Geometric modeling with primitives

Baptiste Angles

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Cotutelle internationale : Université Victoria

Présentée et soutenue par

Baptiste ANGLES

Le 18 avril 2019

Modélisation géométrique par primitives

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Loïc BARTHE et Brian WYVILL

Jury

M. Andrea TAGLIASACCHI, Examinateur
M. Mathias PAULIN, Examinateur
M. Alec JACOBSON, Examinateur
M. Loïc BARTHE, Co-directeur de thèse
M. Brian WYVILL, Co-directeur de thèse

GEOMETRIC MODELING WITH PRIMITIVES

by

Baptiste Angles
B.Sc., Université de Bordeaux, France, 2013
M.Sc., Université de Toulouse, France, 2015

In Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

GEOMETRIC MODELING WITH PRIMITIVES

by

Baptiste Angles
B.Sc., Université de Bordeaux, France, 2013
M.Sc., Université de Toulouse, France, 2015

Supervisory Committee

Prof. Andrea Tagliasacchi, co-supervisor
Department of Computer Science

Prof. Loïc Barthe, co-supervisor
Université de Toulouse

Prof. Brian Wyvill
Department of Computer Science

Prof. Mathias Paulin
Université de Toulouse

Abstract

Both man-made or natural objects contain repeated geometric elements that can be interpreted as primitive shapes. Plants, trees, living organisms or even crystals, showcase primitives that repeat themselves. Primitives are also commonly found in man-made environments because architects tend to reuse the same patterns over a building and typically employ simple shapes, such as rectangular windows and doors. During my PhD I studied geometric primitives from three points of view: their composition, simulation and autonomous discovery.

In the first part I present a method to reverse-engineer the function by which some primitives are combined. Our system is based on a composition function template that is represented by a parametric surface. The parametric surface is deformed via a non-rigid alignment of a surface that, once converged, represents the desired operator. This enables the interactive modeling of operators via a simple sketch, solving a major usability gap of composition modeling.

In the second part I introduce the use of a novel primitive for real-time physics simulations. This primitive is suitable to efficiently model volume-preserving deformations of rods but also of more complex structures such as muscles. One of the core advantages of our approach is that our primitive can serve as a unified representation to do collision detection, simulation, and surface skinning.

In the third part I present an unsupervised deep learning framework to learn and detect primitives. In a signal containing a repetition of elements, the method is able to automatically identify the structure of these elements (i.e. primitives) with minimal supervision. In order to train the network that contains a non-differentiable operation, a novel multi-step training process is presented.

Résumé

Les objets naturels ou artificiels contiennent des éléments géométriques répétés pouvant être interprétés comme des formes primitives. Les plantes, les arbres, les organismes vivants ou même les cristaux représentent des primitives qui se répètent. Les primitives se retrouvent aussi couramment dans les environnements créés par l'homme, car les architectes ont tendance à réutiliser les mêmes modèles sur un bâtiment et utilisent généralement des formes simples, telles que des fenêtres et des portes rectangulaires. Au cours de ma thèse, j'ai étudié les primitives géométriques sous trois angles: leur composition, leur simulation et leur découverte autonome.

Dans la première partie, je présente une méthode pour retrouver la fonction par laquelle certaines primitives sont combinées. Notre système est basé sur un modèle de fonction de composition représenté par une surface paramétrique. La surface paramétrique est déformée via un alignement non rigide d'une surface qui, une fois convergée, représente l'opérateur souhaité. Cela permet de modéliser de manière interactive les opérateurs via une simple esquisse, ce qui résout un problème majeur de facilité de modélisation de la composition.

Dans la deuxième partie, je présente l'utilisation d'une nouvelle primitive pour les simulations de physique en temps réel. Cette primitive convient pour modéliser efficacement les déformations des cordes préservant le volume, mais également celles de structures plus complexes telles que les muscles. L'un des principaux avantages de notre approche est que notre primitive peut servir de représentation unifiée pour la détection de collision, la simulation et la déformation de peau.

Dans la troisième partie, je présente un cadre d'apprentissage en profondeur non supervisé pour apprendre et détecter les primitives. Dans un signal contenant une répétition d'éléments, le procédé est capable d'identifier automatiquement la structure de ces éléments (c'est-à-dire des primitives) avec une supervision minimale. Afin de former le réseau qui contient une opération non différenciable, un nouveau processus d'apprentissage en plusieurs étapes est présenté.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my deep gratitude to my advisors Andrea Tagliasacchi and Loïc Barthe.

I am grateful to Andrea Tagliasacchi for the tremendous amount of time he has invested in advising me. I have learnt a lot by working with him and his work ethic has inspired me to give the best of myself during my PhD and forward.

I thank Loïc Barthe for believing in my potential by offering me the opportunity to pursue a PhD and setting up this jointly supervised program for me.

I am grateful to Brian Wyvill for being my advisor until his retirement. He has been very supportive of my work and has encouraged me to submit to SIGGRAPH Asia, which turned out to be a great piece of advice.

I would also like to thank Kwang Moo Yi for his invaluable guidance on my deep learning work and for being always available to patiently answer the numerous questions I came up with.

I have been very fortunate to meet great researchers during and before my PhD. I am particularly grateful to Hugo Gimbert for taking me as an intern in his lab. This experience motivated me to pursue a research path. Mathias Paulin sparked my interest for computer graphics through his captivating lectures and by supervising my master thesis. I would not have published without my co-authors and I would like to thank them for their contributions: Marco Tarini, J.P. Lewis, Javier von der Pahlen, Miles Macklin, Sofien Bouaziz, Julien Valentin, Shahram Izadi, Daniel Rebain, and Thomas Pellegrini. I am grateful to Alec Jacobson for accepting to be the external examiner of my dissertation defense and flying across Canada for the occasion on short notice.

During my PhD, I was fortunate to be surrounded by great labmates and I would like to thank them for the various discussions that helped me with my research: Maurizio Kovacic, Wei Jiang, Kedi Xia, Weiwei Sun, Sri Raghu Malireddi, Abhishake Kumar Bojja, Nicolas Guillemot, and Daniel Rebain.

Finally, my deepest gratitude goes to my father, my mother, my brother, my sister, and my life partner Xiaolu for their support. I thank my parents for suggesting me to study computer science while giving me the freedom to follow my own path. To Xiaolu, thank you for making me happy.

Chapter 1

Introduction

(a) (b) (c)

Figure 1.1: Examples of recurring primitives in nature. (a) A pine cone is composed on dozens of instances of the same primitive shape. (b) The skin of a snake exhibits a repetition of small diamond-shaped scales. (c) Bees build honeycomb structures that are made of a single volumetric primitive. (Sources: pratheep.com, Care_SMC, Dominic Heard)

1.1 Motivation

What is a geometric primitive? Looking up the word "primitive" in a dictionary does not give a clear idea. A primitive is defined as something "simple", "basic", "elemental", "unrefined", or "not derivative". But the definition that is the most relevant to my research can be found in [Collins, 2004]: "a form from which another is derived". Primitives are, in fact, defined not by their intrinsic properties but by their role within a larger structure. In this research, we consider geometric primitives to be recurring patterns or shapes that compose an object. Just looking through a window and observing how objects are composed should convince the reader that primitives are very common in our world. But the concept of geometric primitives is also important in computer graphics and computer vision, from geometric modeling to pattern recognition and others. We will then have a closer look at the role of primitives in these three domains: nature, computer graphics and computer vision.

(a) (b) (c)

Figure 1.2: Examples of fractals in nature. (a) The Romanesco Broccoli has a fractal-like shape with 4 levels of subdivisions. (b) The branches of a tree split in smaller versions of themselves. (c) A *stellar dendrite* snowflake that is composed of a central hexagon whose corners spawn another hexagon each. Each secondary hexagon spawns small hexagons at each corner. (Sources: Albrecht Fietz, Sunil Patel, SnowCrystals.com)

Primitives in nature

Repetitions & tessellations. We find plenty of instances of repetitions of elements in nature. Animals' skin often exhibit a recurring pattern. This is due to various reasons, involving chemistry [Kondo, 2002] and natural selection [Cott, 1940; Endler, 1980]. For example the skin of a snake is covered by small diamond-shaped scales; see Figure 1.1b. Similarly, many plants' structures are composed of the same primitives with very little variation stemming from the randomness of the natural process. For instance, pine cones are mainly composed of dozens of instances of a single primitive; see Figure 1.1a. The same thing can be observed for a *spiral aloe* or many other plants. The leaves of a tree can be seen as randomly varying instances of the same primitive. Bees build hexagon-shaped structures called honeycombs in order to store larvae and honey; see Figure 1.1c. The striking coherence of this structure can be explained by the fact that this shape is a *near*-optimal solution to the problem of maximizing the cells volume for a given amount of wax [Hales, 2001; Weaire and Phelan, 1994]. One can think of primitives as nature's response to an optimization problem. Because a shape is optimal for a certain function, it will naturally occur in many places.

Fractals. Many plants are not only structured repetitions of the same primitive but also self-similar at different scales. Shapes that are infinitely self-similar and have a fractal dimension are called fractals [Mandelbrot, 1982]. In nature however, infinite repetition is not attainable but there are many examples of few levels of recursion. For instance the Romanesco Broccoli can roughly be seen as a cone-shaped primitive; see Figure 1.2a. But this cone is filled with smaller cone-shaped elements. And these are themselves filled with similar elements. Trees often exhibit a fractal-like growth where the trunk splits into a few branches, each branch splits into smaller branches, etc; see Figure 1.2b. The fern leaves are composed of leaflets that are themselves composed of subleaflets and look similar to a whole leaf. All snowflakes are different but a lot of them exhibit some form of fractal-like shape. This is the case of the *stellar dendrite* in Figure 1.2c that is formed of three levels of hexagons.

Primitives in computer graphics

Efficient rendering. Three-dimensional objects are often represented by a mesh model that explicitly describes its surface [Botsch et al., 2010]. A mesh is a collection of vertices and polygonal faces. Faces are usually simple geometric primitives such as triangles or quadrilaterals. One of the reasons this representation is popular is that rendering triangles can be done very efficiently using a rasterizer [Akenine-Moller et al., 2018]. Rasterizers are typically implemented to run on GPUs that are designed to execute Single Instruction Multiple Data (SIMD). The OpenGL graphics API only offers to render a few primitives: points, lines, triangles, etc [Shreiner et al., 2013].

