear in the UV space (right). To ensure the boundary curves remain consistent, these cases result in three different sets of colinearity constraints (see the bottom row), where we use q to represent the *next* boundary vertex of the edge i, j and we use p to represent the *previous* boundary vertex of the edge i, j.

able U by choosing best of three possible choices. Suppose the boundary edge i, j is collapsed to a boundary vertex k, we consider the cases where (1) vertex k lies on vertex i, (2) vertex k lies on vertex j, and (3) vertex k lies on the line defined by i, j. Even though case (1), (2) seem unnecessary when we have case (3), these cases end up with different sets of constraints in the joint flattening optimization. Thus, we consider all three cases and take the one with the minimum energy value. In Fig. 8.12, we show how we group variables for the three cases and their corresponding colinearity constraints to maintain the same boundary curve. We impose the colinearity via adding Dirichlet constraints $(\mathsf{u}_i)_y = 0$ for all the vertices i that are colinear.

8.4.4 Decimation Strategies & Distortion Energies

Our joint flattening makes no assumption on the edge collapse algorithm in use. For instance, one could use the *quadric error edge collapse* (qslim) [Garland and Heckbert, 1997] to preserve the appearance, *mid-point edge collapse* to encourage the coarse triangulation to be more uniformly distributed, and the *vertex removal* (via half-edge collapses [Kobbelt et al., 1998]) to ensure that the vertices on the coarsened mesh are a subset of the fine vertices.

The distortion energy E in Eq. (8.5) provides another design parameter. In Fig. 8.13, we demonstrate the flexibility of our joint flattening by minimizing the *as-rigid-as-possible* (ARAP) [Liu et al., 2008] and the *least square conformal map* (LSCM) [Lévy et al., 2002] energies.

Depending on the intended application, different combinations of the decimation strategy and the parameterization algorithm may lead to different performance. For instance, in Fig. 8.14 we compare the convergence behavior of our Galerkin multigrid solvers constructed using these combinations. In our experiments, using the uniform decimation with LSCM leads to the best performance among these options. Other options for minimizing the distortion [Khodakovsky et al., 2003], computing the map [Guskov et al., 2000, 2002; Friedel et al., 2004], and decimation strategies (e.g., [Trettner and Kobbelt, 2020]) seem attractive to combine with our joint flattening. It is however more challenging and thus left as a future work.

8.4.5 Prolongation Operator

The above discussion computes a bijective map between a pair of meshes that undergoes a single edge collapse. We can easily extend the method to compute a map between two consecutive multigrid levels via composition (see Eq. (8.2)). Given this information, we can now compute a prolongation operator for surface multigrid.

We choose linear interpolation as our prolongation operator because it is sufficient for the convergence of the second-order PDEs typically employed in computer graphics [Hemker, 1990]. Although some of our experiments consider higher order PDEs, many of them are

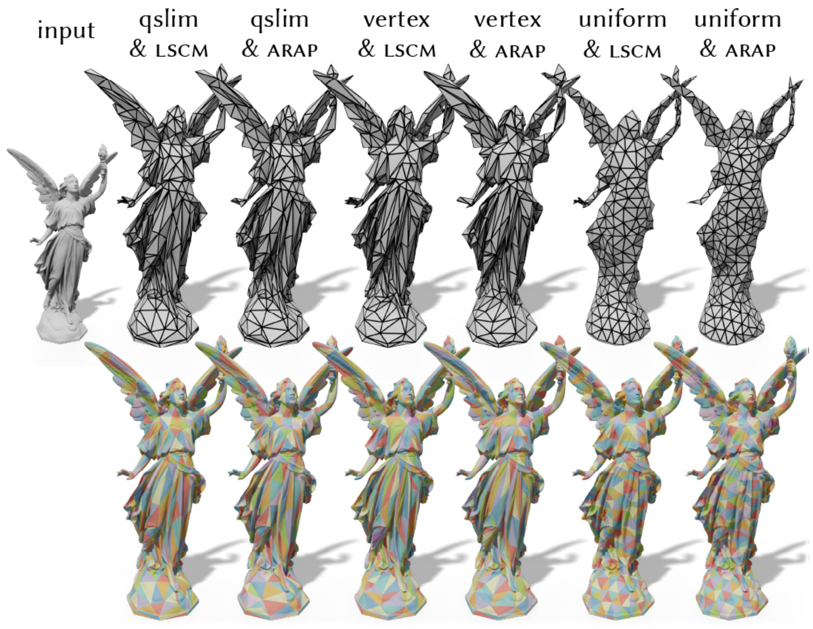


Figure 8.13: Our method allows to plug-and-play different decimation strategies and parameterization algorithms. In these examples, we decimate the model using qslim [Garland and Heckbert, 1997], vertex removal via half-edge collapse, and uniform mid-point edge collapse. We use ARAP [Liu et al., 2008] and LSCM [Lévy et al., 2002] as the parameterization algorithms. The influence of these combinations to the solver convergence is shown in Fig. 8.14.

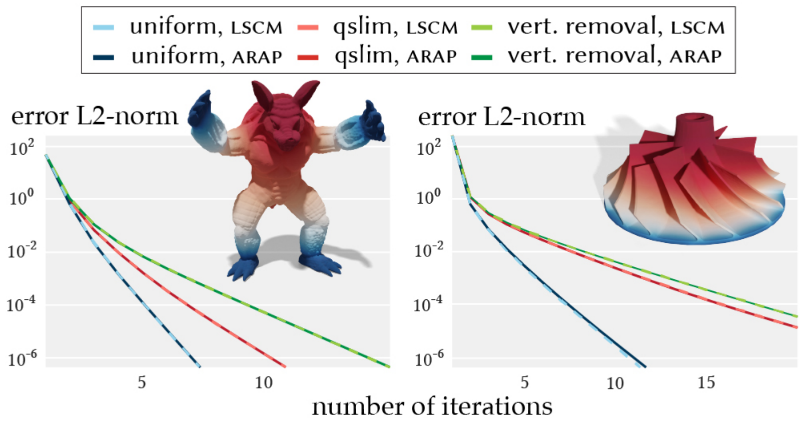


Figure 8.14: Different combinations of the decimation and the parameterization methods lead to different performance in down-stream applications. For example, in the context of multigrid solvers on a Poisson problem, the uniform edge decimation with LSCM leads to a better convergence rate.

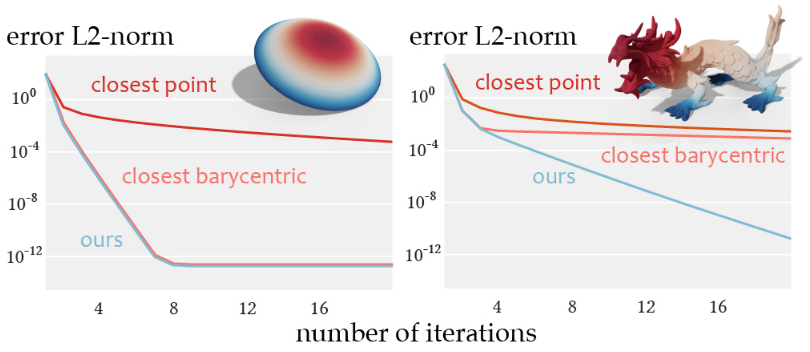


Figure 8.15: We compare our intrinsic prolongation with naive extrinsic prolongations based on the closest-point projection. When evaluating on a simple shape (left), most methods can converge; when evaluating on a complex shape (right), only our intrinsic prolongation converges. ©model by Oliver Laric (right) under CC BY-NC-SA.

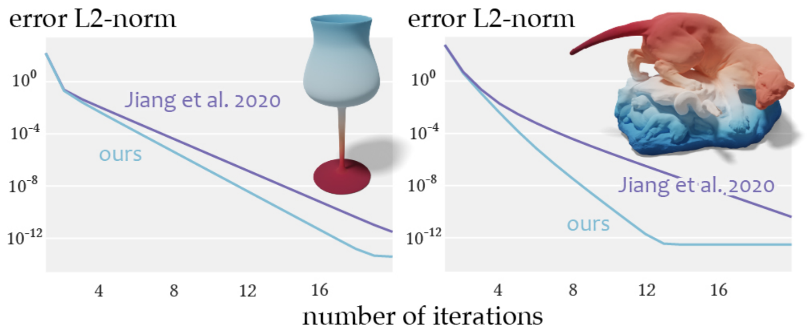


Figure 8.16: Compared to an extrinsic prolongation proposed by Jiang et al. [2020], our prolongation leads to faster convergence.

reduced to low-order systems in practice via mixed finite elements [Jacobson et al., 2010]. Empirically, we find that linear prolongation still converges in most cases.

Our linear prolongation P is a tall matrix whose size is the number of fine vertices by the number of coarse vertices. Each row of P contains 3 non-zeros corresponding to the barycentric coordinates of the fine vertex with respect to the vertices of the coarse triangle containing it. We evaluate the quality of our prolongation on solving Poisson problems on a variety meshes. We demonstrate that our intrinsic prolongation leads to faster convergence compared to the naive closest point projection (Fig. 8.15), an extrinsic bijective projection by Jiang et al. [2020] (Fig. 8.16), vertex 1-ring average [Aksoylu et al., 2005] (Fig. 8.4), and algebraic multigrid prolongations (Fig. 8.3).

8.5 Multigrid Implementation

Switching from direct solvers to our multigrid method is very simple. In Alg. 8 we summarize how one can implement a Galerkin geometric multigrid method to solve a linear system. The key difference is that the pre-computation happens right after loading the in-

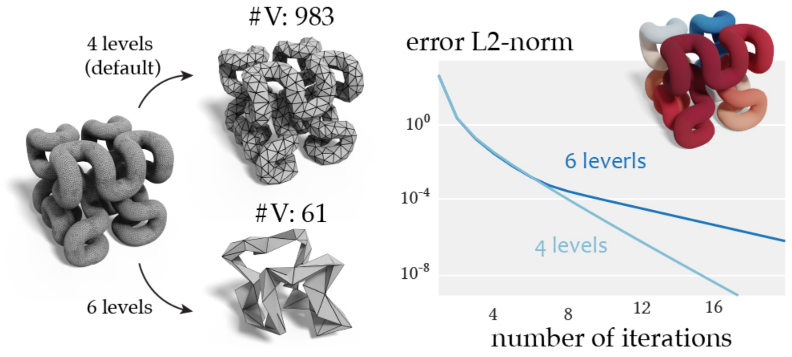


Figure 8.17: We compare the multigrid convergence between our default parameters (resulting in four multigrid levels) and an extreme coarsening (6 levels). The extreme coarsening hurts the performance (right) because the coarsest mesh fails to represent the target solution.

Algorithm 8: Galerkin Surface Multigrid Solver

```
1. $\mathsf{V}, \mathsf{F} \leftarrow$ load triangle mesh
2. $\mathsf{P}_1, \cdots, \mathsf{P}_H \leftarrow$ precompute multigrid hierarchy $(\mathsf{V}, \mathsf{F})$
3. $\mathsf{A} \leftarrow$ build left-hand-side
4. $\mathsf{u}^l \leftarrow$ build right-hand-side
5. initialize a solution $\mathsf{x}$
6. while error is larger than $\epsilon$ do
7.     $\mathsf{x} \leftarrow$ V-cycle $(\mathsf{A}, \mathsf{x}, \mathsf{u}^l, 0)$                // see Alg. 9
```

Algorithm 9: $\mathsf{x}_{new} = \textit{V-cycle}\,(\mathsf{A}, \mathsf{x}_{old}\,, \mathsf{u}^l, h)$

```
Param. : $P_1, P_2, \cdots, P_H$,                    // hierarchy of prolongations
         $\mu_{pre}$, $\mu_{post}$                     // pre- and post-relaxation iterations
Input  : $A$,                                        // left-hand-side system matrix
         $x_{old}$,                                  // current solution
         $u^l$,                                      // right-hand-side of the linear system
         $h$,                                        // current multigrid level
Output: $x_{new}$                                    // new solution
1.  if $h$ is not the coarsest level $H$ then
2.      // PRE-RELAXATION
3.      $x'_{old} \leftarrow \textit{Relaxation}\,(A, x_{old}\,, u^l, \mu_{pre})$
4.      // COARSE-GRID CORRECTION
5.      $r_{h+1} \leftarrow P^\top_{h+1}(u^l - Ax'_{old})$                // restrict residual
6.      $c_{h+1} \leftarrow \textit{V-cycle}\,(P^\top_{h+1} A P_{h+1}, 0, r_{h+1}, h+1)$
7.      $c_h \leftarrow P_{h+1} c_{h+1}$                              // prolong correction
8.      $x'_{new} \leftarrow x_{old} + c_h$                           // update solution
9.      // POST-RELAXATION
10.     $x_{new} \leftarrow \textit{Relaxation}\,(A, x'_{new}\,, u^l, \mu_{post})$;
11. else
12.     solve $Ax_{new} = u^l$                                   // direct solve
13. return $x_{new}$
```

put mesh. This pre-computation takes the vertex and face lists as the inputs, and outputs a series of prolongation operators $\mathsf{P}_1, \cdots, \mathsf{P}_H$ for different levels on the hierarchy. After building the linear system A, u^l, one can run the V-cycle until getting the desired accuracy.

In Alg. 9, we summarize the pseudo code of the V-cycle algorithm which consists of two procedures: relaxation and coarse-grid correction. For the relaxation step, we use the standard serial Gauss-Seidel method. In the coarse-grid correction step, the process is well-defined given the prolongation operator P. We start by restricting the residual to the coarser level via $\mathsf{P}^\top$, solving a coarsened linear system with the left-hand-side defined as $\mathsf{P}^\top\mathsf{A}\mathsf{P}$, prolonging the low-res solution back to the fine domain using P, and using it to update the current high-res solution. We can further accelerate the computation by storing the system matrix hierarchy $\mathsf{A}_{h+1} = \mathsf{P}^\top_{h+1}\mathsf{A}_h\mathsf{P}_{h+1}$ to save some redundant computation.