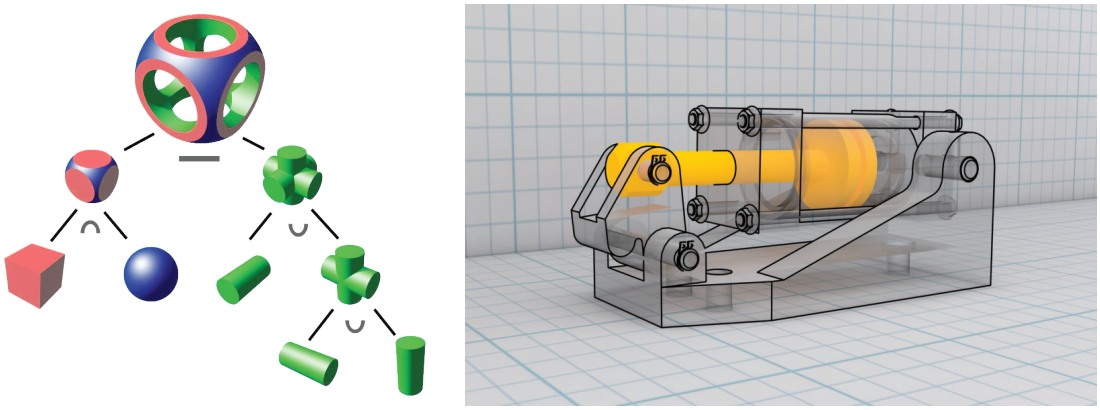


Figure 1.3: (left) A CSG composition tree. Primitives are combined pairwise using boolean operations. In the right branch, cylinders are combined by a union operator ($\cup$ symbol). In the left branch, a cube and a sphere are combined by intersection ($\cap$ symbol). At the root the final object is produced by computing the difference between two shapes ($-$ symbol). (right) CAD softwares use CSG operations to model complex objects from simple primitives. (Sources: Zottie, Frank Hull)

Composition modeling. In vector graphics, images are defined by combining together simple primitives (circles, polygons, curves, etc). These primitives can combined in many ways such as union, difference or intersection. This is exactly the same principle that is applied to 3D in constructive solid geometry (CSG) [Requicha and Voelcker, 1977]; see Figure 1.3. Users can use simple primitives to model complex objects or scenes. L-systems use simple grammar rules to generate realistic shapes such as plants [Prusinkiewicz et al., 1996; Prusinkiewicz and Lindenmayer, 2012] or buildings [Parish and Müller, 2001] with simple primitives.

Scene compression. Streaming of 3D data is becoming popular and has applications in scientific visualization, cultural heritage, digital entertainment, healthcare, robotics, etc. One major challenge is the size of the data that needs to be transferred. Because bandwidth is limited it can cause latency issues which may not be acceptable for some applications, especially for ones where there is a real-time component involved. It is possible to use traditional compression methods but a more efficient way to compress scene data would be to leverage the repetitions and coherency in the geometry of our world. It is possible to represent scenes accurately with simple primitives e.g. man-made environments can often be described by a compact set of recurring elements, such as the windows of a building, or the arrangement of offices and desk in a university. These primitives can be detected or even be learnt and have a much smaller footprint than their corresponding point cloud [Kaiser et al., 2018; Tang et al., 2018a].

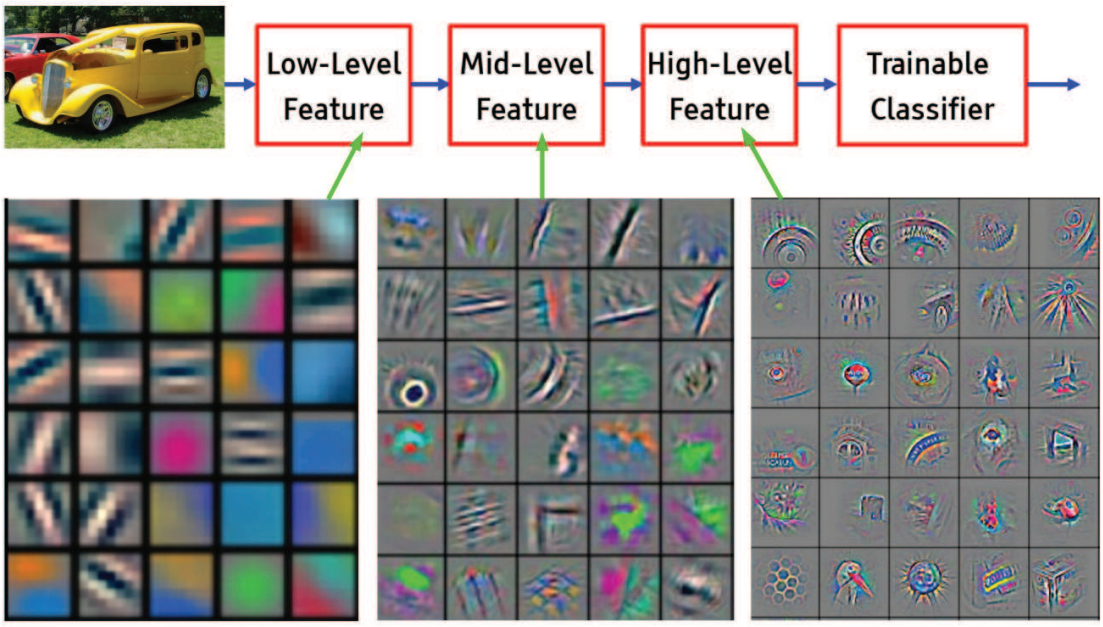


Figure 1.4: Visualization of features maps of a convolutional neural network. (left) The first layers detect edges and intensity contrasts. (middle) The mid-level layers build abstraction upon the previous layers and respond to textures and patterns. (right) The high-level layers identify the present of objects they are trained for in the mid-level feature maps. (Source: [LeCun and Ranzato, 2013])

Physics simulation. Primitives are often used in physics simulation. In the finite element method (FEM) [Zienkiewicz et al., 1977], complex structures are decomposed into tetrahedra or hexahedra to homogenize the computations. In Lagrangian fluid simulation methods fluids are represented with particles that interact with each other and combine to represent a fluid [Ihmsen et al., 2014]. We can simulate complex objects by discretizing them with simple primitives. This is demonstrated by NVIDIA's FleX [NVIDIA, 2018], a popular physics simulation library based on position-based dynamics [Müller et al., 2007; Macklin et al., 2014]. In this framework, everything including soft bodies, cloth and fluids is represented as collections of particles connected by constraints. Using simple primitives also makes a physics engine simpler because there no need for specialized functions for each object. On GPU it is also more efficient to execute the same function on all the objects. Finally primitives are also used for collision detection [Ericson, 2004]. Testing for collisions between two arbitrary shapes is expensive, so objects are often approximated by simpler primitives [Ritter, 1990; Larsen et al., 2000; Rimon and Boyd, 1992]. For instance the bounding boxes of two objects can be used [O'Rourke, 1985]. It is also common to use bounding volume hierarchies (BVH) [Kaplan, 1985; Kay and Kajiya, 1986] where objects are roughly approximated by one primitive and if the collision test is positive the object's shape is refined into several primitives, which can themselves contain more levels of hierarchy.

Primitives in computer vision

Pattern recognition. The human visual system is built upon primitives. The cells connected to the retina respond to small circles within their respective regions. By measuring the response of

cells of the visual cortex to visual stimuli, Hubel and Wiesel [1959] discovered that these cells build upon several dot detections to detect lines. They also identified *complex cells* that use groups of *simple* cells to detect the movement of lines [Hubel and Wiesel, 1962]. Similarly, one could say that computer vision uses simple "primitives" to detect complex patterns and shapes [LeCun et al., 2015]. Nowadays convolutional neural networks (CNN) are the state of the art method for object detection tasks [Liu et al., 2018]. CNNs learn convolutional filters to detect patterns in images. These filters detect recurring spatial patterns, i.e. primitives in the image domain, in pictures of objects they are trained to detect. On the first layer, the filters convolving on the input images are in the image domain. Typically the first layers' filters respond to low-level patterns such as edges and contrast. The ones in the hidden layers respond to primitives in the feature domain. The deeper they are, the higher-level patterns they will be looking for; see Figure 1.4. For instance for face detection, the next layer would be looking to find elements such as noses or eyes. While the last layer should be using looking for a arrangement of eyes, noses, mouth and ears to finally detect a face.

Scene simplification. Primitives can also serve as geometric proxies to simplify a scene and make it easier to understand. In one of the earliest examples of this, Roberts [1963] attempted to reconstruct a three-dimensional object from a single photograph. His method was based on edge detection which were then converted to simple polygonal primitives to reconstruct the 3D object. This method was only applicable to simple shapes with planar faces but more complex objects can be simplified as aggregates of simple primitives. Fischler and Elschlager [1973] proposed a system where objects like planes are abstracted by a template made of generalized cylinders. This template is not a fixed 3D model and has constrained parameters that vary to match images. Similarly, in [Brooks et al., 1979] objects are decomposed as parts such as eyes and noses that are linked by spatial relations. This serves as a template that can be detected in images. It is a similar idea that is exploited in the recent work on "capsule networks" [Sabour et al., 2017; Hinton et al., 2018].

Body tracking. Body tracking has a wide variety of applications including computer-human interactions, performance capture, entertainment and telepresence. Thanks to the introduction of low-cost depth sensors on the consumer market, several recent methods propose to tackle the problem from a geometric alignment point of view. These generative techniques require to have a geometric template that can be aligned to the sensor data. One of the problems with these methods is that their template is either inaccurate or inefficient. Typically, implicit templates are used because of their ability to efficiently compute distance queries. But they require a lot of primitives to represent a hand accurately, which makes them inefficient. To solve this, Tkach et al. [2016] represent their hand template with a *sphere-mesh*, a representation that allows good accuracy with very few primitives.

1.2 Contributions

With the target of modelling a variety of different objects and shapes with simple primitives, we studied how to combine these primitives. Implicit composition operators are very useful for this but they are complex to design. For this reason, in our first project we introduced a method

to design these operators with simple sketches, which is accessible even to non-technical users. In this first project we often used "tapered capsules", a primitive that has the ability to model a lot of shapes by varying its few degrees of freedom. This primitive is also very efficient for distance queries and, most importantly, it can preserve its volume while being stretched. For these reasons, in our second project we applied this primitive to real-time simulation of rods, muscles and soft-bodies. In our final project, we perform generative modeling in an "indirect" way, by automatically detecting and learning the primitives that occur repeatedly in a given signal. This is done without any supervision regarding the location or shape of the primitives.