In terms of hyperparameters of our multigrid method, we conduct an ablation study summarized in App. 8.9.4. In each V-cycle, we use the Gauss-Seidel relaxation with 2 pre- and post-relaxation iterations. Our default setup coarsens the geometry down to 0.25 of the number of vertices at its previous level until we reach the coarsest mesh with no fewer than 500 vertices. Note that we do not recommend to coarsen the mesh to an extreme. In Fig. 8.17, we show that an extremely aggressive coarsening often hurts the performance because the coarsest mesh fails to represent the target solution. In terms of stopping criteria, the accuracy ε that allows us to get visually indistinguishable results compared to the

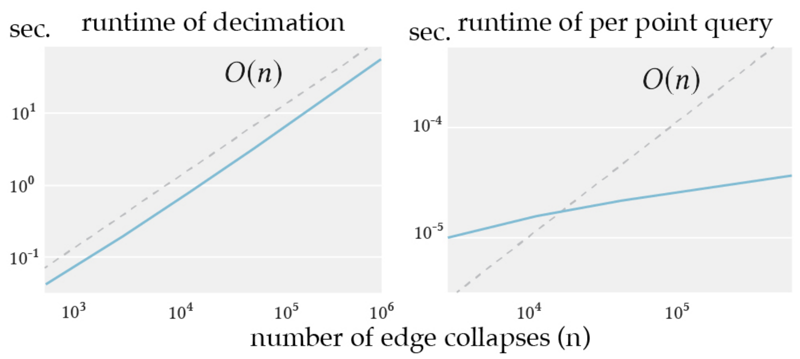


Figure 8.18: On the left, we report the runtime of our successive self-parameterization for constructing the bijective map. On the right, we report the query time of a single point through the bijective map.

ground truth depends on the problem and the size of the mesh. In our experiments, we set it to be $10^{-3} \geq \varepsilon \geq 10^{-5}$. Our experiments suggest that the optimal set of parameters that minimizes the wall-clock runtime depends on the geometry and the PDE of interest. But we use our default parameters for all our experiments in Sec. 8.6 for consistency.

We implement our algorithm in C++ with Eigen and evaluate our method on a MacBook Pro with an Intel i5 2.3GHz processor. In comparison with the Cholesky solver where pre-factorization is required whenever the system matrix A is changed, our multigrid solver leads to orders of magnitude speed-ups (see Fig. 8.21).

Boundary conditions Our implementation currently supports natural boundary conditions (Fig. 8.21 right), zero Neumann boundary conditions (Fig. 8.21 left), and the Dirichlet boundary condition. We handle the Dirichlet constrains by reformulating the system using only the unknown variables. This results in a reduced and unconstrained linear system, allowing us to solve it as usual. For more details, please refer to App. 8.9.2.

Successive Self-Parameterization We report the runtime of our pre-computation (querying points and self-parameterization) in Fig. 8.18 and detail the implementation in App. 8.9.3. Note that this pre-computation solely depends on the geometry. We only need to do this computation once for each shape and we can reuse the same hierarchy for many different linear systems. Thus, in our runtime comparisons in Sec. 8.6, we do not include the runtime of this pre-computation.

8.6 Applications

We evaluate our method on a variety of geometry processing applications that involve solving linear systems as a subroutine. We especially focus on the case where the system matrix A is changing due to different time steps (e.g., simulation) or user interactions (e.g., data

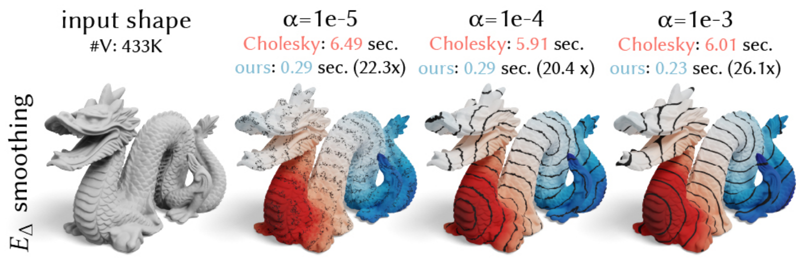


Figure 8.19: In the data smoothing application, one would adjust the smoothness parameter α until getting the desired smoothness. Using direct solvers (e.g., Cholesky) would need to recompute the expensive pre-factorization, but our multigrid solver can reuse the same hierarchy and leads to interactive performance.

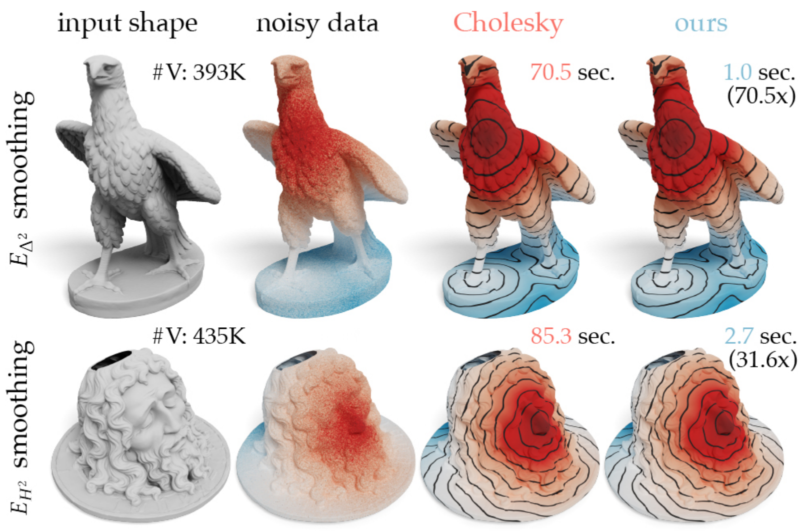


Figure 8.20: We evaluate our method on data smoothing with different smoothness energies, including the Bilaplacian E_{Δ^2} and the squared Hessian E_{H^2}. Our method is orders of magnitude faster than the direct solver.

smoothing). In our experiments, we ignore the multigrid pre-computation and compare our multigrid V-cycle (in blue) against the runtime of both the factorization and the solving time combined of the Cholesky solver (in red) because both these steps are required when A is changing. We also pick the applications that involve different system matrices with different sparsity patterns. This includes the cotangent Laplacian (1-ring sparsity), the Bilaplacian (2-ring sparsity), the squared Hessian (2-ring sparsity) [Stein et al., 2020], a system matrix derived from Lagrange multipliers [Azencot et al., 2015], and also the Hessian matrices from shell simulation which has $3|V|$-by-$3|V|$ dimensionality.

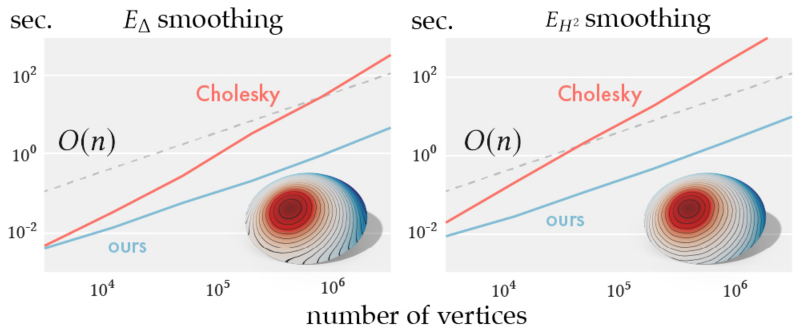


Figure 8.21: We compare the runtime of our multigrid solver against the Cholesky solver on smoothing the noisy data on a sphere cap at different resolutions until reaching a sufficiently small mean squared error (visually indistinguishable). We evaluate the smoothing with the Dirichlet energy E_Δ (1-ring sparsity) and with the squared Hessian energy E_{H^2} [Stein et al., 2020] (2-ring sparsity). Our method is asymptotically faster than the direct solver. On meshes with 200K vertices and 3 million vertices (using E_{H^2}), a serial implementation of our method is 39× and 231× faster, respectively.

Data smoothing Smoothing data on the surface is a fundamental task in geometry processing. We often treat it as an energy minimization problem

$$x^* = \arg\min_x \alpha E_s(x) + (1-\alpha)\int_\Omega \|x - f\|^2 dx, \tag{8.6}$$

where α is the parameter controlling the smoothness, f is the input noisy function, and E_s is an energy of choice, measuring the smoothness of the output signal x. As a different input f may contain a different amount of noise, one would typically adjust the α or the smoothness energy E_s until getting the desired smoothness. However, these adjustments boil down to solving a different linear system. When using direct solvers, this requires recomputing the factorization in order to solve the system. In comparison, using our multigrid allows one to reuse the same precomputed multigrid hierarchy and leads to orders of magnitude speed-ups (see Fig. 8.19). We evaluate our method on different smoothness energies, including the Dirichlet energy E_Δ (Fig. 8.19), the squared Laplacian energy E_{Δ^2} (Fig. 8.20 top), and the squared Hessian energy E_{H^2} [Stein et al., 2020] (Fig. 8.20 bottom). In Fig. 8.21, we quantitatively evaluate the runtime on the same shape at different resolutions obtained via subdivision. On a mesh with millions of vertices, our approach has over 100× speed-ups. With our multigrid setup, the precomputed prolongation operator can be reused not only when changing the value of α (full rank update to A), but also when swapping between energies E_s.

Mesh deformation We also evaluate our method on mesh deformations to demonstrate that even though the vertex positions have changed, as long as the connectivity of the mesh remains the same, we can still reuse the same multigrid hierarchy computed on the rest

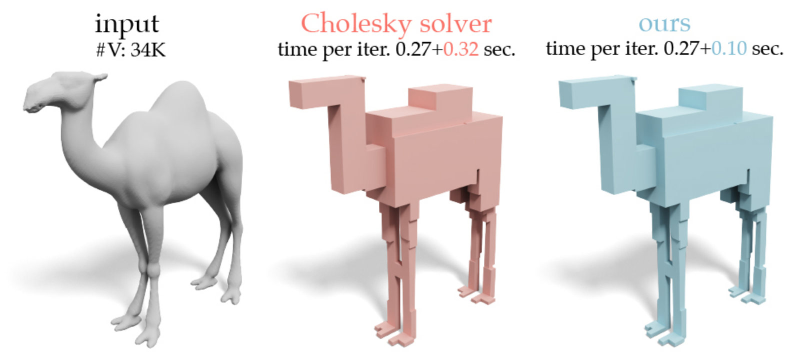


Figure 8.22: We compare our multigrid against the direct solver on the polycube deformation proposed by [Zhao et al., 2017]. Although the vertex positions are changed at every iteration, we can still reuse the precomputed multigrid hierarchy because the connectivity remains the same. We report the runtime of other steps in the algorithm in black and the runtime for solving linear systems in red and in blue.

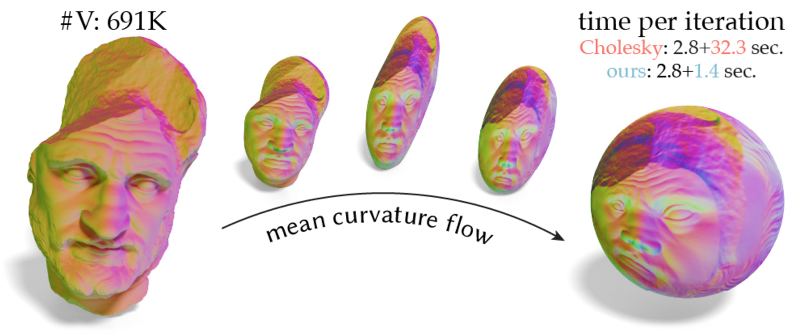


Figure 8.23: Running the mean curvature flow [Kazhdan et al., 2012] requires to update the system matrix at every step according to the mass matrix of the current mesh. By reusing the hierarchy computed on the input shape (left), our multigrid method is orders of magnitude faster than the direct solver. We report the runtime of other subroutines in black and the time for solving the linear system in red (direct solver) and in blue (our multigrid). ©model by Oliver Laric under CC BY-NC-SA.

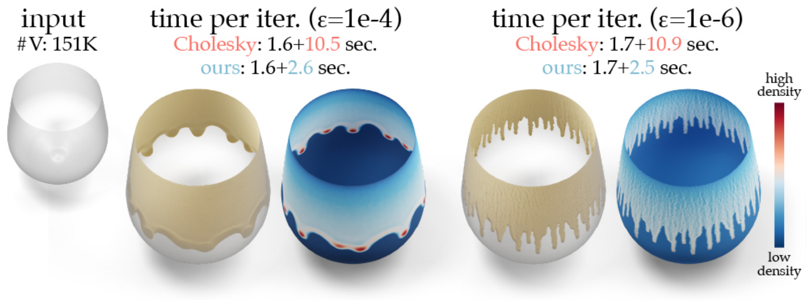


Figure 8.24: The surface fluid simulation [Azencot et al., 2015] involves solving different linear systems at each time step. Our method reuses the precomputed hierarchy and leads to a faster solver in contrast to the direct solver. We split the runtime of other procedures (black) and the runtime of solving the linear system (red and blue). Note that this runtime comparison is in MATLAB using the original implementation from [Azencot et al., 2015].

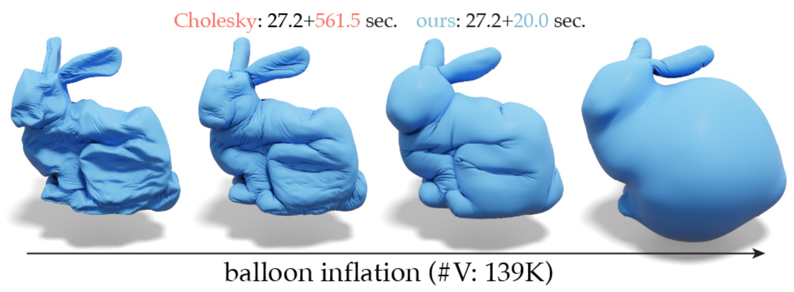


Figure 8.25: Replacing the Cholesky solver with our surface multigrid method, we can accelerate the linear solve part in the balloon simulation proposed by [Skouras et al., 2012] by 28× so that solving linear systems becomes no longer the bottleneck of the algorithm.

Table 8.1: Multigrid runtime.

profile (sec.)	Fig. 8.23
precompute	50.6
total solve time	1.44
1. prepare $P^\top AP$	0.72
2. relaxation	0.38
3. prolong & restrict	0.10
4. get residual norm	0.10
5. others	0.14

mesh. One possible intuition is to view the deformation field on vertices as a function on the rest shape. Thus, a hierarchy built on the rest shape could still be used to compute the deformation "function". In Fig. 8.22, we evaluate our method on a polycube deformation method proposed by [Zhao et al., 2017] whose system matrix is re-built at every iteration based on the current deformed mesh. Our method accelerates the algorithm by 3.2 × on a relatively low-resolution mesh. In Fig. 8.23, we replace the Cholesky solver with our method on a mean curvature flow method proposed in [Kazhdan et al., 2012] and achieve 23× speedup.

In many simulation algorithms, the system matrix A changes at every time step. In Fig. 8.24, we demonstrate the speed-up of our multigrid solver on a surface fluid simulation [Azencot et al., 2015]. Note that the surface fluid simulation is evaluated in MATLAB (for both the direct solver and our multigrid) respecting the original implementation. In Fig. 8.25, we evaluate our method on a balloon simulation method proposed by [Skouras et al., 2012]. Due to the speedup of our multigrid method, we shift the bottleneck of balloon simulation away from solving linear systems.

8.7 Discussion

In our experiments in Sec. 8.6, we evaluate our runtime using a simple serial implementation of our method. In Table 8.1, we further provide a detailed runtime decomposition of the experiment in Fig. 8.23 as a representative example. We can observe that preparing the matrix hierarchy $\mathsf{P}^\top \mathsf{A} \mathsf{P}$ and doing the relaxation take most of the time when solving a linear system. Thus, we can achieve even more speedup if we leverage the structure of the problem when computing the matrix hierarchy, such as the data smoothing detailed in App. 8.9.5, or a parallel implementation of the entire V-cycle. To validate our hypothesis, our initial attempt uses the CPU to parallelize the Gauss-Seidel method based on graph coloring. This reduces the runtime of our relaxation from 0.38 seconds down to 0.17 seconds ($2.2\times$ speedup) for the experiment in Fig. 8.23. Similarly, for the top and the bottom examples in Fig. 8.20, the fast Gauss-Seidel accelerates the relaxation by $1.8\times$ and $3.3\times$ repetitively. We provide details about our graph coloring Gauss-Seidel in the App. 8.9.1 for completeness. An even higher speed-up can be expected via a GPU implementation of the Gauss-Seidel relaxation (cf. [Fratarcangeli et al., 2016]). Besides the Gauss-Seidel relaxation, parallelizing the entire solver could also be an interesting future direction to accelerate our method.

8.8 Limitations & Future Work

We present a geometric multigrid solver for triangulated curved surfaces. Multigrid methods are asymptotically faster than direct solvers, thus it offers a promising direction for *scalable* geometry processing. Our multigrid method can obtain a fast approximation of the solution with orders of magnitude speedup. However, obtaining a highly accurate solution would require more iterations which results in a less significant speed-ups. For higher-order problems, our method may not converge to high accuracy because our choice of linear interpolation is insufficient [Hemker, 1990]. Thus, exploring high-order prolongation (e.g., subdivision barycentric coordinates [Anisimov et al., 2016]) or learning-based prolongation (e.g, [Katrutsa et al., 2020]) would also be valuable directions to improve the solver. Another interesting direction to improve the solver is to use our multigrid solver as the pre-conditioner for other solvers such as the *conjugate gradient* method.

Developing a reliable and robust surface multigrid solver would be an important next step. Our current solver is more sensitive to the triangle quality of the input mesh compared to the existing direct solver. In our experiments, we remesh troublesome input shapes using available methods [Jakob et al., 2015; Hu et al., 2020; Schmidt and Singh, 2010]. A better future approach would be extending our self-parameterization to the entire remeshing process, to maintain bijectivity from the remeshed object to the input mesh. Having a deeper understanding of the relationship between the convergence and mesh quality would give insights towards developing a suitable remeshing algorithm for surface multigrid solvers.

Achieving this may also require theoretical tools to estimate the convergence property, such as extending the *Local Fourier Analysis* from subdivision meshes [Gaspar et al., 2009] to generic unstructured meshes. Once surface multigrid has become a reliable solver for linear systems on manifold meshes, generalizing it to non-manifolds or point clouds would be another exciting future direction.

Another avenue for future work is to further optimize each component of the prolongation construction and multigrid solver routines. Although our method outputs a bijective map in most cases, bijectivity is not guaranteed. A more rigorous analysis is required to identify potential edge cases that may result in non-bijective maps. Currently, we use off-the-shelf simplification and distortion objectives (as-rigid-as-possible [Liu et al., 2008] and conformal [Lévy et al., 2002] energies), but these methods that are designed for other purposes may not be the optimal ones for surface multigrid methods. For instance, we notice that the distortion in the self-parameterization is not closely correlated to the convergence of our multigrid solver (see Fig. 8.14). We however use the off-the-shelf parameterization energy designed to measure the distortion in our multigrid solver. Developing simplification and parameterization methods tailored-made for multigrid solver performance could further improve eventual solver speed.

The relationship between multigrid convergence and bijectivity requires a deeper understanding. Although we empirically demonstrate the superior performance of our prolongation compared to other non-bijective prolongations, bijectivity is not required for a multigrid method to converge. In our construction, we even pay the price of high distortion to achieve bijectivity along the boundary (zoom in Fig. 8.6). Thus, a deeper understanding of the connections between distortion, bijectivity, and multigrid convergence is important to reach optimal performance.

8.9 Appendix

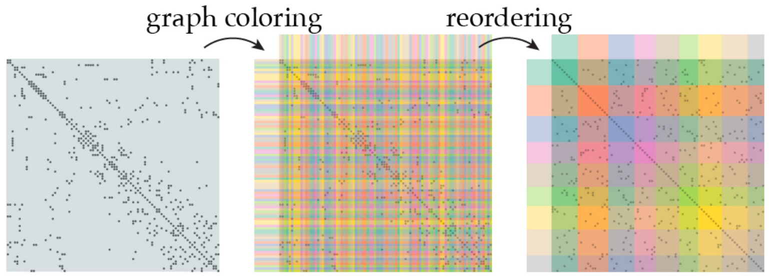


Figure 8.26: Given a system matrix with the sparsity pattern showing on the left, we first use a greedy graph coloring approach detailed in Alg. 10 to "paint" the variables that are independent of each other with the same color (middle). Then we perform reordering to group the variables with the same color together (right) to parallelize our Gauss-Seidel relaxation.

Algorithm 10: Graph Coloring

```
1. sort nodes $\{n_i\}$ by degree
2. $pallette \leftarrow \{\}$
3. for each node $n_i$ do
4.     $N \leftarrow$ gather colors from painted neighbors of $n_i$
5.     $c \leftarrow$ find first entry in $pallette$ not occurring in $N$
6.     if $c$ is not found then
7.         $c \leftarrow$ new color
8.         append $c$ to $pallette$
9.     else
10.        move $c$ to back of $pallette$
11.    paint $n_i$ with color $c$
```

8.9.1 Multi-Color Gauss Seidel

Our multigrid method spends a lot of the runtime on the Gauss-Seidel relaxation. We further accelerate the Gauss-Seidel relaxation by exploiting graph coloring (see Fig. 8.26), a standard optimization. Specifically, we treat the non-zero off-diagonal entries of a given sparse matrix A as a graph. We color this graph so that each node has a different color from its neighbors using a simple modification of the method proposed by Welsh and Powell [1967], summarized in Alg. 10 and repeated here for completeness. We color each node in descending order of degree. When considering node i, we try each color from a list of k colors that have been previously used for nodes $(1, \cdots, i-1)$. A color choice is valid if not matching any of the previously colored neighbors of node i. If valid, node i is colored and that color is moved to the back of the list. If no valid color is found in the list, a new color is used and added to the back of the list. This algorithm has $O(|V|log|V| + |E|k)$ runtime and $O(|V| + |E|)$ memory complexity, respectively, where k is the number of output colors (for sparse matrices, $k \ll |V|$). Although suboptimal (finding the optimal coloring is NP-complete), it handily outperforms the method of [Fratarcangeli et al., 2016] in runtime, memory usage, and color parsimony. By moving selected colors dynamically to the back of the list, we achieve better color balance (see, e.g., Fig. 8.26) than considering the list in fixed order of insertion.

8.9.2 Dirichlet Boundary Conditions

Solving a linear system $\mathsf{A}\mathsf{x} = \mathsf{u}^l$ is equivalent to minimizing a quadratic energy

$$E(\mathsf{x}) = \frac{1}{2}\mathsf{x}^\top \mathsf{A}\mathsf{x} - \mathsf{x}^\top \mathsf{u}^l \tag{8.7}$$

where one can derive the same linear system by setting $\partial E/\partial \mathsf{x} = 0$. One way to handle Dirichlet boundary conditions $\mathsf{x}(known) = \mathsf{c}$ is to reformulate the quadratic energy using only the unknown variables. Here we use *known* and *unknown* to represent indices of knowns and unknowns. We further use $\mathsf{x}_k = \mathsf{x}(\textit{known})$ to denote known variables and $\mathsf{x}_u = \mathsf{x}(\textit{unknown})$ for unknown variables in x. For matrices, we follow the same notation. For example, we use $\mathsf{A}_{uk} = \mathsf{A}(\textit{unknown}, \textit{known})$ to represent the corresponding sub-block in matrix A. We then rewrite the energy as (assuming A is symmetric)

$$E(\mathsf{x}_u) = \frac{1}{2}\mathsf{x}_u^\top \mathsf{A}_{uu}\mathsf{x}_u + \mathsf{x}_u^\top \mathsf{A}_{uk}\mathsf{x}_k - \mathsf{x}_u^\top \mathsf{u}_u^l + \text{constant}. \tag{8.8}$$

By setting the derivative to zero, we can derive a reduced linear system for only unknowns

$$\underbrace{\mathsf{A}_{uu}}_{\text{LHS}}\mathsf{x}_u = \underbrace{-\mathsf{A}_{uk}\mathsf{x}_k + \mathsf{u}_u^l}_{\text{RHS}} \tag{8.9}$$

We can leverage the same trick to incorporate Dirichlet constraints in the multigrid system. We use x_c to denote the coarse variable such that $\mathsf{x} = \mathsf{P}\mathsf{x}_c$ where the P is the Galerkin prolongation operator. We can then write the unknowns as

$$\mathsf{x}_u = \mathsf{P}_{u:}\mathsf{x}_c \tag{8.10}$$

where $\mathsf{P}_{u:} = \mathsf{P}(\textit{unknown}, :)$ (MATLAB notation) represents the rows of P that correspond to the *unknown* indices. Adding this to Eq. (8.11) leads to

$$\begin{aligned} E(\mathsf{x}_c) &= \frac{1}{2}(\mathsf{P}_{u:}\mathsf{x}_c)^\top \mathsf{A}_{uu}(\mathsf{P}_{u:}\mathsf{x}_c) + (\mathsf{P}_{u:}\mathsf{x}_c)^\top \mathsf{A}_{uk}\mathsf{x}_k & (8.11)\\ &\quad - (\mathsf{P}_{u:}\mathsf{x}_c)^\top \mathsf{u}_u^l + \text{constant}. & (8.12)\\ &= \frac{1}{2}\mathsf{x}_c^\top \mathsf{P}_{u:}^\top \mathsf{A}_{uu}\mathsf{P}_{u:}\mathsf{x}_c + \mathsf{x}_c^\top \mathsf{P}_{u:}^\top \mathsf{A}_{uk}\mathsf{x}_k & (8.13)\\ &\quad - \mathsf{x}_c^\top \mathsf{P}_{u:}^\top \mathsf{u}_u^l + \text{constant}. & (8.14) \end{aligned}$$

Similarly, setting the derivative with respect to x_c results in

$$\underbrace{\mathsf{P}_{u:}^\top \mathsf{A}_{uu}\mathsf{P}_{u:}}_{\text{reduced LHS}}\mathsf{x}_c = \underbrace{\mathsf{P}_{u:}^\top(-\mathsf{A}_{uk}\mathsf{x}_k + \mathsf{u}_u^l)}_{\text{reduced RHS}} \tag{8.15}$$

We can notice that, except at the second finest level where we need to extract the rows in prolongation that correspond to the *unknowns* $\mathsf{P}_{u:}$, we can solve the linear system at coarser levels without worrying about the constraints.

Another special case may occur when there are too many *known* indices. If too many variables in x are given the reduced system $\mathsf{P}_{u:}^\top \mathsf{A}_{uu}\mathsf{P}_{u:}$ may have completely zero rows/columns. To handle this edge case, we further remove the columns on $\mathsf{P}_{u:}$ where the maximum value is zero and the corresponding rows in the prolongation at the next level.

8.9.3 Successive self-parameterization

Our pre-computation of multigrid hierarchy involves decimating the triangle mesh with successive self-parameterization and mapping vertices on the fine mesh to the coarse mesh to obtain their barycentric coordinates. We report the runtime of both pre-computation steps in Fig. 8.18.

Implementing successive self-parameterization only requires a small change to an existing edge collapse algorithm. Specifically, right after collapsing a single edge, the only modification is to use the method described in Sec. 8.4.2 and Sec. 8.4.3 to formulate the joint variable and then flatten both patches to a consistent UV domain. To determine whether the collapse and the flattening is valid, we refer to the Sec. 4.8.3 for more details. During the querying stage, for a given query point represented as barycentric coordinates, we simply go through the list of local joint UV parameterization we stored from the decimation stage and update the barycentric coordinates successively using the method described in Fig. 8.8. We pre-store the face indices involved in each edge collapse so that for each query point, we can easily check whether this point is involved in the collapse via checking the face indices.

8.9.4 Ablation Study

In addition to the prolongation operator, the hyperparameters of a multigrid method also play a role in the convergence behavior. In terms of stopping criteria, the accuracy ε depends on the problem and the size of the mesh. We usually set $10^{-5} \leq \varepsilon \leq 10^{-}$ in order to get visually indistinguishable results compared to the ground truth. Using a reasonable initialization, such as the vertex positions in the previous iteration in mesh deformation, would further reduce the number of iterations to get the desired accuracy. For other hyperparameters, we conduct ablation studies on the choice of relaxation methods, pre-/post-relaxation iterations μ_{pre}, μ_{post}, and the coarsening ratio between consecutive levels (see Fig. 8.27). In terms of the relaxation methods, Gauss-Seidel is usually the go-to choice due to its effectiveness in smoothing out the high-frequency error. Practitioners may also prefer the (damped) Jacobi because it is faster and easier to parallelize, even though each iteration is less effective. In terms of the number of relaxation iterations, usually a couple of iterations (2 or 3) are sufficient to handle the high-frequency error. While we also notice that some multigrid methods (e.g., [Xian et al., 2019]) use lower-order prolongation with many more relaxation iterations to compensate for the inter-grid transfer error. In terms of coarsening ratio, using a less aggressive coarsening (e.g., 0.5) could reduce the error caused by the inter-grid transfer, but it often results in a bigger multigrid hierarchy and a longer runtime per cycle. On the other hand, using a more aggressive coarsening often leads to large inter-grid transfer error and slow convergence. Our default setup coarsens the geometry down to 0.25 of its previous resolution until we reach the coarsest mesh with

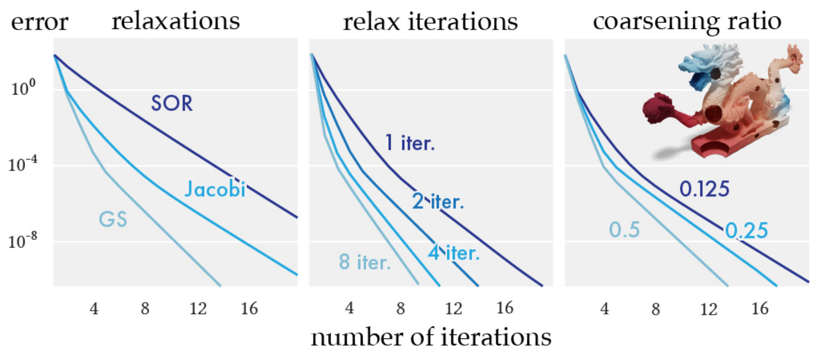


Figure 8.27: We conduct an ablation study on the multigrid hyperparameters, including the relaxation method (left), the number of relaxation iterations (middle), and the coarsening ratio (right). ©model by Oliver Laric under CC BY-NC-SA.

no fewer than 500 vertices. In each V-cycle, we use the Gauss-Seidel relaxation with 2 pre- and post-relaxation iterations. Our experiments suggest that the optimal set of parameters that minimizes the wall-clock runtime depends on the geometry and the PDE of interest. But we use our default parameters for all our experiments in Sec. 8.6 for consistency.