In summary, the contributions of this research are:

1. **A method to design composition operators**. We introduce a method to design functions that combine implicit primitives via simple sketches. This method enables users to apply composition modeling without requiring any technical knowledge. This is achieved by introducing a template that can represent a wide variety of implicit blending operators. This template is optimized to produce the composition described by a user sketch.

2. **A volume-preserving primitive for real-time simulation**. We introduce an efficient primitive for volume-preserving deformations. This primitive can be applied to the simulation of rods, muscles and soft-bodies. In order to simulate it in real-time, we extend position-based dynamics by considering *scale* as a new degree of freedom.

3. **A weakly supervised method to learn and detect primitives**. We introduce a deep learning framework to automatically identify the structure of repeating elements in a signal. This is made possible by the introduction of a multi-stage process to train through the non-differentiable top-k operation.

Parts of this research have been published in the following papers:

1. **Sketch-based Implicit Blending**
   B. Angles, M. Tarini, B. Wyvill, L. Barthe, A. Tagliasacchi
   Transactions on Graphics, 2017

2. **VIPER: Volume Invariant Position-based Elastic Rods**
   B. Angles, D. Rebain, M. Macklin, B. Wyvill, L. Barthe, J. P. Lewis, J. Von Der Pahlen, S. Izadi, J. Valentin, S. Bouaziz, A. Tagliasacchi
   In Transactions on Graphics (currently under submission to SIGGRAPH 2019)

3. **MIST: Multiple Instance Spatial Transformer Network**
   B. Angles, S. Izadi, A. Tagliasacchi, K. M. Yi
   In ArXiv (current under submission to CVPR 2019)

The following chapters detail the contributions of this research. This dissertation is organized so that each chapter contains its own necessary background.

Chapter 2

Composition of Primitives

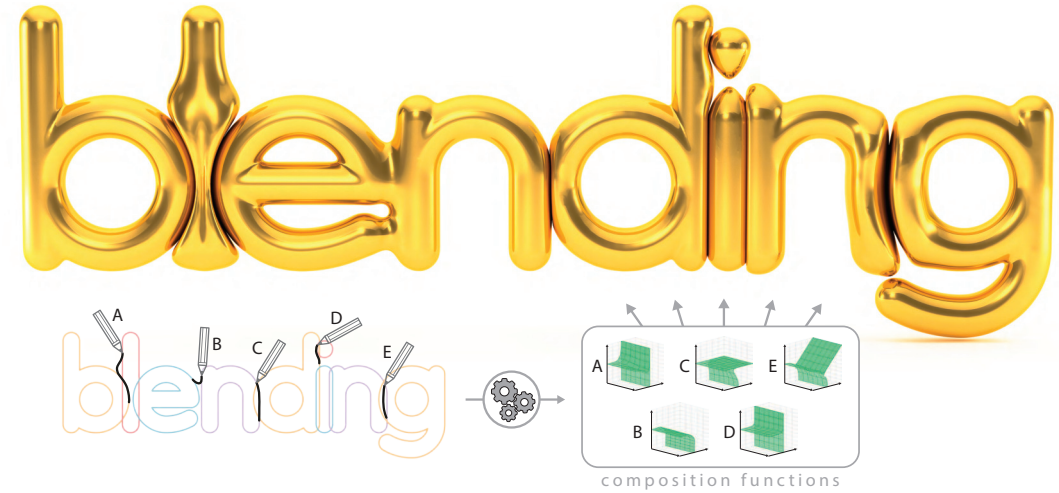


Figure 2.1: A user's 2D sketches (bottom-left) are used to exemplify desired ways in which implicit functions should be composed together. From these, our algorithm automatically derives new custom gradient-based composition operators (bottom-right). These can then be applied to combine any 3D (or 2D) implicit model (top) replicating the user's intentions, and including effects such as contacts, bulging deformation, or smooth blends.

Abstract

Implicit models can be combined by using composition operators; functions that determine the resulting shape. Recently, gradient-based composition operators have been used to express a variety of behaviours including smooth transitions, sharp edges, contact surfaces, bulging, or any combinations. The problem for designers is that building new operators is a complex task that requires specialized technical knowledge. In this work, we introduce an automatic method for deriving a gradient-based implicit operator from 2D drawings that prototype the intended visual behaviour. To solve this inverse problem, in which a shape defines a function, we introduce a general template for implicit operators. A user's sketch is interpreted as samples in the 3D operator's domain. We fit the template to the samples with a non-rigid registration approach. The

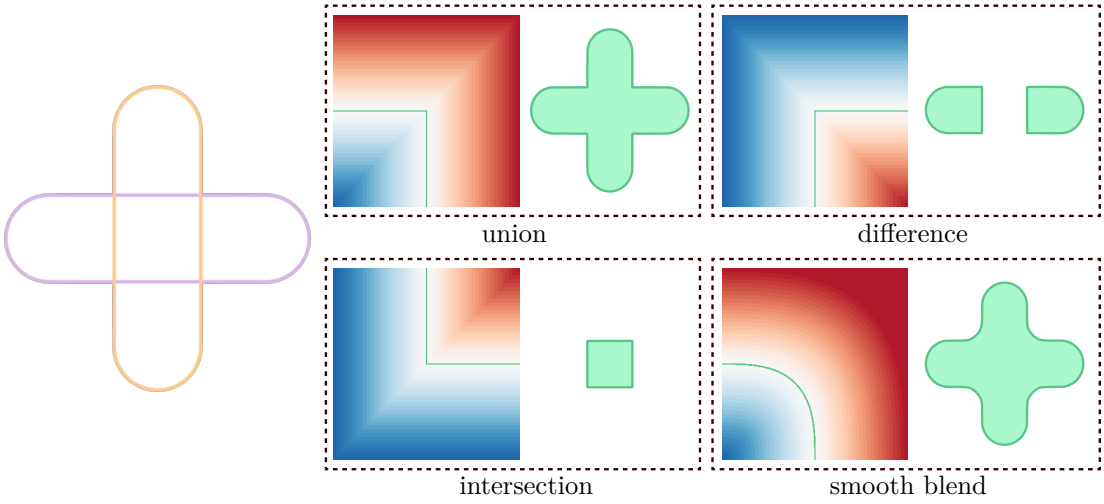


Figure 2.2: Proper design of an implicit composition operator can achieve classical CSG operations such as union, difference, intersection, as well as their smooth variants – i.e. blending.

process works at interactive rates and can accommodate successive refinements by the user. The final result can be applied to 3D surfaces as well as to 2D shapes. Our method is able to replicate the effect of any blending operator presented in the literature, as well as generating new ones such as non-commutative operators. We demonstrate the usability of our method with examples in font-design, collision-response modeling, implicit skinning, and complex shape design.

2.1 Introduction

An implicit representation of a 3D object describes its surface as a set of 3D points on which a scalar function equals a prescribed iso-value [Bloomenthal and Wyvill, 1997]. When modeled or animated, complex objects are defined by assembling their different parts with composition operators, each part being defined by its own scalar function. While the iso-surfaces represent the individual shape of the parts, composition operators control the way they are combined. For instance, the max (min) of two scalar functions produces a union (intersection) operator [Sabin, 1968; Ricci, 1973] which is the basis of *Constructive Solid Geometry* (CSG) [Requicha and Voelcker, 1977]; see Figure 2.2. The blending operator, in some cases a simple sum of the combined scalar functions [Blinn, 1982], smoothes the sharp transition between parts produced by the union. A core feature of implicit representations is that primitives are combined by simply applying an operator to their respective scalar functions, regardless of their relative positions. This means that no detection or specific treatment for collision is required. This is convenient when the combined primitives are particles of a point-based fluid simulation [Ihmsen et al., 2014], or limbs of a character [Vaillant et al., 2013a].

Several composition operators have been proposed, including controlled blending [Hoffmann and Hopcroft, 1985; Rockwood, 1989; Pasko et al., 1995; Hsu and Lee, 2003], localized blending [Pasko and Adzhiev, 2004], and contact operators that model the contact surface where the combined objects are colliding [Cani, 1993]; see Figure 2.1 for a few examples. Even though these extend the

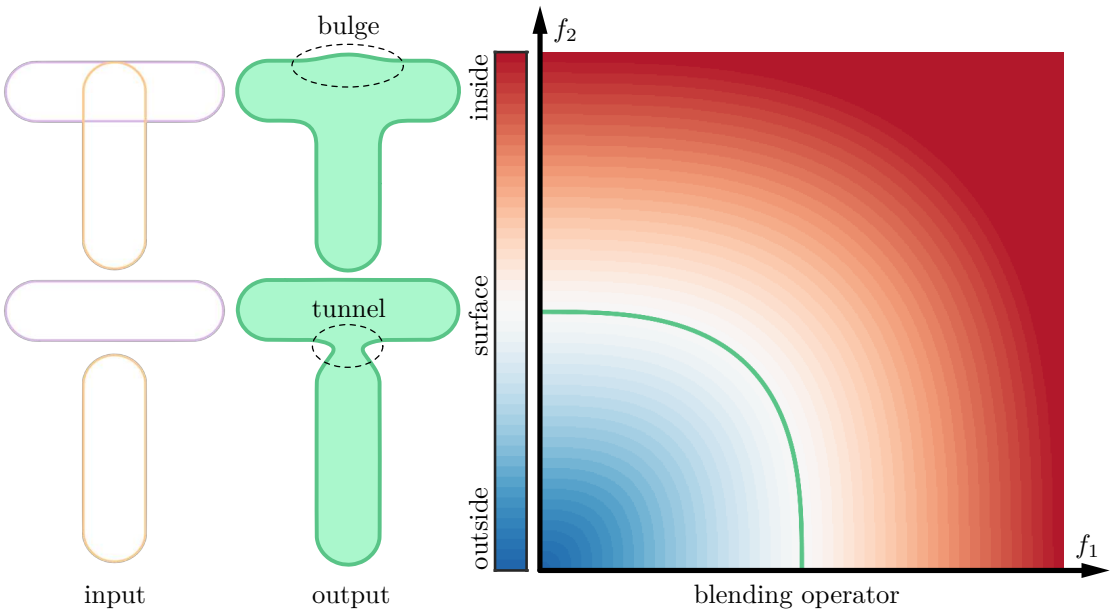


Figure 2.3: A composition operator defined as a function of solely the values of the two input objects presents undesirable bulge and tunnel artifacts.

variety of composition possibilities, they are not commonly used in practice. Some reasons are that meshes are the standard representation for modeling/animation and implicit modeling is not popular on its own, these operators can be computationally intensive, the shape they produce can be unsatisfactory in some cases, and they are often difficult to control by a user.

Recently, gradient-based composition operators [Gourmel et al., 2013] addressed various unsatisfactory shapes in compositions and computationally expensive operator evaluations. Noting that an implicit surface can approximate a mesh by computing a signed distance field [Macedo et al., 2011], implicit skinning [Vaillant et al., 2014] exploits the automatic contact handling of gradient-based operators on 3D scalar functions for deforming meshes more efficiently when they are animated. This is an example of how implicit modeling/animation tools can be *complementary* to existing techniques for mesh processing; the current work provides an effective solution to the generalization and intuitive design of free-form gradient-based composition operators.

Composition operators. The scalar functions $f_a(\mathbf{x}), f_b(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}$ of two objects can be combined with a binary composition operator $g : \mathbb{R}^2 \to \mathbb{R}$, and the function f_c defines the resulting object as $f_c(p) = g(f_a(p), f_b(p))$. Even though by suitable choices of composition operators g a wide variety of transitions can be obtained, many desired behaviors cannot be captured by any choice of the operator [Gourmel et al., 2013]. For example, an operator that produces a *smooth* blend at a transition will also cause a potentially undesired bulging deformation where two objects overlap, as well as premature bulging before the objects make contact; see Figure 2.3.

The gradient-blend operator. Stemming from these considerations, a richer class of operators has recently been introduced by Gourmel et al. [2013]. The key idea is to select the operator at each point depending on the value of the angle θ between the local gradient directions of the two scalar functions. More formally, a gradient-based operator g (from now on, simply an *operator*) is

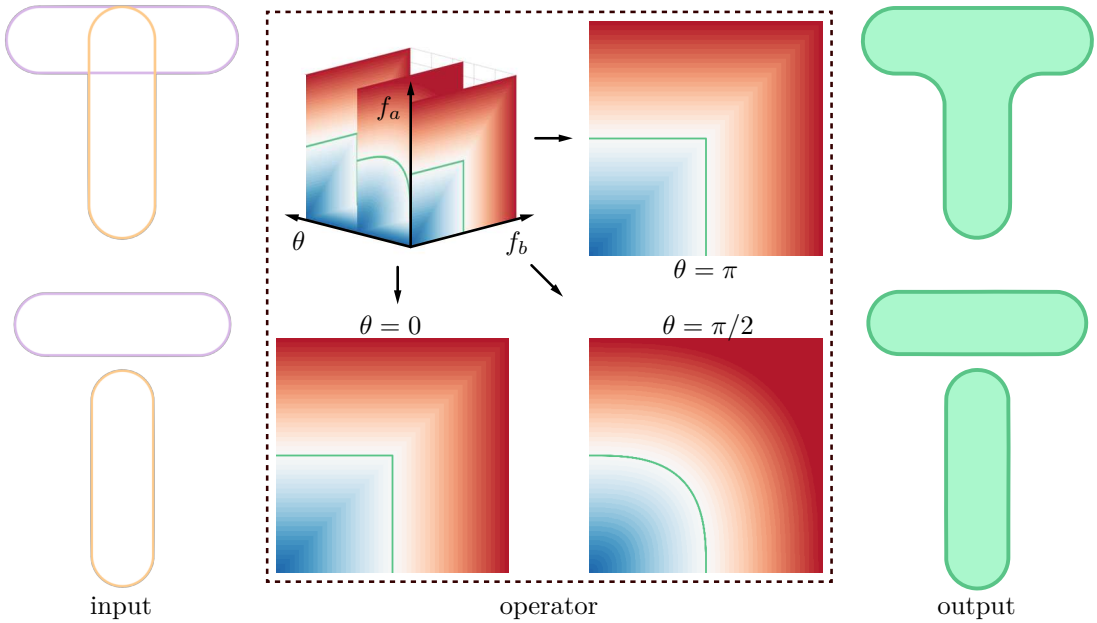


Figure 2.4: Through inclusion of the gradients angle θ, an operator that is capable of resolving the shortcomings of implicit blending can be designed.

a function ($\mathbb{D} \subset \mathbb{R}^3 \rightarrow \mathbb{R}$) that combines two primitives a and b into a new shape c defined as

$$f_c(p) \;=\; g(\, f_a(p),\; f_b(p),\; \theta\,) \tag{2.1}$$

$$\text{where } \theta = \angle(\, \nabla f_a(p),\; \nabla f_b(p)\,) \tag{2.2}$$

$\angle(v, w)$ being the angle between vectors v and w. In the rare degenerate cases where this angle is undefined, because the gradient vanishes, θ is 0. The domain of the operator g, i.e. the *operator-space*, is $\mathbb{D} = [0, 1] \times [0, 1] \times [0, \pi]$

This can be understood as using different composition operators for different values of θ. That is, according to whether the two gradients are in opposite, orthogonal, matching directions, or anything in between. Undesired bulging can be resolved, and pre-contact deformations can be disabled, while in the other cases (intermediate angles) the transition can be kept smooth. Four specific problems of implicit modelling (*bulging*, *locality*, *absorption*, *topology*) were addressed by defining appropriate instances of g [Gourmel et al., 2013]. The main challenges with gradient-based operators is that designing and fine tuning an operator to obtain some desired effect is a highly technical task, and not possible for non-expert end-users such as 3D artists and modelers. Thus, only predefined and fully parametrized operators can be provided to users and they have to be set in the system by experts.

Contributions. In this research, we address this *usability-gap* in composition modeling. We introduce a novel, interactive editing pipeline where the user sketches the desired behavior directly in 2D over one example, and an automatic optimization produces the corresponding operator, transparently to the user. Importantly, the design of an operator and its usage are kept orthogonal: an operator can be applied in any context (2D or 3D) where the same shape or behavior is desired, irrespective of the example where it was designed. Our method can be applied to *any* shape for

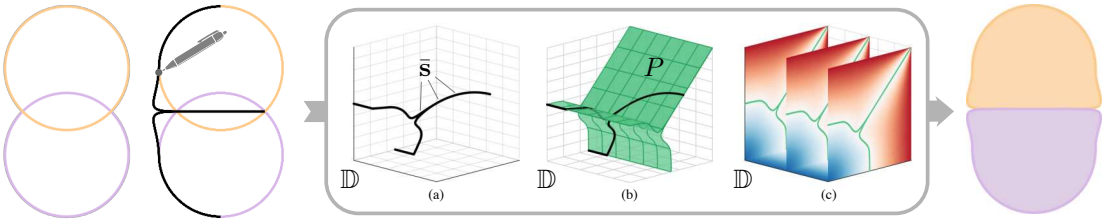


Figure 2.5: Given a pair of input primitives (two overlapping circles), the user sketches the desired resulting shape (a contact surface and a bulging effect). (a) The system generates a set of samples in the operator space $\mathbb{D}$. (b) An operator template is fitted to the samples. (c) A dense regular sampling of the operator g is generated. The resulting operator, if applied to the initial pair of primitives, produces the desired behavior and can be applied wherever this effect is desired.

which we can compute a signed distance function. The following contributions were necessary to achieve this result: (1) we introduce a new template to represent operators, which avoids the use of transfer functions such as those proposed in [Gourmel et al., 2013], and is suitable both in the design and in the application phases; (2) we present a way to map user-sketches into samples of operator-space $\mathbb{D}$; (3) we observe that the problem of fitting this template to the samples can be cast as a deformable surface registration problem, and we identify suitable regularizers; (4) we introduce the concept of non-commutative operators, which we show to be useful in certain scenarios, and finally (5) we introduce some novel interesting applications for these operators.

2.2 Related Work

Because of the ease of representing arbitrary and changing topologies, CSG, blended and contact surfaces, implicit modeling has some advantages over traditional surface models [Marschner and Shirley, 2015]. After presenting the previous works on controllable composition operators and an overview of sketch-based implicit modeling, we review some potential applications of our operators.

On freeform operators. Over the years, several operators have been designed to try to fill the gap between their mathematical formulation and their manipulation by end-users. In aesthetic blends, Pasko and Savchenko [1994] optimize the three parameters of an algebraic blending operator to approximate a user's sketch. More complex free-form 2D operators have been defined by blending implicit lines [Barthe et al., 2003] or by deforming a blending operator [Barthe et al., 2004]. These operators are subject to all the limitations solved by gradient-based operators and are not compatible with this operator formulation. Rather than focusing on the operator shape, Pasko et al. [2005] and Bernhardt et al. [2010] propose to localize the influence of blending operators on the combined objects by adding an additional 3D scalar function, placed automatically or user-defined. These approaches focus on the definition of where the blending should occur, and no explicit control is performed on the shape of the operator itself. As introduced by Gourmel et al. [2013], the shape of the gradient-based operators are defined by a set of 2D profile curves in cylindrical coordinates. This definition is unintuitive to manipulate and restricted in the variety of shapes it can produce. To achieve better skinning with accurate contact deformations, Vaillant et al. [2014] introduced gradient-based operators discretely computed in a 3D grid, 2D slice by 2D

slice, by bi-harmonic interpolation of Dirichlet constraints at boundaries and additional constraints on the iso-value. Their specification relies on a lengthy trial and error process, consisting of editing a set of spline curves for a given θ, as well as a trigonometric transfer function to non-linearly interpolate these curves along the θ axis. Furthermore, an interactive exploration is not feasible, as a re-computation of the computationally intensive fairing optimization is necessary on each update. Finally, their operator construction is tailored to a small set of effects useful in the targeted context (symmetric contacts and bulges), whereas our operators are generic. In our work, we enable the specification of freeform gradient-based operators at interactive rates through the use of 2D annotations, which directly describe the intended user-defined behavior.