8.9.5 Fast Data Smoothing

When we discretize the data smoothing energy Eq. (8.6), we often arrive the following linear system

$$\underbrace{(\alpha \mathsf{Q} + (1-\alpha)\mathsf{M})}_{\mathsf{A}}\,\mathsf{x} = (1-\alpha)\mathsf{M}\mathsf{f} \tag{8.16}$$

where Q is a matrix that depends on the choice of the smoothness energy, M is the mass matrix, f is the noisy function, and α is the smoothness parameter. In order to build the coarsened system matrix, a straightforward implementation would be doing $\mathsf{P}^\top\mathsf{A}\mathsf{P}$ directly, but we can actually split the computation via

$$\mathsf{P}^\top\mathsf{A}\mathsf{P} = \alpha(\mathsf{P}^\top\mathsf{Q}\mathsf{P}) + (1-\alpha)(\mathsf{P}^\top\mathsf{M}\mathsf{P}). \tag{8.17}$$

Then we can pre-compute $\mathsf{P}^\top\mathsf{Q}\mathsf{P}$ and $\mathsf{P}^\top\mathsf{M}\mathsf{P}$ even before knowing the parameter α. As a results, during the online stage when a user adjusts α, we only require an efficient matrix addition to compute the system matrices for all the multigrid levels.

Chapter 9

Conclusion

3D content creation has enabled breakthroughs in engineering and manufacturing industries. However, existing tools are still insufficient to unlock future experiences, such as customized manufacturing, 3D virtual shopping, and the metaverse. This is because using existing modeling tools requires professional training, thus preventing the general public from using them to create, for instance, their digital avatars. The geometric stylization algorithms we explored in this thesis demonstrate the possibility of significantly lowering the difficulty in manipulating 3D shapes.

We began by showing how *rendering* can be used, not as a visualization tool, but as a "translator" to generalize image filters to filter 3D objects (Chapter. 2). By a careful treatment on rendering parameters, camera sampling, and mesh surgery, we can analytically differentiate through the rendering function and enable one to plug-and-play image filters on 3D meshes. In Chapter. 3, we further show that how this differentiable renderer can be used improve robustness of deep learning image classifers against geometry and lighting changes.

We demosntrate the effectiveness of geometric machine learning in capturing detailed geometric styles and transferring from one shape to another in Chapter. 4. This is enabled by a novel data-efficient self-supervised training formulation which can learn from only a single or a few training shapes. Furthermore, we provide solutions to handle challenges in mesh-based geometric learning, such as irregular mesh structure, various discretizations, and rigid transformations.

We propose variational methods to turn a geometry into the cubic style (Chapter. 5). Our key contribution is to define a mathematical formula to characterize the style of the cubic geometry and a reformulation to enable efficient optimization. The resulting algorithm is an easy-to-use and interactive stylization framework. In Chapter. 6, we show how this formulation can be further extended to different polygonal styles, in addition to cubes.

We further expand our topics to numerical tools for assisting interactive geometric stylization and computation. In Chapter. 7, we focus on high-quality simplification of a 3D shape where the quality is measured by the spectral property of the model. Our main contribution is a commutative energy for measuring the spectral similarity before and after the simplification process. This energy enables us to do spectrum-preserving simplification in order to maintain the quality of the solutions on coarsened meshes.

However, some applications (e.g., physics simulation) desire working on high-resolution models to capture high-frequency details in the solution. We provide an alternative route by proposing a scalable multigrid solver to efficiently solve linear systems defined on surface trianglular meshes. The key contribution is a self-parameterization technique to build the multigrid hierarchy and define the inter-resolution transfer operators. Our approach enables generalizing the scalable multigrid solvers to surface meshes.

9.1 Future Directions

However, the algorithms we explore in the thesis are merely the first steps towards a broader goal of making generic 3D content creation fast and accessible to everyone. Here are several important questions for future exploration.

Parallel Mesh Data-Structures Accelerating and scaling up geometry processing requires to harness the power of parallel computation. However, common data-structures for storing meshes (e.g., list of vertices/faces and half-edge data-structure) are difficult to deploy in parallel. Can we develop new mesh data-structure that can support existing geometry processing applications in parallel?

Robust & Scalable Solvers Our surface multigrid method in Chapter. 8 has demonstrated the promise in utilizing multigrid methods to scale up geometry processing, but it still suffers from mesh defects and does not fully leverage the parallel computation power. Can we develop a robust multigrid methods with parallel processes to extend geometry processing to the scale of billions of vertices?

Robust Geometric Learning Geometric learning is a fundamental building block for developing data-driven geometry processing tools. However, existing geometric learning models fail to operate to real-world "dirty" geometric data with defects, such as non-manifold, terrible triangle quality. Can we develop novel ingredients to support robust geometric machine learning?

Geometric Learning on Large Data In this thesis, we demonstrate the effectiveness of learning from scarce geometric data. These data-efficient methods are capable of reasoning about lower-level geometric features but are limited to capturing high-level semantics. With the assistance of *robust geometric learning*, can we learn from a large amount of raw geometric data to achieve high-level shape understandings?

Data-Driven 3D Content Creation This thesis only focuses on a sub-problem, geometric stylization, within the larger field of 3D content creation. With advances in 3D capturing technologies (e.g., scanning, photogrammetry), geometric learning, and large geometric datasets, can we extend the vision to renovating the entire 3D modeling pipeline with data-driven approaches?

Bibliography

Aanjaneya, M., Han, C., Goldade, R., and Batty, C. (2019). An efficient geometric multigrid solver for viscous liquids. *Proceedings of the ACM in Computer Graphics and Interactive Techniques*.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282.

Adams, M. and Demmel, J. (1999). Parallel multigrid solver for 3d unstructured finite element problems. In *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 1999, November 13-19, 1999, Portland, Oregon, USA*, page 27. ACM.

Adobe Inc. (2021). Adobe photoshop.

Aigerman, N., Poranne, R., and Lipman, Y. (2014). Lifted bijections for low distortion surface mappings. *ACM Transactions on Graphics (TOG)*, 33(4):69.

Aigerman, N., Poranne, R., and Lipman, Y. (2015). Seamless surface mappings. *ACM Transactions on Graphics (TOG)*, 34(4):72.

Akenine-Möller, T., Haines, E., and Hoffman, N. (2008). *Real-time rendering*. CRC Press.

Akhtar, N. and Mian, A. S. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430.

Aksoylu, B., Khodakovsky, A., and Schröder, P. (2005). Multilevel solvers for unstructured surface meshes. *SIAM J. Sci. Comput.*, 26(4):1146–1165.

Alexa, M. (2021). Polycover: Shape approximating with discrete surface orientation. *IEEE Computer Graphics and Applications*.

Andersen, M. S., Dahl, J., and Vandenberghe, L. (2010). Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones. *Mathematical Programming Computation*, 2(3).

Andreux, M., Rodola, E., Aubry, M., and Cremers, D. (2014). Anisotropic laplace-beltrami operators for shape analysis. In *European Conference on Computer Vision*, pages 299–312. Springer.

Anisimov, D., Deng, C., and Hormann, K. (2016). Subdividing barycentric coordinates. *Computer Aided Geometric Design*, 43:172–185.

Arfken, G. B. and Weber, H. J. (1999). Mathematical methods for physicists.

Athalye, A., Engstrom, L., Ilyas, A., and Kwok, K. (2017). Synthesizing robust adversarial examples.

Attene, M. (2010). A lightweight approach to repairing digitized polygon meshes. *The visual computer*, 26(11):1393–1406.

Attene, M. (2016). As-exact-as-possible repair of unprintable stl files. *arXiv preprint arXiv:1605.07829*.

Aubry, M., Schlickewei, U., and Cremers, D. (2011). The wave kernel signature: A quantum mechanical approach to shape analysis. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1626–1633. IEEE.

Azencot, O., Vantzos, O., Wardetzky, M., Rumpf, M., and Ben-Chen, M. (2015). Functional thin films on surfaces. In Barbic, J. and Deng, Z., editors, *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA 2015, Los Angeles, CA, USA, August 7-9, 2015*, pages 137–146. ACM.

Barill, G., Dickson, N., Schmidt, R., Levin, D. I., and Jacobson, A. (2018). Fast winding numbers for soups and clouds. *ACM Transactions on Graphics*.

Barron, J. T. and Malik, J. (2015). Shape, illumination, and reflectance from shading. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1670–1687.

Barrow, H. G., Tenenbaum, J. M., Bolles, R. C., and Wolf, H. C. (1977). Parametric correspondence and chamfer matching: Two new techniques for image matching. In Reddy, R., editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, pages 659–663. William Kaufmann.

Basri, R. and Jacobs, D. W. (2003). Lambertian reflectance and linear subspaces. *IEEE transactions on pattern analysis and machine intelligence*, 25(2):218–233.

Belkin, M., Sun, J., and Wang, Y. (2009). Constructing laplace operator from point clouds in r d. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1031–1040. Society for Industrial and Applied Mathematics.

Bell, W. N. (2008). Algebraic multigrid for discrete differential forms.

Berkiten, S., Halber, M., Solomon, J., Ma, C., Li, H., and Rusinkiewicz, S. (2017). Learning detail transfer based on geometric features. *Comput. Graph. Forum*, 36(2):361–373.

Bharaj, G., Levin, D. I., Tompkin, J., Fei, Y., Pfister, H., Matusik, W., and Zheng, C. (2015). Computational design of metallophone contact sounds. *ACM Transactions on Graphics (TOG)*, 34(6):223.

Bhat, P., Ingram, S., and Turk, G. (2004). Geometric texture synthesis by example. In Boissonnat, J. and Alliez, P., editors, *Second Eurographics Symposium on Geometry Processing, Nice, France, July 8-10, 2004*, volume 71 of *ACM International Conference Proceeding Series*, pages 41–44. Eurographics Association.

Bian, Z. and Hu, S.-M. (2011). Preserving detailed features in digital bas-relief making. *Computer Aided Geometric Design*, 28(4):245–256.

Blumenson, J. J. (1995). *Identifying American Architecture: A Pictorial Guide to Styles and Terms, 1600-1945*. Rowman Altamira.

Bobenko, A. I., Pottmann, H., and Rörig, T. (2020). Multi-nets. classification of discrete and smooth surfaces with characteristic properties on arbitrary parameter rectangles. *Discret. Comput. Geom.*, 63(3):624–655.

Bottou, L., Curtis, F. E., and Nocedal, J. (2016). Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*.

Bouaziz, S., Deuss, M., Schwartzburg, Y., Weise, T., and Pauly, M. (2012). Shape-up: Shaping discrete geometry with projections. *Comput. Graph. Forum*, 31(5):1657–1667.

Bouaziz, S., Martin, S., Liu, T., Kavan, L., and Pauly, M. (2014). Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11.

Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Brandt, A. (1977). Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390.

Brandt, A. and Livne, O. E. (2011). *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. SIAM.

Brandt, A., McCoruick, S., and Huge, J. (1985). Algebraic multigrid (amg) f0r sparse matrix equati0ns. *Sparsity and its Applications*, 257.

Briggs, W. L., McCormick, S. F., et al. (2000). *A multigrid tutorial*, volume 72. Siam.

Brochu, T. and Bridson, R. (2009). Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493.

Bronstein, A. M., Bronstein, M. M., and Kimmel, R. (2009). *Numerical Geometry of Non-Rigid Shapes*. Monographs in Computer Science. Springer.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42.

Burt, P. and Adelson, E. (1983). The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540.

Catmull, E. and Clark, J. (1998). *Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes*, page 183–188. Association for Computing Machinery, New York, NY, USA.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*.

Chao, I., Pinkall, U., Sanan, P., and Schröder, P. (2010). A simple geometric model for elastic deformations. *ACM transactions on graphics (TOG)*, 29(4):38.

Chaudhuri, S., Ritchie, D., Wu, J., Xu, K., and Zhang, H. R. (2020). Learning to generate 3d structures. In *Eurographics State-of-the-Art Report (STAR)*.

Chen, D., Levin, D. I., Matusik, W., and Kaufman, D. M. (2017a). Dynamics-aware numerical coarsening for fabrication design. *ACM Trans. Graph.*, 34(4).

Chen, D., Levin, D. I. W., Sueda, S., and Matusik, W. (2015). Data-driven finite elements for geometry and material design. *ACM Trans. Graph.*, 34(4).

Chen, J., Bao, H., Wang, T., Desbrun, M., and Huang, J. (2018). Numerical coarsening using discontinuous shape functions. *ACM Trans. Graph.*, 37(4).

Chen, J. and Safro, I. (2011). Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490.

Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. (2017b). Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM.

Chen, W., Wang, H., Li, Y., Su, H., Wang, Z., Tu, C., Lischinski, D., Cohen-Or, D., and Chen, B. (2016). Synthesizing training images for boosting human 3d pose estimation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 479–488. IEEE.

Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. (2008). Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3).

Chen, Z., Kim, V. G., Fisher, M., Aigerman, N., Zhang, H., and Chaudhuri, S. (2021). Decorgan: 3d shape detailization by conditional refinement. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Cherchi, G., Livesu, M., and Scateni, R. (2016). Polycube simplification for coarse layouts of surfaces and volumes. In *Computer Graphics Forum*, volume 35, pages 11–20. Wiley Online Library.

Cheshmi, K., Kaufman, D. M., Kamil, S., and Dehnavi, M. M. (2020). Nasoq: Numerically accurate sparsity-oriented qp solver. *ACM Trans. Graph.*, 39(4).

Choe, G., Park, J., Tai, Y., and Kweon, I. S. (2017). Refining geometry from depth sensors using IR shading images. *Inter. Journal of Computer Vision*.

Chuang, M., Luo, L., Brown, B. J., Rusinkiewicz, S., and Kazhdan, M. M. (2009). Estimating the laplace-beltrami operator by restricting 3d functions. *Comput. Graph. Forum*, 28(5):1475–1484.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136.

Cignoni, P., Montani, C., and Scopigno, R. (1998a). A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54.