Sketch based implicit systems. Sketches have long been recognized as a powerful tool for modeling [Igarashi et al., 1999]. Sketch-based implicit systems added the ability to do blending and CSG with volume models in work such as [Singh and Fiume, 1998; de Araújo and Jorge, 2003; Alexe et al., 2005; Tai et al., 2004], and there have been several examples, including the popular ShapeShop by Schmidt et al. [2005]. [Singh and Fiume, 1998] shares with us the idea that the final surface shape is modelled by a 3D curve. Closer to our proposal, Karpenko et al. [2002] built models using an implicit representation based on Radial Basis Functions. Their system used the input stroke to edit a mesh, which would in turn change the implicit representation. This implicitization approach changes the field locally according to the user's edit. In the above approaches, sketches define the shape in one particular modeling instance. The input sketch in our system defines an operator, which is not tied to the context where it is defined, but can be applied wherever the user desires.

2.2.1 Applications

This work impacts several application domains, offering interesting contributions in each of them.

Character skinning. In the context of character skeleton-driven animations, composition operators have been used to achieve more realistic procedural skin distortions [Vaillant et al., 2013a, 2014]. This is also a motivation for our work. With respect to these applications, our approach offers the ability for the designer to intuitively sketch the exact intended deformation (e.g. skin bulging) in one instance, and produce an operator which will reproduce that deformation in real time. This approach is analogous to example-based deformation schemes (see [Jacobson et al., 2014b] or [Shi et al., 2008]), but in our case, the exemplar sketch can be drawn in 2D, and the extracted operator can be directly applied to any other joint.

Font design. One potential application of our method is to assist font-design, where glyphs for each letter are the result of composition operators in 2D from a pre-defined skeleton; similarly to [Suveeranont and Igarashi, 2010]. The literature on font design is large, and covers many problems which are not addressed here, including automatic construction of the skeletons, or a consistent cross-parameterization for all glyphs, or even a generative manifold of all possible fonts; see [Campbell and Kautz, 2014]. In pipelines where the skeletons of each glyph becomes available, our method offers the ability to control the shape of one glyph (e.g. serifs and joints), and apply it consistently to the entire type-face.

Botanical modeling. In the context of procedural synthesis of botanical models (both realistic and stylized), implicit models have been recognized early as suitable solution, due to their natural ability to recreate smoothly blending branching structures [Bloomenthal, 1995; Hart and Baker, 1996; Galbraith et al., 2004]. Using our tool, a user can simply trace the required shape to mimic the geometry of these features, and incorporate this effect into the operator.

2.3 Method Overview

A visual outline of our framework is illustrated in Figure 2.5. Our design process begins with the user placing two exemplar implicit primitives in a 2D sketch. The user then annotates the desired blending behavior by sketching a curve. Given this input, an automatic system derives an operator g by solving an inverse optimization problem; see Section 2.6. The resulting operator can then be used both to combine implicit curves in 2D and to combine surfaces in 3D. This observation is crucial, as it allows the user to simply work in 2D to produce operators for 3D modelling. This is even more relevant for the design of contact surfaces, which would otherwise be difficult to edit (or even just to visualize) in 3D, due to self-occlusions.

Feedback loop. In many cases, a single set of user sketches are sufficient to fully determine the operator g that produces the desired result; see Figure 2.5. For more complex cases, an iterative feedback loop can be used to refine the operator and at the same time observe its effect; see Figure 2.6. First, an operator g_0 is constructed from an initial sketch and automatically applied to the exemplar primitives. The resulting 2D drawing provides feedback to the user. The user can then add new sketches, and a new operator is produced from the union of all sketches. This is repeated until a satisfactory shape is returned to the user, limited only by the expressiveness of the gradient-based approach; see Section 2.7. In practice, we found we needed no more than three iterations. For the feedback loop to be interactive, we need to ensure that our algorithms are computationally efficient for the optimization of the operator and for its application.

2.4 Background and preliminaries

An implicit model (surface in 3D, or contour, in 2D) is defined by a scalar field-function f, as the set of the points where f assumes a given iso-value. Following the convention from Bloomenthal and Wyvill [1997], we define the surface as $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n | f(\mathbf{x}) = 0.5\}$ which bounds an interior where $f(x) > 0.5$. The field-functions of the primitives are, in turn, defined by their *skeleton*. For example, a sphere is generated by a point skeleton, and a capsule (a cylinder with hemispherical caps) by a line-segment skeleton. Any other shape can be used; see Figure 2.13,2.18 for some examples.

Support and continuity. The field has a value which decreases with the distance from the skeleton, according to a given falloff function; see [Marschner and Shirley, 2015]. The area where the field function has values $|f(\mathbf{x}) - 0.5| < 0.5$ is denoted the *support* of the implicit object. Outside its support, the field function equals zero and the primitive has no influence on the composition operations. The support is compact, i.e. bounded; see Figure 2.7.

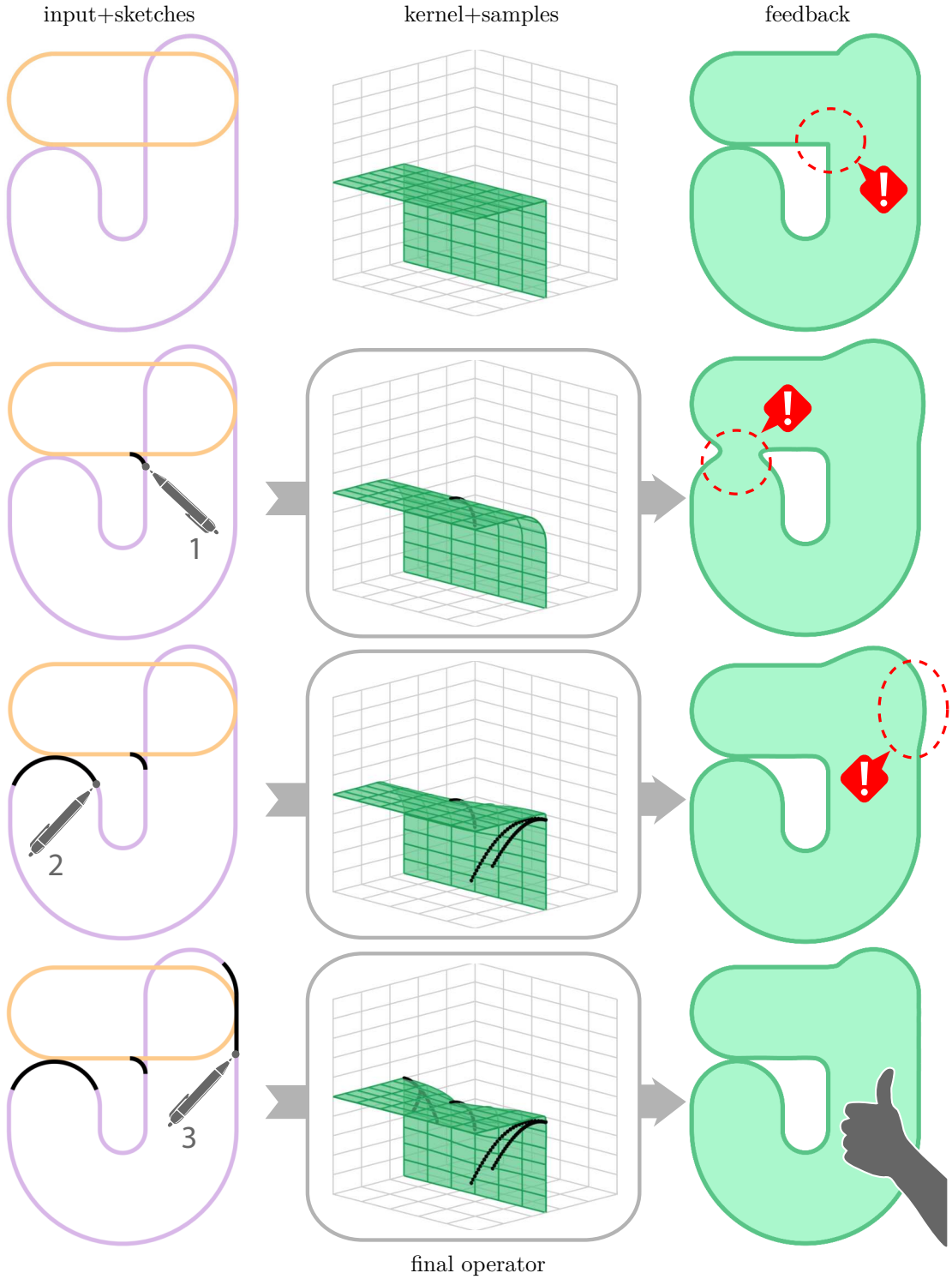


Figure 2.6: Operator design feedback loop: because our pipeline (Fig. 2.5) has interactive response times, our system allows progressive refinement of the operator though successive strokes. Undesired blending artifacts (right) are corrected until a final operator is constructed which is capable of reproducing the desired behavior.

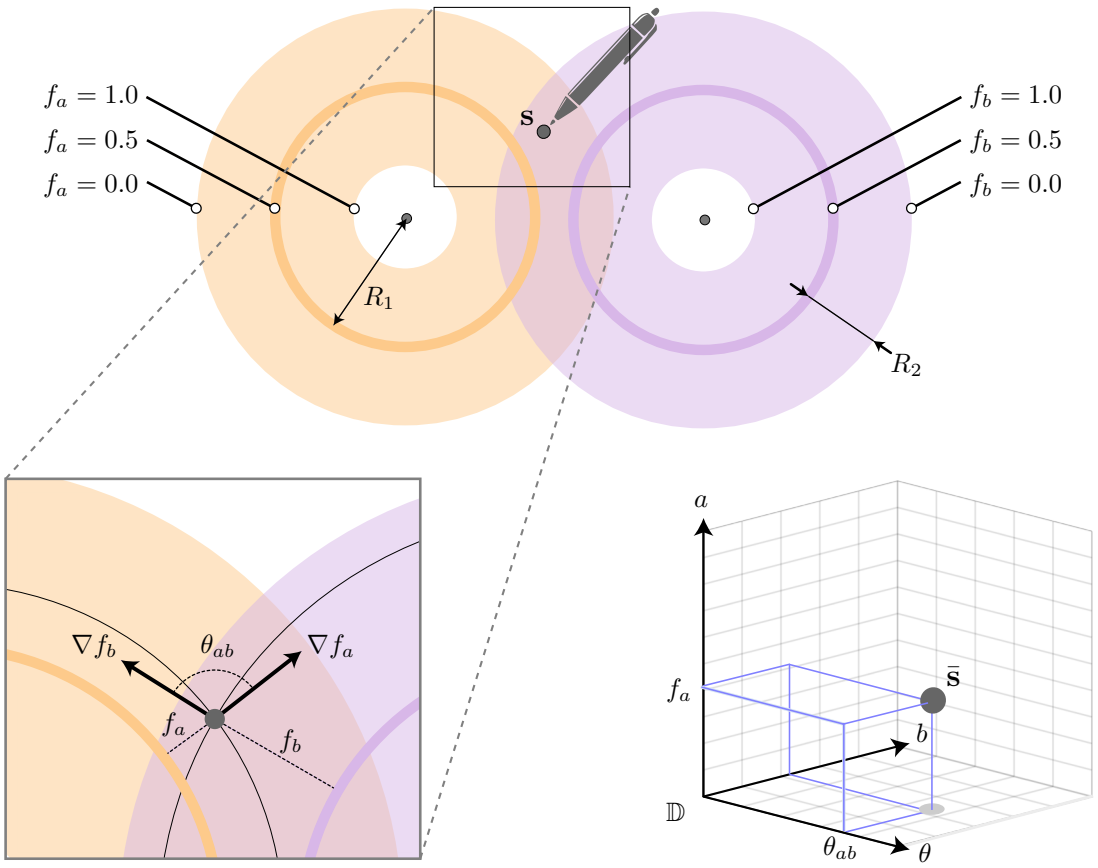


Figure 2.7: Top: a sample $\mathbf{s}$ is drawn in the intersection of the supports of the two primitives (bold colored lines) generated by the skeletons (black dots). Lower-left: a zoom-in around $\mathbf{s}$: the gradients of the two field functions f_a and f_b form an angle θ. Lower-right, the corresponding sample $\bar{\mathbf{s}}$ in the operator's domain $\mathbb{D}$ (see Eq. 2.5).

A central concern in implicit operator design is to ensure smooth blends and avoid normal discontinuities at the boundaries of the support. To this end, where functions such as min or max are used for g [Barthe et al., 2003], filter fall-off functions are required to be at least C^1. In our approach, where we fully control the composition function g, this requirement can be completely dropped. Instead, we rely on functions g with *built-in smoothness*, by defining appropriate value and derivative constraints at the boundaries of its domain $\mathbb{D}$; see Section 2.6.3. This observation allows us to use any monotonic C^0 fall-off function for our primitives. In our examples we opt for a simple linear function controlled by two intuitive parameters: R_1, the iso-value of the implicit model, and R_2, the thickness of the support; R_1 is mapped to $1/2$ and $R_1 + R_2$ to 0 (see Figure 2.7, top). In the example of Figure 2.13, the curly branches are obtained by linearly interpolating two values of R_1 along the skeletal curve of the branch.

Intersection and difference. In this research, we concentrate on *union* composition operators g, which fuse two primitives into one in some prescribed manner, i.e. g always returns 1 when *either* of the first two parameters is 1, and 0 when both are 0. Generalization to intersections and

differences is straightforward using the same g:

$$g_{intersection}(a, b, \theta) = 1 - g(1 - a, 1 - b, \theta) \tag{2.3}$$

$$g_{difference}(a, b, \theta) = 1 - g(1 - a, b, \theta) \tag{2.4}$$

Implicit composition. The creation of complex geometry requires the composition of more than just two input functions. Following the ideas introduced by Wyvill et al. [1999], we employ different *binary* operators at each node of a tree, with primitive shapes at its leaves. For example, see Figure 2.1 where different operators are used in cascade.

2.5 Capturing user inputs

In our system, a pair of 2D primitives are arranged freely by the user; see Figure 2.6 for an example. The linear field functions f_a and f_b of the two primitives are expressed in closed form, and to the user, the two primitives are visualized as closed poly-lines. In Figure 2.7, we also visualize the supports S_a and S_b of the two shapes. By construction, the target operator can only determine the resulting shape in the area $S_a \cap S_b$, therefore user input strokes are restricted to be inside this area with a stencil mask. The user draws the desired shape of the resulting surface over the input primitives, with one or multiple sketches. For most experiments we employ parametric curves, but any drawing method such as the strokes in Figure 2.20c, can be used. As described below, this is possible as only a *sampling* of the sketches is required to derive an operator.

Sampling user input. From the user's sketch, we extract a set of n samples $\{\mathbf{s}_1, \dots \mathbf{s}_N\}$. Each sample $\mathbf{s}_n \in \mathbb{R}^2$ represents a position that the user expects the result/output surface to cross. As illustrated in Figure 2.7, for each sample $\mathbf{s}_n$ we define a corresponding sample $\bar{\mathbf{s}}_n$ in the operator domain $\mathbb{D}$:

$$\bar{\mathbf{s}}_n = \begin{pmatrix} a_n \\ b_n \\ \theta_n \end{pmatrix} = \begin{pmatrix} f_a(\mathbf{s}_n) \\ f_b(\mathbf{s}_n) \\ \angle(\, \nabla f_a(\mathbf{s}_n)\,,\, \nabla f_b(\mathbf{s}_n)\,) \end{pmatrix} \tag{2.5}$$

Computations of $\bar{\mathbf{s}}_n$ from $\mathbf{s}_n$ are conveniently fast because functions f_a and f_b are available in closed form. Their gradient is either available in closed form or approximated by finite differences. The regressed operator g evaluated at $\bar{\mathbf{s}}_n$ should return the value (0.5), or in other words $g(\bar{\mathbf{s}}_n) = 0.5$. Hence, after optimization, the designed blending operator kernel g should *interpolate* each sample $\bar{\mathbf{s}}_n$.

Sketches over 3D rendering. As a variation, sketches can be drawn over a 3D rendering of the implicit surfaces, seen from arbitrary viewing angle; e.g. see Fig. 2.17. In our prototype, the sketch is in this case assumed to lie on an plane parallel to the image at a predefined depth. In situations where the gradients of the primitives are not co-planar, there is no ideal plane to sketch on and results could be unintuitive. In this case, it is often better for the user to sketch on a simpler configuration of primitives and to then apply the operator to the original scene. Our framework requires no further adaptation to deal with this case. A more advanced interface could identify depth automatically, for instance by projecting the first point of the sketch on the shape, similarly

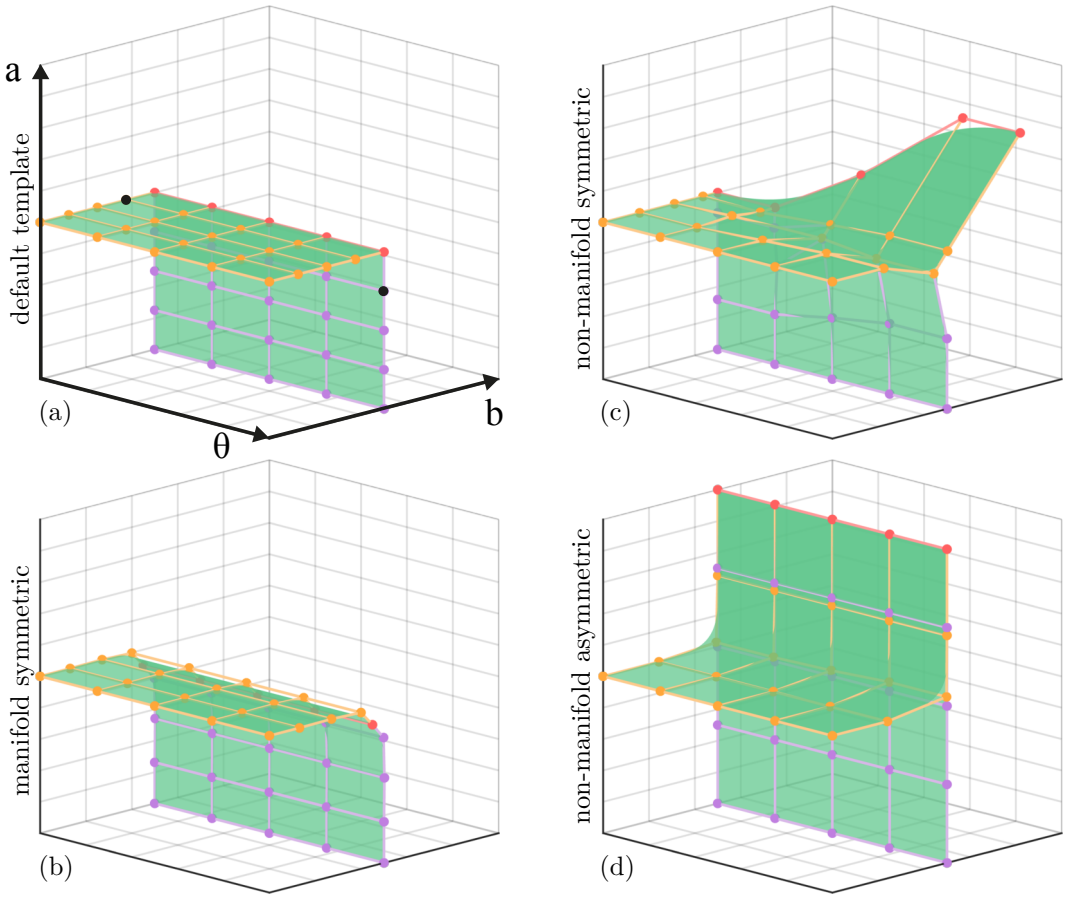


Figure 2.8: Our template is a surface P is made of two bi-quadratic patches, depicted with orange and violet control points – red ones are shared, creating a seam at the junction. This template is able to represent a variety of useful composition operators; see Section 2.6.2.

to the sketch-based modeling interface proposed by Bernhardt et al. [2008].

2.6 Fitting the composition operator

The blending operator requires a 3D function g to be defined over its entire space $\mathbb{D}$, however samples collected from user's strokes only define the behavior of g in a small subset of $\mathbb{D}$. On the other hand, many characteristics of the function g are known *a-priori*, such as its general shape (Sec. 2.6.1), its continuity requirements (Sec. 2.6.2) and the values on the boundaries of $\mathbb{D}$ (Sec. 2.6.3). We approach this reconstruction problem in two steps: first, we identify the set of 0.5 values of g, as a parametric surface P embedded in $\mathbb{D}$, which is fitted to the samples (Sec. 2.6.4); we then compute a 3D lattice covering the domain of g by propagating the iso-values in P (Sec. 2.6.5). The final operator is then evaluated by tri-linear interpolation of the lattice values.