Cignoni, P., Rocchini, C., and Scopigno, R. (1998b). Metro: measuring error on simplified surfaces. In *Computer graphics forum*, volume 17, pages 167–174. Wiley Online Library.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*.

Cohen, J., Olano, M., and Manocha, D. (1998). Appearance-preserving simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 115–122, New York, NY, USA. Association for Computing Machinery.

Cohen, J. D., Manocha, D., and Olano, M. (1997). Simplifying polygonal models using successive mappings. In *IEEE Visualization '97, Proceedings, Phoenix, AZ, USA, October 19-24, 1997*, pages 395–402. IEEE Computer Society and ACM.

Cohen, J. D., Manocha, D., and Olano, M. (2003). Successive mappings: An approach to polygonal mesh simplification with guaranteed error bounds. *Int. J. Comput. Geom. Appl.*, 13(1):61.

Cohen-Steiner, D., Alliez, P., and Desbrun, M. (2004). Variational shape approximation. *ACM Trans. on Graph.*

Corker-Marin, Q., Pasko, A., and Adzhiev, V. (2018). 4d cubism: Modeling, animation, and fabrication of artistic shapes. *IEEE computer graphics and applications*, 38(3):131–139.

Corman, E., Solomon, J., Ben-Chen, M., Guibas, L., and Ovsjanikov, M. (2017). Functional characterization of intrinsic and extrinsic geometry. *ACM Transactions on Graphics (TOG)*, 36(2):14.

Crane, K., Pinkall, U., and Schröder, P. (2011). Spin transformations of discrete surfaces. *ACM Trans. Graph.*, 30(4):104.

Crane, K., Weischedel, C., and Wardetzky, M. (2017). The heat method for distance computation. *Commun. ACM*, 60(11):90–99.

Dai, A. and Nießner, M. (2019). Scan2mesh: From unstructured range scans to 3d meshes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 5574–5583. Computer Vision Foundation / IEEE.

de Goes, F., Desbrun, M., Meyer, M., and DeRose, T. (2016). Subdivision exterior calculus for geometry processing. *ACM Trans. Graph.*, 35(4):133:1–133:11.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852.

Delaunoy, A. and Prados, E. (2011). Gradient flows for optimizing triangular mesh-based surfaces: Applications to 3d reconstruction problems dealing with visibility. *International journal of computer vision*, 95(2):100–123.

Deng, B., Pottmann, H., and Wallner, J. (2011). Functional webs for freeform architecture. *Comput. Graph. Forum*, 30(5):1369–1378.

DeRose, T., Kass, M., and Truong, T. (1998). Subdivision surfaces in character animation. In Cunningham, S., Bransford, W., and Cohen, M. F., editors, *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, pages 85–94. ACM.

Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. (1999). Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on*

Computer graphics and interactive techniques, pages 317–324. ACM Press/Addison-Wesley Publishing Co.

Dey, T. K., Edelsbrunner, H., Guha, S., and Nekhayev, D. V. (1999). Topology preserving edge contraction. *Publ. Inst. Math.(Beograd)(NS)*, 66(80):23–45.

Dick, C., Georgii, J., and Westermann, R. (2011). A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simul. Model. Pract. Theory*, 19(2):801–816.

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. (2018). Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Doo, D. (1978). A subdivision algorithm for smoothing down irregularly shaped polyhederons. *Computer Aided Design*, pages 157–165.

Doo, D. and Sabin, M. (1998). *Behaviour of Recursive Division Surfaces near Extraordinary Points*, page 177–181. Association for Computing Machinery, New York, NY, USA.

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Dumas, J., Lu, A., Lefebvre, S., Wu, J., and Dick, C. (2015). By-example synthesis of structurally sound patterns. *ACM Transactions on Graphics (TOG)*, 34(4):137.

Dunster, T. (2010). Legendre and related functions. *NIST handbook of mathematical functions*, pages 351–381.

Dyn, N., Levine, D., and Gregory, J. A. (1990). A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9(2):160–169.

El-Sana, J., Azanli, E., and Varshney, A. (1999). Skip strips: maintaining triangle strips for view-dependent rendering. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 131–138. IEEE Computer Society Press.

Eslami, S. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Hinton, G. E., et al. (2016). Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust Physical-World Attacks on Deep Learning Visual Classification. In *Computer Vision and Pattern Recognition (CVPR)*.

Fan, H., Su, H., and Guibas, L. J. (2017). A point set generation network for 3d object reconstruction from a single image. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2463–2471. IEEE Computer Society.

Fang, X., Xu, W., Bao, H., and Huang, J. (2016). All-hex meshing using closed-form induced polycube. *ACM Transactions on Graphics (TOG)*, 35(4):124.

Fernie, E. (1995). *Art history and its methods: a critical anthology*. Phaidon London.

Fiser, J., Jamriska, O., Lukác, M., Shechtman, E., Asente, P., Lu, J., and Sýkora, D. (2016). Stylit: illumination-guided example-based stylization of 3d renderings. *ACM Trans. Graph.*, 35(4):92:1–92:11.

Fisher, M., Springborn, B., Bobenko, A. I., and Schroder, P. (2006). An algorithm for the construction of intrinsic delaunay triangulations with applications to digital geometry processing. In *ACM SIGGRAPH 2006 Courses*, pages 69–74. ACM.

Floater, M. S. and Micchelli, C. A. (1997). Nonlinear stationary subdivision. *Journal of Approximation Theory*.

Fratarcangeli, M., Tibaldo, V., and Pellacini, F. (2016). Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph.*, 35(6):214:1–214:9.

Friedel, I., Schröder, P., and Khodakovsky, A. (2004). Variational normal meshes. *ACM Trans. Graph.*, 23(4):1061–1073.

Fu, X., Bai, C., and Liu, Y. (2016). Efficient volumetric polycube-map construction. *Comput. Graph. Forum*, 35(7):97–106.

Fumero, M., Möller, M., and Rodolà, E. (2020). Nonlinear spectral geometry processing via the TV transform. *ACM Trans. Graph.*, 39(6):199:1–199:16.

Gal, R., Sorkine, O., Popa, T., Sheffer, A., and Cohen-Or, D. (2007). 3d collage: expressive non-realistic modeling. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 7–14. ACM.

García Fernández, I., Xia, J., He, Y., Xin, S.-Q., and Patow, G. (2013). Interactive applications for sketch-based editable polycube map. *© IEEE Transactions on Visualization and Computer Graphics, 2013, vol. 19, núm. 7, p. 1158-1171*.

Gargallo, P., Prados, E., and Sturm, P. (2007). Minimizing the reprojection error in surface reconstruction from images. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

Garland, M. (1999). Multiresolution modeling: Survey & future opportunities. *State of the art report*, pages 111–131.

Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In Owen, G. S., Whitted, T., and Mones-Hattal, B., editors, *Proceedings of the 24th Annual*

Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997, pages 209–216. ACM.

Garland, M. and Heckbert, P. S. (1998). Simplifying surfaces with color and texture using quadric error metrics. In *Visualization'98. Proceedings*, pages 263–269. IEEE.

Gaspar, F. J., Gracia, J. L., and Lisbona, F. J. (2009). Fourier analysis for multigrid methods on triangular grids. *SIAM J. Sci. Comput.*, 31(3):2081–2102.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2414–2423.

Genova, K., Cole, F., Maschinot, A., Sarna, A., Vlasic, D., and Freeman, W. T. (2018). Unsupervised training for 3d morphable model regression. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Georgii, J. and Westermann, R. (2006). A multigrid framework for real-time simulation of deformable bodies. *Comput. Graph.*, 30(3):408–415.

Gilmer, J., Adams, R. P., Goodfellow, I., Andersen, D., and Dahl, G. E. (2018). Motivating the rules of the game for adversarial example research. *arXiv preprint arXiv:1807.06732*.

Gingold, Y. and Zorin, D. (2008). Shading-based surface editing. *ACM Transactions on Graphics (TOG)*, 27(3):95.

Giorgi, D., Biasotti, S., and Paraboschi, L. (2007). Shape retrieval contest 2007: Watertight models track. `http://watertight.ge.imati.cnr.it/`.

Gkioulekas, I., Zhao, S., Bala, K., Zickler, T., and Levin, A. (2013). Inverse volume rendering with material dictionaries. *ACM Transactions on Graphics (TOG)*, 32(6):162.

Gooch, B. and Gooch, A. (2001). *Non-photorealistic rendering*. AK Peters/CRC Press.

Goodfellow, I. (2018). Defense against the dark arts: An overview of adversarial example security research and future research directions. *arXiv preprint arXiv:1806.04169*.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Gouraud, H. (1971). Continuous shading of curved surfaces. *IEEE transactions on computers*, 100(6):623–629.

Gower, J. C., Dijksterhuis, G. B., et al. (2004). *Procrustes problems*, volume 30. Oxford University Press on Demand.

Green, R. (2003). Spherical harmonic lighting: The gritty details. In *Archives of the Game Developers Conference*, volume 56, page 4.

Green, S., Turkiyyah, G., and Storti, D. W. (2002). Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In Seidel, H., Shapiro, V., Lee, K., and Patrikalakis, N., editors, *Seventh ACM Symposium on Solid Modeling and Applications, Max-Planck-Institut für Informatik, Saarbrücken, Germany, June 17-21, 2002*, pages 265–272. ACM.

Gregson, J., Sheffer, A., and Zhang, E. (2011). All-hex mesh generation via volumetric polycube deformation. *Comput. Graph. Forum*, 30(5):1407–1416.

Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. (2018a). 3d-coded: 3d correspondences by deep deformation. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II*, volume 11206 of *Lecture Notes in Computer Science*, pages 235–251. Springer.

Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. (2018b). A papier-mâché approach to learning 3d surface generation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 216–224. IEEE Computer Society.

Gu, X., Gortler, S. J., and Hoppe, H. (2002). Geometry images. *ACM Trans. Graph.*, 21(3):355–361.

Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

Guskov, I., Khodakovsky, A., Schröder, P., and Sweldens, W. (2002). Hybrid meshes: multiresolution using regular and irregular refinement. In Hurtado, F., Sacristán, V., Bajaj, C., and Suri, S., editors, *Proceedings of the 18th Annual Symposium on Computational Geometry, Barcelona, Spain, June 5-7, 2002*, pages 264–272. ACM.

Guskov, I., Vidimce, K., Sweldens, W., and Schröder, P. (2000). Normal meshes. In Brown, J. R. and Akeley, K., editors, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000*, pages 95–102. ACM.

Habel, R., Mustata, B., and Wimmer, M. (2008). Efficient spherical harmonics lighting with the preetham skylight model. In *Eurographics* (*Short Papers*), pages 119–122.

Haber, T., Mertens, T., Bekaert, P., and Reeth, F. V. (2005). A computational approach to simulate subsurface light diffusion in arbitrarily shaped objects. In Inkpen, K. and van de Panne, M., editors, *Proceedings of the Graphics Interface 2005 Conference, May 9-11, 2005,* *Victoria, British Columbia, Canada*, pages 79–86. Canadian Human-Computer Communications Society.

Hackbusch, W. (2013). *Multi-grid methods and applications*, volume 4. Springer Science & Business Media.

Haeberli, P. (1990). Paint by numbers: abstract image representations. In Baskett, F., editor, *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1990, Dallas, TX, USA, August 6-10, 1990*, pages 207–214. ACM.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. (2019). Meshcnn: A network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):90.

He, K. and Sun, J. (2015). Fast guided filter. *CoRR*, abs/1505.00996.

He, K., Sun, J., and Tang, X. (2010). Guided image filtering. In *European conference on computer vision*, pages 1–14. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, L. and Schaefer, S. (2013). Mesh denoising via l 0 minimization. *ACM Transactions on Graphics (TOG)*, 32(4):64.

He, Y., Wang, H., Fu, C.-W., and Qin, H. (2009). A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics*, 33(3):369–380.

Hemker, P. (1990). On the order of prolongations and restrictions in multigrid procedures. *Journal of Computational and Applied Mathematics*, 32(3):423–429.

Henderson, L. D. (1983). *The Fourth Dimension and Non-Euclidean Geometry*. Princeton, Princeton University Press.

Hendrycks, D. and Dietterich, T. G. (2018). Benchmarking neural network robustness to common corruptions and surface variations. *arXiv preprint arXiv:1807.01697*.

Herholz, P. and Alexa, M. (2018). Factor Once: Reusing Cholesky Factorizations on Sub-Meshes. *ACM Transaction on Graphics (Proc. of Siggraph Asia)*, 37(6):9.

Herholz, P. and Sorkine-Hornung, O. (2020). Sparse cholesky updates for interactive mesh parameterization. *ACM Trans. Graph.*, 39(6):202:1–202:14.

Hertz, A., Hanocka, R., Giryes, R., and Cohen-Or, D. (2020). Deep geometric texture synthesis. *ACM Trans. Graph.*, 39(4):108.

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. (2001). Image analogies. In Pocock, L., editor, *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001, Los Angeles, California, USA, August 12-17, 2001*, pages 327–340. ACM.

Hertzmann, A., Oliver, N., Curless, B., and Seitz, S. M. (2002). Curve analogies. In Gibson, S. and Debevec, P. E., editors, *Proceedings of the 13th Eurographics Workshop on Rendering Techniques, Pisa, Italy, June 26-28, 2002*, volume 28 of *ACM International Conference Proceeding Series*, pages 233–246. Eurographics Association.

Hertzmann, A., O'Sullivan, C., and Perlin, K. (2009). Realistic human body movement for emotional expressiveness. In *ACM SIGGRAPH 2009 Courses*, page 20. ACM.

Hoppe, H. (1996). Progressive meshes. In Fujii, J., editor, *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*, pages 99–108. ACM.

Hoppe, H. (1997). View-dependent refinement of progressive meshes. In Owen, G. S., Whitted, T., and Mones-Hattal, B., editors, *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997*, pages 189–198. ACM.

Hoppe, H. (1999). New quadric metric for simplifying meshes with appearance attributes. In *Visualization'99. Proceedings*, pages 59–510. IEEE.

Hoppe, H., DeRose, T., Duchamp, T., Halstead, M. A., Jin, H., McDonald, J. A., Schweitzer, J., and Stuetzle, W. (1994). Piecewise smooth surface reconstruction. In Schweitzer, D., Glassner, A. S., and Keeler, M., editors, *Proceedings of the 21th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994, Orlando, FL, USA, July 24-29, 1994*, pages 295–302. ACM.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. A., and Stuetzle, W. (1993). Mesh optimization. In Whitton, M. C., editor, *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1993, Anaheim, CA, USA, August 2-6, 1993*, pages 19–26. ACM.