2.6.1 The operator template

In previous work [Gourmel et al., 2013; Vaillant et al., 2013a, 2014], $g(a, b, \theta)$ is formulated as a collection of two dimensional functions for a few particular values of θ, each independently defined as a curve defining the portions of the domain where g evaluates to (0.5). These cases can be interpreted as axis aligned slices of the domain $\mathbb{D}$. In our work, we conveniently represent the operator g as one *surface* P embedded in $\mathbb{D}$ representing its (0.5) iso-value; see Figure 2.5b. This approach allows us to define a template for the surface P, designed to represent a wide class of useful operators: sharp unions (see Figure 2.8a), smooth blends (see Figure 2.8b), articulated contact (see Figure 2.8c) and asymmetric contact (see Figure 2.8d) amongst many others; see Figure 2.21.

Parametric representation. Our template for P is a surface made by a pair of third-order B-Spline patches, each with $I \times J$ control points, joined at their boundary and arranged as in Figure 2.8a. The surface P is fully determined by the positions of the control points $p^a_{i,j}$ and $p^b_{i,j}$ in $\mathbb{D}$. We found I=J=5, for a total 45 distinct control points, to provide the necessary expressiveness while avoiding excessive redundancy. The continuity of P at the junction is enforced by imposing $\forall i \in [1..I] : p^a_{i,J} = p^b_{i,J}$, while other boundary and regularization constraints will be discussed later.

2.6.2 Template expressiveness

A fundamental characteristic of our operator template lies in its expressiveness. In particular, in Figure 2.21 we illustrate how, to the best of our knowledge, *all results obtained by any composition operators which have been proposed in the literature can be expressed by our template*. We now detail how several operators can be realized by properly deforming our template.

Sharp creases: if a sharp crease is desired in the transition between the two operand surfaces (e.g. with union) then g needs to break C^1continuity, and consequently P must have a normal discontinuity. In our template, this discontinuity is easily accommodated by construction at the junctions between the two splines.

Smooth blends: if g is required to generate surfaces without any sharp creases in the transition between the two operand surfaces, it needs to be C^1continuous, and consequently, surface P must be smooth, including at the junction between splines. This can be easily obtained by aligning the points $\{p^a_{i,J-1}, p^a_{i,J} = p^b_{i,J}, p^b_{i,J-1}\}$.

Contact surfaces: another important feature of operators is the ability to produce *contact surfaces* at the transitions between the two primitives [Cani, 1993; Vaillant et al., 2014]; see Figure 2.21efg. Our template can easily reproduce this situation by making the two patches partially coincide, thus realizing a non-manifold operator.

Symmetry: many existing composition operators g are commutative in their first two parameters, which in our setup makes surfaces P symmetric with respect to the $a = b$ plane in $\mathbb{D}$. Given $\phi : \mathbb{D} \to \mathbb{D}$ is the planar mirroring transformation $\phi(x, y, z) = (y, x, z)$, this can be easily obtained by enforcing $p^a_{i,j} = \phi(p^b_{i,j})$. Clearly, non-commutative operators, such as those in Figure 2.8d can also be represented.

In summary, our template can recreate each of the features above. If required, our system allows users to explicitly enforce these conditions; however, it is not strictly necessary to do so, as surface P will naturally conform to these conditions whenever they are suggested by the user data as a result of the fitting process. This observation can aid the design of user interfaces based on our method.

2.6.3 Boundary conditions

When $f_a(p) = 0$, the point p is outside the (compact) support of f_a, and hence beyond its range of influence, so f_c should exactly reproduce the values of $f_b(p)$ to ensure C^0 continuity. Analogous considerations apply to the $f_b(p) = 0$, $f_a(p) = 1$ and $f_b(p) = 1$ boundary planes of $\mathbb{D}$, leading to the constraints:

$$\begin{aligned} \forall\theta, \forall a: \quad & g(a, 0, \theta) = a \quad \text{and} \quad g(a, 1, \theta) = 1 \\ \forall\theta, \forall b: \quad & g(0, b, \theta) = b \quad \text{and} \quad g(1, b, \theta) = 1 \end{aligned} \tag{2.6}$$

Further, to achieve (normal) *shading* smoothness at the boundaries of the supports, we must enforce C^1continuity of the blending operation by vanishing the derivatives:

$$\forall\theta, \forall a: \ \frac{\partial g}{\partial b}(a, 0, \theta) = 0, \quad \forall\theta, \forall b: \ \frac{\partial g}{\partial a}(0, b, \theta) = 0 \tag{2.7}$$

As θ represents the *unsigned* angle between the two gradient vectors, the function g is implicitly mirrored at both ends $[0, \pi]$ of its third parameter. Therefore, to achieve C^1continuity we also impose:

$$\forall a, \forall b: \frac{\partial g}{\partial a}(a, b, 0) = 0 \quad \forall a, \forall b: \frac{\partial g}{\partial b}(a, b, \pi) = 0 \tag{2.8}$$

The constraints above translate into Dirichlet and Neumann boundary conditions for surface P, which can be conveniently expressed in terms of linear hard constraints on its control points. To enforce Equation 2.6, the control points on opposite ends of P are constrained to lie on the two line segments at the boundary of $\mathbb{D}$: $(0.5, 0, \theta)$ and $(0, 0.5, \theta)$. To enforce Equation 2.7 and Equation 2.8, the control points neighboring the boundaries of P must be constrained to be vertically/horizontally aligned to the boundary control points.

2.6.4 Surface registration

In this phase, we fit the template parametric surface P to the collected samples $\bar{\mathbf{s}}_i$. P maps each point $uv = (u, v)$ of its 2D parametric space into a position $P(uv) \in \mathbb{D}$, as determined by the control points $\{p^k_{i,j}\}$. We start with an initial guess, which corresponds to a union operator: a surface P where the two B-spline patches are simply planar and reciprocally orthogonal; see Figure 2.8a. We then non-rigidly register the template onto the samples following the approach in [Bouaziz et al., 2016]; see Figure 2.9. This optimization consists of the alternation of two

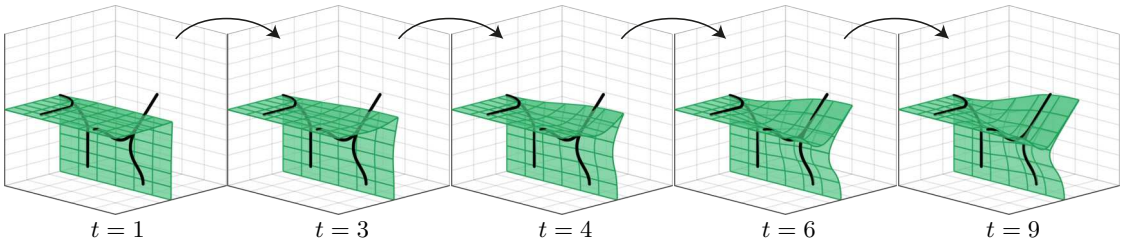


Figure 2.9: We illustrate the sketches in operator space, and a few iterations of the operator registration optimization in Equation 2.10. The manifold surface folds onto itself to be able to reproduce a non-manifold configuration. At the same time, the way we enforce the symmetry constraint helps the optimization to avoid undesirable self-intersections. In this example, the optimization converged after 9 (two-steps) iterations; none of our experiemnts required more than 20.

optimization steps:

$$\textit{local:} \quad \underset{(uv_n)}{\arg\min} \quad \|P(uv_n) - \bar{\mathbf{s}}_n\|_2, \quad \forall \bar{\mathbf{s}}_n \tag{2.9}$$

$$\textit{global:} \quad \underset{\{p^k_{i,j}\}}{\arg\min} \quad E_{match}(\{p^k_{i,j}\}, \{uv_n\}) + E_{priors}(\{p^k_{i,j}\}) \tag{2.10}$$

That is, our local-global ICP optimization first computes closest-point projections, and then it globally modifies the surface control points, resulting in a non-rigid deformation of our surface. For sample points corresponding to a contact surface, both sides of the template must be separately fitted to these. These sample points could be labeled automatically but we let the user provide this information. To efficiently implement the local step, we first triangulate P (we employ a resolution of 40×40), and use a regular volumetric grid to accelerate closest-point queries from $\bar{\mathbf{s}}_n$ to the triangles. We also constrain each of the I rows of control points to lie at a constant equally spaced θ values. The global step is implemented as one Least Squares minimization, as both energy terms are quadratic in the variables; E_{match} represents the data-to-model error, while E_{prior} accounts for shape-priors as well as optimization regularization. Further implementation details are available in our publicly released source code.

Matching term. The data-to-model error is computed as the averaged squared distances of samples from the tangent planes of their projections onto P:

$$E_{match} = \frac{1}{N} \sum_n [\mathbf{n}_n \cdot (\bar{\mathbf{s}}_n - P(uv_n))]^2 \tag{2.11}$$

where $\mathbf{n}_n$ is the normal at $P(uv_n)$. This point-to-plane metric leads to better convergence compared to point-to-point errors. The term is a quadratic function of the variables, because $P(uv_n)$ and $\mathbf{n}_n$ are both constants in the global step.

Priors. The prior energy includes several terms weighted by parameters. These terms enforce: operator *fairness*, potential *contact* constraints, and *regularization* of the optimization:

$$E_{prior} \;=\; w_{fair}\, E_{fair} \;+\; w_{contact}\, E_{contact} \;+\; w_{tikh}\, E_{tikh} \tag{2.12}$$

Fairness prior. We penalize oscillations in the surface with a bi-harmonic energy defined on the control points $p^k_{i,j}$:

$$E_{fair} = \sum_{i,j} \|\Delta_{uv} p^k_{i,j}\|^2 \tag{2.13}$$

where Δ_{uv} represents the 2D Laplacian operator defined in the uv domain, adapted to have control vertices stored in matrix form $p^k_{i,j}$. This regularizer has multiple advantages: (1) in under-sampled areas it ensures a *smooth interpolation*, (2) it prevents *over-fitting* and, (3) as the sketches only provide a sparse sampling in $\mathbb{D}$, it regularizes our optimization ensuring the problem remains well-conditioned. Additionally, these fairness energies are known to penalize surface fold-overs [Botsch and Kobbelt, 2004]. Following [Li et al., 2008], we start by a strong enforcement of fairness to avoid local minima, and progressively relax this constraint to allow the surface to eventually closely fit to the data. Specifically, we employ the weight scheduling $w_{fair} = 10^3 \cdot 2^{-t} + 10^{-4}$, where t is the iteration number.