Hu, R., Li, W., Kaick, O. V., Huang, H., Averkiou, M., Cohen-Or, D., and Zhang, H. (2017). Co-locating style-defining elements on 3d shapes. *ACM Transactions on Graphics (TOG)*, 36(3):33.

Hu, Y., Schneider, T., Wang, B., Zorin, D., and Panozzo, D. (2020). Fast tetrahedral meshing in the wild. *ACM Trans. Graph.*, 39(4):117.

Hu, Y., Zhou, Q., Gao, X., Jacobson, A., Zorin, D., and Panozzo, D. (2018). Tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)*, 37(4):60.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269.

Huang, J., Jiang, T., Shi, Z., Tong, Y., Bao, H., and Desbrun, M. (2014). l1-based construction of polycube maps from complex shapes. *ACM Trans. Graph.*, 33(3).

Huang, Q., Adams, B., Wicke, M., and Guibas, L. J. (2008). Non-rigid registration under isometric deformations. *Comput. Graph. Forum*, 27(5):1449–1457.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Igarashi, T., Moscovich, T., and Hughes, J. F. (2005a). As-rigid-as-possible shape manipulation. In *ACM transactions on Graphics (TOG)*, volume 24, pages 1134–1141. ACM.

Igarashi, T., Moscovich, T., and Hughes, J. F. (2005b). As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141.

J., K., Smith, E., Lafleche, J.-F., Fuji Tsang, C., Rozantsev, A., Chen, W., Xiang, T., Lebaredian, R., and Fidler, S. (2019). Kaolin: A pytorch library for accelerating 3d deep learning research. *arXiv:1911.05063*.

Jacobson, A., Panozzo, D., et al. (2018). libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

Jacobson, A., Tosun, E., Sorkine, O., and Zorin, D. (2010). Mixed finite elements for variational surface modeling. *Comput. Graph. Forum*, 29(5):1565–1574.

Jakob, W., Tarini, M., Panozzo, D., and Sorkine-Hornung, O. (2015). Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6):189:1–189:15.

James, S. and Johns, E. (2016). 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*.

Jeon, I., Choi, K., Kim, T., Choi, B., and Ko, H. (2013). Constrainable multigrid for cloth. *Comput. Graph. Forum*, 32(7):31–39.

Jiang, Z., Schneider, T., Zorin, D., and Panozzo, D. (2020). Bijective projection in a shell. *ACM Trans. Graph.*, 39(6):247:1–247:18.

Jin, M., Kim, J., Luo, F., and Gu, X. (2008). Discrete surface ricci flow. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1030–1043.

Johnson-Roberson, M., Barto, C., Mehta, R., Sridhar, S. N., Rosaen, K., and Vasudevan, R. (2017). Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 746–753. IEEE.

Kahl, K. and Rottmann, M. (2018). Least angle regression coarsening in bootstrap algebraic multigrid. *arXiv preprint arXiv:1802.00595*.

Kajiya, J. T. (1986). The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM.

Kanbak, C., Moosavi Dezfooli, S. M., and Frossard, P. (2018). Geometric robustness of deep networks: analysis and improvement. *Proceedings of IEEE CVPR*.

Karciauskas, K. and Peters, J. (2018). A new class of guided C2 subdivision surfaces combining good shape with nested refinement. *Comput. Graph. Forum*, 37(6):84–95.

Kato, H., Ushiku, Y., and Harada, T. (2018). Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916.

Katrutsa, A., Daulbaev, T., and Oseledets, I. V. (2020). Black-box learning of multigrid parameters. *J. Comput. Appl. Math.*, 368.

Kavan, L., Gerszewski, D., Bargteil, A. W., and Sloan, P.-P. (2011). Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph.*, 30(4).

Kazhdan, M. and Hoppe, H. (2019). An adaptive multi-grid solver for applications in computer graphics. In *Computer Graphics Forum*, volume 38, pages 138–150. Wiley Online Library.

Kazhdan, M., Solomon, J., and Ben-Chen, M. (2012). Can mean-curvature flow be modified to be non-singular? *Comput. Graph. Forum*, 31(5):1745–1754.

Kazhdan, M. M. and Hoppe, H. (2008). Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph.*, 27(3):21.

Kazhdan, M. M. and Hoppe, H. (2013). Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13.

Keraütret, B., Granier, X., and Braquelaire, A. (2005). Intuitive shape modeling by shading design. In *International Symposium on Smart Graphics*, pages 163–174. Springer.

Kerber, J., Tevs, A., Belyaev, A., Zayer, R., and Seidel, H.-P. (2009). Feature sensitive bas relief generation. In *2009 IEEE International Conference on Shape Modeling and Applications*, pages 148–154. IEEE.

Kharevych, L., Mullen, P., Owhadi, H., and Desbrun, M. (2009). Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. on Graph.*

Khodakovsky, A., Litke, N., and Schröder, P. (2003). Globally smooth parameterizations with low distortion. *ACM Trans. Graph.*, 22(3):350–357.

Kim, V. G., Lipman, Y., and Funkhouser, T. (2011). Blended intrinsic maps. In *ACM Transactions on Graphics (TOG)*, volume 30, page 79. ACM.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Knöppel, F., Crane, K., Pinkall, U., and Schröder, P. (2015). Stripe patterns on surfaces. *ACM Trans. Graph.*

Kobbelt, L. (1996). Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Comput. Graph. Forum*, 15(3):409–420.

Kobbelt, L. (2000). 3-subdivision. In Brown, J. R. and Akeley, K., editors, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2000, New Orleans, LA, USA, July 23-28, 2000*, pages 103–112. ACM.

Kobbelt, L. and Botsch, M. (2004). A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814.

Kobbelt, L., Campagna, S., and Seidel, H. (1998). A general framework for mesh decimation. In Davis, W. A., Booth, K. S., and Fournier, A., editors, *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, pages 43–50. Canadian Human-Computer Communications Society.

Kostrikov, I., Jiang, Z., Panozzo, D., Zorin, D., and Bruna, J. (2018). Surface networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2540–2548. IEEE Computer Society.

Kovalsky, S. Z., Galun, M., and Lipman, Y. (2016). Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.*, 35(4):134:1–134:11.

Kraevoy, V. and Sheffer, A. (2004). Cross-parameterization and compatible remeshing of 3d models. *ACM Transactions on Graphics (TOG)*, 23(3):861–869.

Kratt, J., Eisenkeil, F., Pirk, S., Sharf, A., and Deussen, O. (2014). Non-realistic 3d object stylization. In *Proceedings of the Workshop on Computational Aesthetics*, CAe '14, pages 67–75, New York, NY, USA. ACM.

Krishnan, D., Fattal, R., and Szeliski, R. (2013). Efficient preconditioning of laplacian matrices for computer graphics. *ACM Trans. Graph.*, 32(4):142:1–142:15.

Krishnan, D. and Szeliski, R. (2011). Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph.*, 30(6):177.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial machine learning at scale.

Kyng, R. and Sachdeva, S. (2016). Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 573–582. IEEE.

Kyprianidis, J. E., Collomosse, J., Wang, T., and Isenberg, T. (2013). State of the "art": A taxonomy of artistic stylization techniques for images and video. *IEEE transactions on visualization and computer graphics*, 19(5):866–885.

Lai, J., Chen, Y., Gu, Y., Batty, C., and Wan, J. W. L. (2020). Fast and scalable solvers for the fluid pressure equations with separating solid boundary conditions. *Comput. Graph. Forum*, 39(2):23–33.

Lai, Y.-K., Hu, S.-M., Gu, D., and Martin, R. R. (2005). Geometric texture synthesis and transfer via geometry images. In *Proc. SPM*.

Landreneau, E. and Schaefer, S. (2010). Scales and scale-like structures. *Comput. Graph. Forum*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L. C., and Dobkin, D. P. (1998). MAPS: multiresolution adaptive parameterization of surfaces. In Cunningham, S., Bransford, W., and Cohen, M. F., editors, *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, pages 95–104. ACM.

Levi, Z. and Gotsman, C. (2015). Smooth rotation enhanced as-rigid-as-possible mesh animation. *IEEE Trans. Vis. Comput. Graph.*, 21(2):264–277.

Lévy, B., Petitjean, S., Ray, N., and Maillot, J. (2002). Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371.

Li, D., Fei, Y., and Zheng, C. (2015). Interactive acoustic transfer approximation for modal sound. *ACM Trans. Graph.*, 35(1).

Li, H., Zhang, H., Wang, Y., Cao, J., Shamir, A., and Cohen-Or, D. (2013). Curve style analysis in a set of shapes. In *Computer Graphics Forum*, volume 32, pages 77–88. Wiley Online Library.

Li, M. and Zhang, H. (2021). D^2IM-Net: Learning detail disentangled implicit fields from single images. In *Proc. of CVPR*.

Li, R., Li, X., Fu, C., Cohen-Or, D., and Heng, P. (2019). PU-GAN: A point cloud upsampling adversarial network. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 7202–7211. IEEE.

Lim, I., Gehre, A., and Kobbelt, L. (2016). Identifying style of 3d shapes using deep metric learning. In *Computer Graphics Forum*, volume 35, pages 207–215. Wiley Online Library.

Lin, J., Jin, X., Fan, Z., and Wang, C. C. (2008). Automatic polycube-maps. In *International Conference on Geometric Modeling and Processing*, pages 3–16. Springer.

Lindstrom, P. and Turk, G. (2000). Image-driven simplification. *ACM Transactions on Graphics (ToG)*, 19(3):204–241.

Litany, O., Remez, T., Rodolà, E., Bronstein, A. M., and Bronstein, M. M. (2017). Deep functional maps: Structured prediction for dense shape correspondence. In *ICCV*, pages 5660–5668.

Liu, B. and Todd, J. T. (2004). Perceptual biases in the interpretation of 3d shape from shading. *Vision research*.

Liu, G., Ceylan, D., Yumer, E., Yang, J., and Lien, J.-M. (2017a). Material editing using a physically based rendering network. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2280–2288. IEEE.

Liu, H. D. and Jacobson, A. (2019a). Cubic stylization. *ACM Trans. Graph.*, 38(6):197:1–197:10.

Liu, H. D., Kim, V. G., Chaudhuri, S., Aigerman, N., and Jacobson, A. (2020). Neural subdivision. *ACM Trans. Graph.*, 39(4):124.

Liu, H.-T. D. and Jacobson, A. (2019b). Cubic stylization. *ACM Transactions on Graphics*.

Liu, H.-T. D., Tao, M., and Jacobson, A. (2018). Paparazzi: Surface editing by way of multi-view image processing. In *SIGGRAPH Asia 2018 Technical Papers*, page 221. ACM.

Liu, L., Zhang, L., Xu, Y., Gotsman, C., and Gortler, S. J. (2008). A local/global approach to mesh parameterization. *Comput. Graph. Forum*, 27(5):1495–1504.

Liu, S., Ferguson, Z., Jacobson, A., and Gingold, Y. (2017b). Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Transactions on Graphics (TOG)*, 36(6):216:1–216:15.

Liu, S., Ferguson, Z., Jacobson, A., and Gingold, Y. I. (2017c). Seamless: seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.*, 36(6):216:1–216:15.

Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L. (2013a). Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):214.

Liu, T., Bargteil, A. W., O'Brien, J. F., and Kavan, L. (2013b). Fast simulation of mass-spring systems. *ACM Trans. Graph.*, 32(6):214:1–214:7.

Liu, T., Bouaziz, S., and Kavan, L. (2017d). Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.*, 36(3):23:1–23:16.

Liu, T., Hertzmann, A., Li, W., and Funkhouser, T. (2015). Style compatibility for 3d furniture models. *ACM Transactions on Graphics (TOG)*, 34(4):85.

Liu, Y., Pottmann, H., Wallner, J., Yang, Y.-L., and Wang, W. (2006). Geometric modeling with conical meshes and developable surfaces. In *ACM transactions on graphics (TOG)*, volume 25, pages 681–689. ACM.

Livesu, M., Vining, N., Sheffer, A., Gregson, J., and Scateni, R. (2013). Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics (TOG)*, 32(6):171.

Livne, O. E. and Brandt, A. (2012). Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522.

Loop, C. (1987). Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics*.

Loper, M. M. and Black, M. J. (2014). OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer.

Luebke, D. and Erikson, C. (1997). View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 199–208. ACM Press/Addison-Wesley Publishing Co.

Luebke, D. and Hallen, B. (2001). Perceptually driven simplification for interactive rendering. In *Rendering Techniques 2001*, pages 223–234. Springer.

Lun, Z., Kalogerakis, E., and Sheffer, A. (2015). Elements of style: learning perceptual shape style similarity. *ACM Transactions on Graphics (TOG)*, 34(4):84.

Lun, Z., Kalogerakis, E., Wang, R., and Sheffer, A. (2016). Functionality preserving shape style transfer. *ACM Transactions on Graphics (TOG)*, 35(6):209.

Luo, S.-J., Yue, Y., Huang, C.-K., Chung, Y.-H., Imai, S., Nishita, T., and Chen, B.-Y. (2015). Legolization: optimizing lego designs. *ACM Transactions on Graphics (TOG)*, 34(6):222.

Ma, C., Huang, H., Sheffer, A., Kalogerakis, E., and Wang, R. (2014). Analogy-driven 3d style transfer. *Comput. Graph. Forum*, 33(2):175–184.

MacNeal, R. H. (1949). *The solution of partial differential equations by means of electrical networks*. PhD thesis, California Institute of Technology.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*.

Manson, J. and Schaefer, S. (2011). Hierarchical deformation of locally rigid meshes. In *Computer Graphics Forum*, volume 30, pages 2387–2396. Wiley Online Library.

Manteuffel, T. A., Olson, L. N., Schroder, J. B., and Southworth, B. S. (2017). A root-node–based algebraic multigrid method. *SIAM Journal on Scientific Computing*, 39(5):S723–S756.

Maron, H., Galun, M., Aigerman, N., Trope, M., Dym, N., Yumer, E., Kim, V. G., and Lipman, Y. (2017). Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph.*, 36(4):71:1–71:10.

Marschner, S. R. and Greenberg, D. P. (1997). Inverse lighting for photography. In *Color and Imaging Conference*.

Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on riemannian manifolds. In *2015 IEEE International Conference on Computer Vision Workshop, ICCV Workshops 2015, Santiago, Chile, December 7-13, 2015*, pages 832–840. IEEE Computer Society.

Mattingley, J. and Boyd, S. (2012). Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27.

McAdams, A., Sifakis, E., and Teran, J. (2010). A parallel multigrid poisson solver for fluids simulation on large grids. In Popovic, Z. and Otaduy, M. A., editors, *Proceedings of the 2010 Eurographics/ACM SIGGRAPH Symposium on Computer Animation, SCA 2010, Madrid, Spain, 2010*, pages 65–73. Eurographics Association.

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., and Sifakis, E. (2011). Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.*, 30(4):37.

Mehra, R., Zhou, Q., Long, J., Sheffer, A., Gooch, A., and Mitra, N. J. (2009). Abstraction of man-made shapes. In *ACM transactions on graphics (TOG)*, volume 28, page 137. ACM.

Miller, G. (1994). Efficient algorithms for local and global accessibility shading. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 319–326, New York, NY, USA. ACM.

Miyato, T., Maeda, S.-i., Ishii, S., and Koyama, M. (2018). Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*.

Moosavi-Dezfooli, S., Fawzi, A., Fawzi, O., and Frossard, P. (2017). Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 86–94.

Moosavi Dezfooli, S. M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, number EPFL-CONF-218057.

Movshovitz-Attias, Y., Kanade, T., and Sheikh, Y. (2016). How useful is photo-realistic rendering for visual learning? In *European Conference on Computer Vision*, pages 202–217. Springer.

Mullen, P., Tong, Y., Alliez, P., and Desbrun, M. (2008). Spectral conformal parameterization. *Comput. Graph. Forum*, 27(5):1487–1494.

Muntoni, A., Livesu, M., Scateni, R., Sheffer, A., and Panozzo, D. (2018). Axis-aligned height-field block decomposition of 3d shapes. *ACM Transactions on Graphics (TOG)*, 37(5):169.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International*

Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel, pages 807–814. Omnipress.

Nasikun, A., Brandt, C., and Hildebrandt, K. (2018). Fast approximation of laplace-beltrami eigenproblems. In *Computer Graphics Forum*, volume 37, pages 121–134. Wiley Online Library.

Ni, X., Garland, M., and Hart, J. C. (2004). Fair morse functions for extracting the topological structure of a surface mesh. *ACM Trans. Graph.*, 23(3):613–622.

Nogneng, D. and Ovsjanikov, M. (2017). Informative descriptor preservation via commutativity for shape matching. In *Computer Graphics Forum*, volume 36, pages 259–267. Wiley Online Library.

Oh, S., Noh, J., and Wohn, K. (2008). A physically faithful multigrid method for fast cloth simulation. *Comput. Animat. Virtual Worlds*, 19(3-4):479–492.

Olson, L. N., Schroder, J., and Tuminaro, R. S. (2010). A new perspective on strength measures in algebraic multigrid. *Numerical Linear Algebra with Applications*, 17(4):713–733.

Olson, L. N. and Schroder, J. B. (2018). PyAMG: Algebraic multigrid solvers in Python v4.0. Release 4.0.

Or-El, R., Hershkovitz, R., Wetzler, A., Rosman, G., Bruckstein, A. M., and Kimmel, R. (2016). Real-time depth refinement for specular objects. In *Proc. CVPR*.

Otaduy, M. A., Germann, D., Redon, S., and Gross, M. H. (2007). Adaptive deformations with fast tight bounds. In Gleicher, M. and Thalmann, D., editors, *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2007, San Diego, California, USA, August 2-4, 2007*, pages 181–190. Eurographics Association.

Ovsjanikov, M., Ben-Chen, M., Solomon, J., Butscher, A., and Guibas, L. (2012). Functional maps: a flexible representation of maps between shapes. *ACM Transactions on Graphics (TOG)*, 31(4):30.

Öztireli, A. C., Alexa, M., and Gross, M. (2010). Spectral sampling of manifolds. In *ACM Transactions on Graphics (TOG)*, volume 29, page 168. ACM.

Ozturk, C., Hancer, E., and Karaboga, D. (2014). Color image quantization: a short review and an application with artificial bee colony algorithm. *Informatica*, 25(3):485–503.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8024–8035.

Peng, Y., Deng, B., Zhang, J., Geng, F., Qin, W., and Liu, L. (2018). Anderson acceleration for geometry optimization and physics simulation. *ACM Trans. Graph.*, 37(4):42:1–42:14.

Peyré, G. and Mallat, S. (2005). Surface compression with geometric bandelets. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 601–608. ACM.

Pharr, M., Jakob, W., and Humphreys, G. (2016). *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

Phong, B. T. (1975). Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317.

Pinkall, U. and Polthier, K. (1993). Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36.

Pixologic Inc. (2020). Zbrush.

Popović, J. and Hoppe, H. (1997). Progressive simplicial complexes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 217–224. ACM Press/Addison-Wesley Publishing Co.

Poranne, R., Ovreiu, E., and Gotsman, C. (2013). Interactive planarization and optimization of 3d meshes. *Comput. Graph. Forum*, 32(1):152–163.

Poulenard, A. and Ovsjanikov, M. (2018). Multi-directional geodesic neural networks via equivariant convolution. *ACM Trans. Graph.*, 37(6):236:1–236:14.

Prada, F. and Kazhdan, M. (2015). Unconditionally stable shock filters for image and geometry processing. *Comput. Graph. Forum*, 34(5):201–210.

Prados, E. and Faugeras, O. (2006). Shape from shading. *Handbook of mathematical models in computer vision*, pages 375–388.

Praun, E., Sweldens, W., and Schröder, P. (2001). Consistent mesh parameterizations. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 179–184, New York, NY, USA. Association for Computing Machinery.

Preetham, A. J., Shirley, P., and Smits, B. (1999). A practical analytic model for daylight. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM Press/Addison-Wesley Publishing Co.

Preiner, R., Boubekeur, T., and Wimmer, M. (2019). Gaussian-product subdivision surfaces. *ACM Trans. Graph.*, 38(4):35:1–35:11.

Qiu, Y. (2018). spectra: C++ library for large scale eigenvalue problems. https://github.com/yixuan/spectra/.

Rabinovich, M., Hoffmann, T., and Sorkine-Hornung, O. (2018). The shape space of discrete orthogonal geodesic nets. *ACM Trans. Graph.*, 37(6):228:1–228:17.

Rabinovich, M., Poranne, R., Panozzo, D., and Sorkine-Hornung, O. (2017). Scalable locally injective mappings. *ACM Transactions on Graphics (TOG)*, 36(2):16.

Ramamoorthi, R. and Hanrahan, P. (2001a). An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500. ACM.

Ramamoorthi, R. and Hanrahan, P. (2001b). A signal-processing framework for inverse rendering. In *Proc. SIGGRAPH*.

Ranjan, A., Bolkart, T., Sanyal, S., and Black, M. J. (2018). Generating 3d faces using convolutional mesh autoencoders. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part III*, volume 11207 of *Lecture Notes in Computer Science*, pages 725–741. Springer.

Ray, N. and Lévy, B. (2003). Hierarchical least squares conformal map. In *11th Pacific Conference on Computer Graphics and Applications, PG 2003, Canmore, Canada, October 8-10, 2003*, pages 263–270. IEEE Computer Society.

Reinert, B., Ritschel, T., and Seidel, H. (2012). Homunculus warping: Conveying importance using self-intersection-free non-homogeneous mesh deformation. *Comput. Graph. Forum*, 31(7-2):2165–2171.

Robertini, N., Casas, D., Aguiar, E., and Theobalt, C. (2017). Multi-view performance capture of surface details. *Int. J. Comput. Vision*.

Rozsa, A., Rudd, E. M., and Boult, T. E. (2016). Adversarial diversity and hard positive generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 25–32.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Ruge, J. W. and Stüben, K. (1987). Algebraic multigrid. In *Multigrid methods*, pages 73–130. SIAM.

Rustamov, R. M., Ovsjanikov, M., Azencot, O., Ben-Chen, M., Chazal, F., and Guibas, L. (2013). Map-based exploration of intrinsic shape differences and variability. *ACM Transactions on Graphics (TOG)*, 32(4):72.

Sabin, M. and Dodgson, N. (2004). A circle-preserving variant of the four-point subdivision scheme. *Mathematical Methods for Curves and Surfaces: Tromsø 2004*.

Sacht, L., Vouga, E., and Jacobson, A. (2015). Nested cages. *ACM Trans. Graph.*, 34(6):170:1–170:14.

Sadeghi, F. and Levine, S. (2016). Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*.

Schaefer, S., Vouga, E., and Goldman, R. (2008). Nonlinear subdivision through nonlinear averaging. *Comput. Aided Geom. Des.*, 25(3):162–180.

Schmidt, R. and Singh, K. (2010). Meshmixer: An interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, SIGGRAPH '10, New York, NY, USA. Association for Computing Machinery.

Schönemann, P. H. and Carroll, R. M. (1970). Fitting one matrix to another under choice of a central dilation and a rigid motion. *Psychometrika*, 35(2):245–255.

Schreiner, J., Asirvatham, A., Praun, E., and Hoppe, H. (2004). Inter-surface mapping. *ACM Trans. Graph.*, 23(3):870–877.

Schroder, P. (1996). Wavelets in computer graphics. *Proceedings of the IEEE*, 84(4):615–625.

Schroeder, W. J., Zarge, J. A., and Lorensen, W. E. (1992). Decimation of triangle meshes. In *ACM siggraph computer graphics*, volume 26, pages 65–70. ACM.

Schüller, C., Panozzo, D., and Sorkine-Hornung, O. (2014). Appearance-mimicking surfaces. *ACM Transactions on Graphics (TOG)*, 33(6):216.

Sellán, S., Aigerman, N., and Jacobson, A. (2020). Developability of heightfields via rank minimization. *ACM Trans. Graph.*, 39(4):109.

Setaluri, R., Aanjaneya, M., Bauer, S., and Sifakis, E. (2014). Spgrid: a sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph.*, 33(6):205:1–205:12.

Sharf, A., Lewiner, T., Shklarski, G., Toledo, S., and Cohen-Or, D. (2007). Interactive topology-aware surface reconstruction. *ACM Trans. on Graph.*

Sharp, N., Soliman, Y., and Crane, K. (2019). Navigating intrinsic triangulations. *ACM Trans. Graph.*, 38(4):55:1–55:16.

Shen, L., Luo, S., Huang, C., and Chen, B. (2012). SD models: Super-deformed character models. *Comput. Graph. Forum*, 31(7-1):2067–2075.

Shi, L., Yu, Y., Bell, N., and Feng, W. (2006). A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph.*, 25(3):1108–1117.

Shi, X., Bao, H., and Zhou, K. (2009). Out-of-core multigrid solver for streaming meshes. *ACM Trans. Graph.*, 28(5):173.

Shreiner, D. and Group, T. K. O. A. W. (2009). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Addison-Wesley Professional, 7th edition.

Shtengel, A., Poranne, R., Sorkine-Hornung, O., Kovalsky, S. Z., and Lipman, Y. (2017). Geometric optimization via composite majorization. *ACM Trans. Graph.*, 36(4):38–1.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sitzmann, V., Thies, J., Heide, F., Nießner, M., Wetzstein, G., and Zollhöfer, M. (2019). Deepvoxels: Learning persistent 3d feature embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2437–2446. Computer Vision Foundation / IEEE.

Skouras, M., Thomaszewski, B., Bickel, B., and Gross, M. H. (2012). Computational design of rubber balloons. *Comput. Graph. Forum*, 31(2):835–844.

Sloan, P. J., Martin, W., Gooch, A., and Gooch, B. (2001). The lit sphere: A model for capturing NPR shading from art. In *Proceedings of the Graphics Interface 2001 Conference, Ottawa, Ontario, Canada, June 7-9, 2001*, pages 143–150. Canadian Human-Computer Communications Society.

Sloan, P.-P., Luna, B., and Snyder, J. (2005). Local, deformable precomputed radiance transfer. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1216–1224. ACM.

Song, W., Belyaev, A., and Seidel, H.-P. (2007). Automatic generation of bas-reliefs from 3d shapes. In *IEEE International Conference on Shape Modeling and Applications 2007 (SMI'07)*, pages 211–214. IEEE.

Sorkine, O. (2005). Laplacian mesh processing. In Chrysanthou, Y. and Magnor, M. A., editors, *Eurographics 2005 - State of the Art Reports, Dublin, Ireland, August 29 - September 2, 2005*, pages 53–70. Eurographics Association.

Sorkine, O. and Alexa, M. (2007). As-rigid-as-possible surface modeling. In Belyaev, A. G. and Garland, M., editors, *Proceedings of the Fifth Eurographics Symposium on Geometry Processing, Barcelona, Spain, July 4-6, 2007*, volume 257 of *ACM International Conference Proceeding Series*, pages 109–116. Eurographics Association.

Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., and Seidel, H. (2004). Laplacian surface editing. In Boissonnat, J. and Alliez, P., editors, *Second Eurographics Symposium on Geometry Processing, Nice, France, July 8-10, 2004*, volume 71 of *ACM International Conference Proceeding Series*, pages 175–184. Eurographics Association.

Stam, J. (1998). Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In Cunningham, S., Bransford, W., and Cohen, M. F., editors, *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, Orlando, FL, USA, July 19-24, 1998*, pages 395–404. ACM.

Stein, O., Grinspun, E., and Crane, K. (2018a). Developability of triangle meshes. *ACM Trans. Graph.*, 37(4):77:1–77:14.

Stein, O., Grinspun, E., Wardetzky, M., and Jacobson, A. (2018b). Natural boundary conditions for smoothing in geometry processing. *ACM Trans. Graph.*, 37(2):23:1–23:13.

Stein, O., Jacobson, A., and Grinspun, E. (2019). Interactive design of castable shapes using two-piece rigid molds. *Computers & Graphics*.

Stein, O., Jacobson, A., Wardetzky, M., and Grinspun, E. (2020). A smoothness energy without boundary distortion for curved surfaces. *ACM Trans. Graph.*, 39(3):18:1–18:17.

Struyf, A., Hubert, M., and Rousseeuw, P. (1997). Clustering in an object-oriented environment. *Journal of Statistical Software*.

Stuben, K. (2000). Algebraic multigrid (amg): an introduction with applications. *Multigrid*.

Su, H., Qi, C. R., Li, Y., and Guibas, L. J. (2015). Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3d model views. pages 2686–2694.

Su, J., Vargas, D. V., and Kouichi, S. (2017). One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*.

Sun, S., Yeh, C.-F., Ostendorf, M., Hwang, M.-Y., and Xie, L. (2018). Training augmentation with adversarial examples for robust speech recognition. *arXiv preprint arXiv:1806.02782*.

Sun, Y. and Vandenberghe, L. (2015). Decomposition methods for sparse matrix nearness problems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1691–1717.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks.

Takayama, K., Schmidt, R. M., Singh, K., Igarashi, T., Boubekeur, T., and Sorkine, O. (2011). Geobrush: Interactive mesh geometry cloning. *Comput. Graph. Forum*, 30(2):613–622.

Tamstorf, R., Jones, T., and McCormick, S. F. (2015). Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph.*, 34(6):245:1–245:13.

Tan, Q., Gao, L., Lai, Y., and Xia, S. (2018). Variational autoencoders for deforming 3d mesh models. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 5841–5850. IEEE Computer Society.

Tang, C., Sun, X., Gomes, A., Wallner, J., and Pottmann, H. (2014). Form-finding with polyhedral meshes made simple. *ACM Trans. Graph.*, 33(4):70:1–70:9.

Tarini, M., Hormann, K., Cignoni, P., and Montani, C. (2004). Polycube-maps. *ACM Trans. Graph.*, 23(3):853–860.

Tatarchenko, M., Richter, S. R., Ranftl, R., Li, Z., Koltun, V., and Brox, T. (2019). What do single-view 3d reconstruction networks learn? In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 3405–3414. Computer Vision Foundation / IEEE.

Terzopoulos, D., Platt, J. C., Barr, A. H., and Fleischer, K. W. (1987). Elastically deformable models. In Stone, M. C., editor, *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987*, pages 205–214. ACM.

Testuz, R., Schwartzburg, Y., and Pauly, M. (2013). Automatic Generation of Constructable Brick Sculptures. In *Eurographics 2013 - Short Papers*. The Eurographics Association.

Theobalt, C., Roessl, C., Aguiar, E. d., and Seidel, H.-P. (2007). Animation Collage. In Metaxas, D. and Popovic, J., editors, *Eurographics/SIGGRAPH Symposium on Computer Animation*. The Eurographics Association.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Tikhonov, A. N., Goncharsky, A., Stepanov, V., and Yagola, A. G. (2013). *Numerical methods for the solution of ill-posed problems*, volume 328. Springer Science & Business Media.

Tobler, R. F., Maierhofer, S., and Wilkie, A. (2002a). Mesh-based parametrized l-systems and generalized subdivision for generating complex geometry. *International Journal of Shape Modeling*, 8(2):173–191.

Tobler, R. F., Maierhofer, S., and Wilkie, A. (2002b). A multiresolution mesh generation approach for procedural definition of complex geometry. In *2002 International Conference* *on Shape Modeling and Applications (SMI 2002), 17-22 May 2002, Banff, Alberta, Canada*, pages 35–42. IEEE Computer Society.

Tosun, E., Gingold, Y. I., Reisman, J., and Zorin, D. (2007). Shape optimization using reflection lines. In *Proc. SGP*.

Tramèr, F., Kurakin, A., Papernot, N., Boneh, D., and McDaniel, P. (2017). Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*.

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization.

Trettner, P. and Kobbelt, L. (2020). Fast and robust QEF minimization using probabilistic quadrics. *Comput. Graph. Forum*, 39(2):325–334.

Trottenberg, U., Oosterlee, C. W., and Schuller, A. (2000). *Multigrid*. Elsevier.

Turk, G. (1991). Generating textures on arbitrary surfaces using reaction-diffusion. *Siggraph*.

Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in python. *PeerJ*, 2:e453.

Van Kaick, O., Zhang, H., Hamarneh, G., and Cohen-Or, D. (2011). A survey on shape correspondence. In *Computer Graphics Forum*, volume 30, pages 1681–1707. Wiley Online Library.

Van Overveld, C. (1996). Painting gradients: Free-form surface design using shading patterns. In *Graphics Interface*, volume 96, pages 151–158.

Vandenberghe, L. and Andersen, M. S. (2015). Chordal graphs and semidefinite optimization. *Found. Trends Optim.*, 1(4).

Vanek, P., Mandel, J., and Brezina, M. (1996). Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196.

Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M. J., Laptev, I., and Schmid, C. (2017). Learning from synthetic humans. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*.

Vaxman, A., Müller, C., and Weber, O. (2015). Conformal mesh deformations with möbius transformations. *ACM Trans. Graph.*, 34(4):55:1–55:11.

Vaxman, A., Müller, C., and Weber, O. (2018). Canonical möbius subdivision. *ACM Trans. Graph.*, 37(6):227:1–227:15.

Veeravasarapu, V., Rothkopf, C., and Visvanathan, R. (2017a). Model-driven simulations for computer vision. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 1063–1071. IEEE.

Veeravasarapu, V., Rothkopf, C. A., and Ramesh, V. (2017b). Adversarially tuned scene generation. In *CVPR*, pages 6441–6449.

Velho, L., Perlin, K., Biermann, H., and Ying, L. (2002). Algorithmic shape modeling with subdivision surfaces. *Comput. Graph.*, 26(6):865–875.

Vestner, M., Lähner, Z., Boyarski, A., Litany, O., Slossberg, R., Remez, T., Rodolà, E., Bronstein, A. M., Bronstein, M. M., Kimmel, R., and Cremers, D. (2017a). Efficient deformable shape correspondence via kernel matching. In *3DV*.

Vestner, M., Litman, R., Rodolà, E., Bronstein, A., and Cremers, D. (2017b). Product manifold filter: Non-rigid shape correspondence via kernel density estimation in the product space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3327–3336.

Wang, H., He, Y., Li, X., Gu, X., and Qin, H. (2007). Polycube splines. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 241–251. ACM.

Wang, H., Jin, M., He, Y., Gu, X., and Qin, H. (2008). User-controllable polycube map for manifold spline construction. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 397–404. ACM.

Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., and Jiang, Y. (2018a). Pixel2mesh: Generating 3d mesh models from single RGB images. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*, volume 11215 of *Lecture Notes in Computer Science*, pages 55–71. Springer.

Wang, Y., Aigerman, N., Kim, V. G., Chaudhuri, S., and Sorkine-Hornung, O. (2020). Neural cages for detail-preserving 3d deformations. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 72–80. IEEE.

Wang, Y., Kim, V. G., Bronstein, M., and Solomon, J. (2019a). Learning geometric operators on meshes. *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Wang, Y., Wu, S., Huang, H., Cohen-Or, D., and Sorkine-Hornung, O. (2019b). Patch-based progressive 3d point set upsampling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 5958–5967. Computer Vision Foundation / IEEE.

Wang, Y.-S., Chao, M.-W., Yi, C.-C., and Lin, C.-H. (2011). Cubist style rendering for 3d polygonal models. *Journal of Information Science and Engineering*, 27(6):1885–1899.

Wang, Z., Wu, L., Fratarcangeli, M., Tang, M., and Wang, H. (2018b). Parallel multigrid for nonlinear cloth simulation. *Comput. Graph. Forum*, 37(7):131–141.

Wei, L., Lefebvre, S., Kwatra, V., and Turk, G. (2009). State of the art in example-based texture synthesis. In Pauly, M. and Greiner, G., editors, *30th Annual Conference of the European Association for Computer Graphics, Eurographics 2009 - State of the Art Reports, Munich, Germany, March 30 - April 3, 2009*, pages 93–117. Eurographics Association.

Wei, L.-Y. and Levoy, M. (2001). Texture synthesis over arbitrary manifold surfaces. *Siggraph*.

Weier, M., Stengel, M., Roth, T., Didyk, P., Eisemann, E., Eisemann, M., Grogorick, S., Hinkenjann, A., Kruijff, E., Magnor, M., et al. (2017). Perception-driven accelerated rendering. In *Computer Graphics Forum*, volume 36, pages 611–643. Wiley Online Library.

Welsh, D. J. A. and Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *Comput. J.*, 10(1):85–86.

Wen, C., Zhang, Y., Li, Z., and Fu, Y. (2019). Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1042–1051. IEEE.

Weyrich, T., Deng, J., Barnes, C., Rusinkiewicz, S., and Finkelstein, A. (2007). Digital bas-relief from 3d scenes. In *ACM transactions on graphics (TOG)*, volume 26, page 32. ACM.

Williams, L. (1978). Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pages 270–274. ACM.

Williams, N., Luebke, D., Cohen, J. D., Kelley, M., and Schubert, B. (2003). Perceptually guided simplification of lit, textured meshes. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 113–121. ACM.

Woodham, R. J. (1980). Photometric method for determining surface orientation from multiple images. *Optical engineering*, 19(1):191139.

Wu, C., Varanasi, K., Liu, Y., Seidel, H.-P., and Theobalt, C. (2011). Shading-based dynamic shape refinement from multi-view video under general illumination. In *Proc. ICCV*.

Wu, C., Zollhöfer, M., Nießner, M., Stamminger, M., Izadi, S., and Theobalt, C. (2014). Real-time shading-based refinement for consumer depth cameras. *ACM Transactions on Graphics (TOG)*, 33(6):200.

Wu, J., Tenenbaum, J. B., and Kohli, P. (2017). Neural scene de-rendering. In *Proc. CVPR*, volume 2.

Xia, J. C. and Varshney, A. (1996). Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th conference on Visualization'96*, pages 327–ff. IEEE Computer Society Press.

Xian, Z., Tong, X., and Liu, T. (2019). A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Transactions on Graphics (TOG)*, 38(6):1–13.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xu, J. and Zikatanov, L. (2017). Algebraic multigrid methods. *Acta Numerica*, 26:591–721.

Xu, K., Li, H., Zhang, H., Cohen-Or, D., Xiong, Y., and Cheng, Z.-Q. (2010). Style-content separation by anisotropic part scales. In *ACM Transactions on Graphics (TOG)*, volume 29, page 184. ACM.

Xu, L., Lu, C., Xu, Y., and Jia, J. (2011). Image smoothing via l 0 gradient minimization. In *ACM Transactions on Graphics (TOG)*, volume 30, page 174. ACM.

Yifan, W., Aigerman, N., Kim, V. G., Chaudhuri, S., and Sorkine-Hornung, O. (2020). Neural cages for detail-preserving 3d deformations. In *CVPR*.

Yoshiyasu, Y., Ma, W., Yoshida, E., and Kanehiro, F. (2014). As-conformal-as-possible surface registration. *Comput. Graph. Forum*, 33(5):257–267.

Yu, F., Xu, K., Mahdavi-Amiri, A., Zhang, H., et al. (2018a). Semi-supervised co-analysis of 3d shape styles from projected lines. *ACM Transactions on Graphics (TOG)*, 37(2):21.

Yu, L., Li, X., Fu, C., Cohen-Or, D., and Heng, P. (2018b). Pu-net: Point cloud upsampling network. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2790–2799. IEEE Computer Society.

Yu, W., Zhang, K., Wan, S., and Li, X. (2014). Optimizing polycube domain construction for hexahedral remeshing. *Computer-Aided Design*, 46:58–68.

Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., Guo, B., and Shum, H. (2004). Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651.

Yumer, M. E. and Kara, L. B. (2012). Co-abstraction of shape collections. *ACM Transactions on Graphics (TOG)*, 31(6):166.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*.

Zanjani, F. G., Moin, D. A., Verheij, B., Claessen, F., Cherici, T., Tan, T., et al. (2019). Deep learning approach to semantic segmentation in 3d point cloud intra-oral scans of teeth. In *International Conference on Medical Imaging with Deep Learning*, pages 557–571. PMLR.

Zeng, X., Liu, C., Qiu, W., Xie, L., Tai, Y.-W., Tang, C. K., and Yuille, A. L. (2017). Adversarial attacks beyond the image space. *arXiv preprint arXiv:1711.07183*.

Zhang, H., Wu, C., Zhang, J., and Deng, J. (2015a). Variational mesh denoising using total variation and piecewise constant function space. *IEEE Trans. Vis. Comput. Graph.*, 21(7):873–886.

Zhang, J., Deng, B., Hong, Y., Peng, Y., Qin, W., and Liu, L. (2018). Static/dynamic filtering for mesh geometry. *IEEE transactions on visualization and computer graphics*.

Zhang, J. E., Jacobson, A., and Alexa, M. (2021). Fast updates for least-squares rotational alignment. *Computer Graphics Forum*.

Zhang, R., Tsai, P.-S., Cryer, J. E., and Shah, M. (1999). Shape-from-shading: a survey. *IEEE transactions on pattern analysis and machine intelligence*, 21(8):690–706.

Zhang, W., Deng, B., Zhang, J., Bouaziz, S., and Liu, L. (2015b). Guided mesh normal filtering. *Comput. Graph. Forum*, 34(7):23–34.

Zhao, H. and Gortler, S. J. (2016). A report on shape deformation with a stretching and bending energy. *CoRR*, abs/1603.06821.

Zhao, H., Lei, N., Li, X., Zeng, P., Xu, K., and Gu, X. (2017). Robust edge-preserved surface mesh polycube deformation. In Barbic, J., Lin, W., and Sorkine-Hornung, O., editors, *25th Pacific Conference on Computer Graphics and Applications, PG 2017 - Short Papers, Taipei, Taiwan, October 16-19, 2017*, pages 17–22. Eurographics Association.

Zhao, H., Su, K., Li, C., Zhang, B., Yang, L., Lei, N., Wang, X., Gortler, S. J., and Gu, X. (2020). Mesh parametrization driven by unit normal flow. *Comput. Graph. Forum*, 39(1):34–49.

Zhao, S. (2014). *Modeling and rendering fabrics at micron-resolution*. Cornell University.

Zheng, Y., Fantuzzi, G., Papachristodoulou, A., Goulart, P., and Wynn, A. (2017). Fast ADMM for semidefinite programs with chordal sparsity. In *American Control Conference*.

Zheng, Y., Fu, H., Au, O. K., and Tai, C. (2011). Bilateral normal filtering for mesh denoising. *IEEE Trans. Vis. Comput. Graph.*, 17(10):1521–1530.

Zhou, K., Huang, X., Wang, X., Tong, Y., Desbrun, M., Guo, B., and Shum, H. (2006). Mesh quilting for geometric texture synthesis. *ACM Trans. Graph.*, 25(3):690–697.

Zhou, Q. and Jacobson, A. (2016). Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*.

Zhu, Y., Bridson, R., and Kaufman, D. M. (2018). Blended cured quasi-newton for distortion optimization. *ACM Trans. Graph.*, 37(4):40:1–40:14.

Zhu, Y., Sifakis, E., Teran, J., and Brandt, A. (2010). An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.*, 29(2):16:1–16:18.

Zollhöfer, M., Nießner, M., Izadi, S., Rhemann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A. W., Loop, C. T., Theobalt, C., and Stamminger, M. (2014). Real-time non-rigid reconstruction using an RGB-D camera. *ACM Trans. Graph.*, 33(4):156:1–156:12.

Zorin, D. (2006). Modeling with multiresolution subdivision surfaces. In *ACM SIGGRAPH 2006 Courses*, pages 30–50. ACM.

Zorin, D. (2007). Subdivision on arbitrary meshes: algorithms and theory. In *Mathematics and Computation in Imaging Science and Information Processing*, pages 1–46. World Scientific.

Zorin, D., Schröder, P., De Rose, T., Kobbelt, L., Levin, A., and Sweldens, W. (2000). Subdivision for modeling and animation. *SIGGRAPH Course Notes*.

Zorin, D., Schröder, P., and Sweldens, W. (1996). Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 189–192, New York, NY, USA. Association for Computing Machinery.