Full-contact prior. When we detect that the user sketch corresponds to a full-contact (e.g. see Figure 2.21e, where two objects are *fully* separated by a contact interface), we also enable a prior that ensures the *seam* control points connecting the two patches of our template project on the boundary of the domain $\mathbb{D}$:

$$E_{contact} = \sum_{(i,j)\in\mathcal{S}} \prod_{\{\mathbf{n}_m\}} \|\mathbf{n}_i \cdot (p_{i,j} - [1,1,1])\|^2 \qquad \mathbf{n}_1 = [1,0,0] \quad \mathbf{n}_2 = [0,1,0] \tag{2.14}$$

Because of the multiplication $\prod$, this energy is non-linear, hence when computing its gradient we only enable the term that has the smallest point-to-plane residual.

Tikhonov regularization. As the fitting energy is *linearized* within each local step, we avoid over-shooting with a mild Tickhonov regularizer, by setting $w_{tikh} = 10^{-3}$ which penalizes displacements from the previous solution:

$$E_{tikh} = \sum_{i,j,k} \|p^k_{i,j}(t) - p^k_{i,j}(t-1)\|^2 \tag{2.15}$$

2.6.5 3D-Lattice filling

Once the surface P describing the (0.5) iso-values of g is defined, the next step is to regularly sample its domain $\mathbb{D}$ and assign scalar values on each cell of the lattice. This is executed in three sub-steps:

Step 1 – Initialization: we assign the voxels immediately surrounding P with the signed distance from either of its two patches, by rasterizing them over the 3D lattice. Grid receiving two values are set to the greatest (i.e. most internal) one; this ensures robust evaluation when the two patches are coplanar, as in the case of contact operators, as the contact surface is in the interior of the object.

Step 2 – Boundary assignment: for each θ slice of $\mathbb{D}$, we assign the values to the four sides

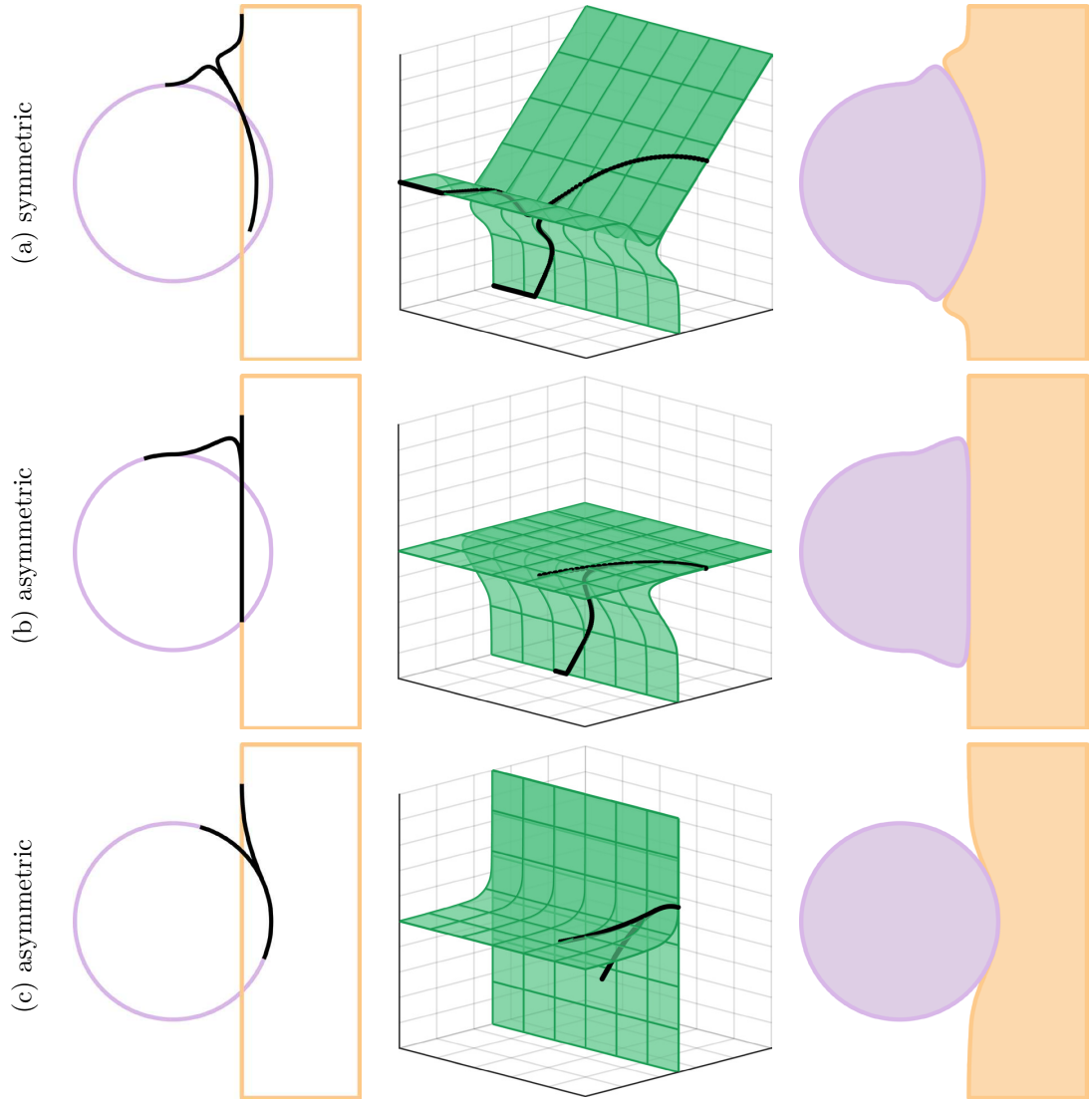


Figure 2.10: (a) *Symmetric* contact operators a-la [Cani, 1993] can be constructed through the enforcement of symmetry in $\mathbb{D}$. We introduce *asymmetric* contact operators which can model phenomena as: (b) a rubber ball hitting a concrete wall, or (c) a steel ball hitting a sheet of softer metal.

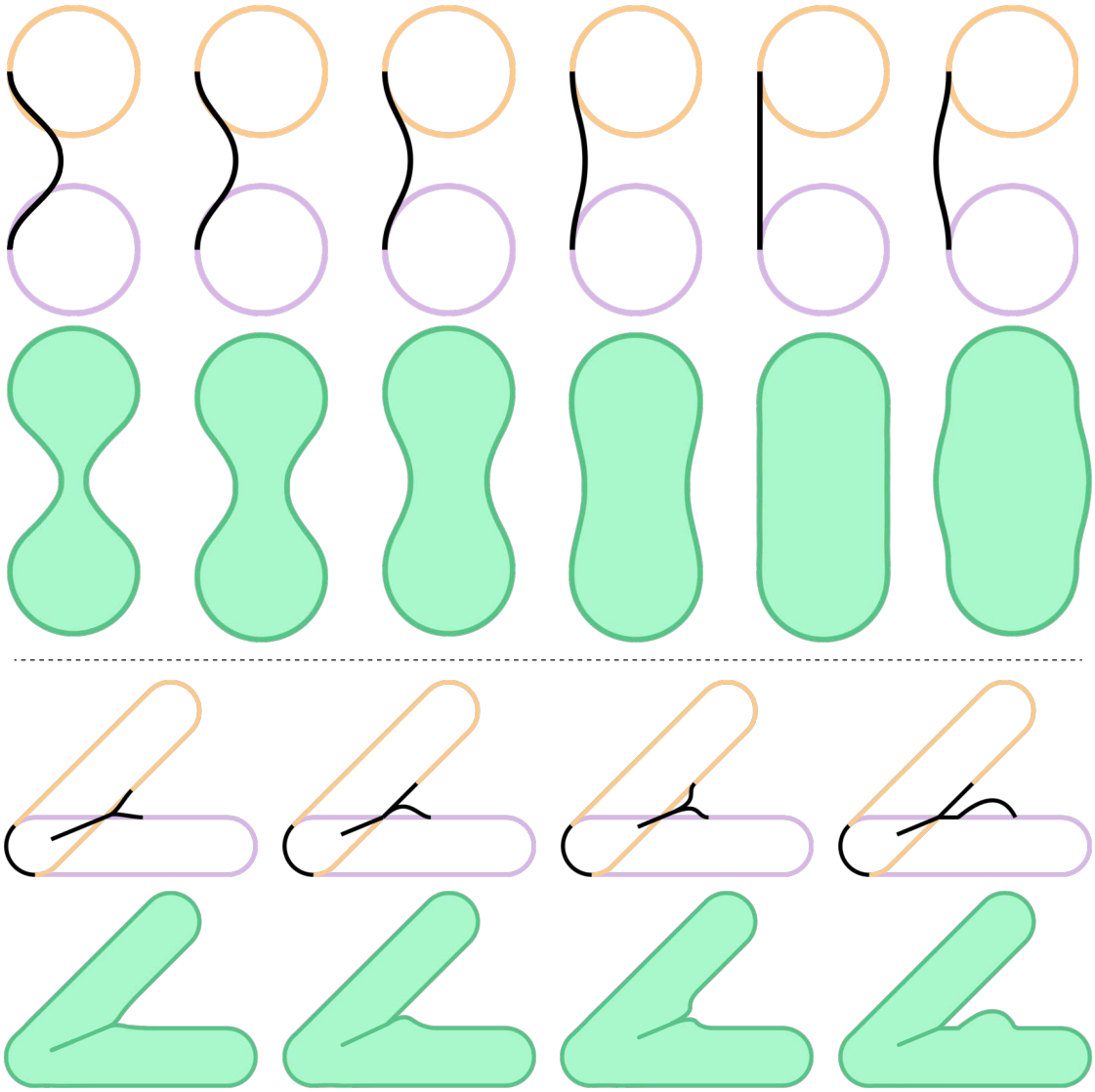


Figure 2.11: Our sketched implicit operators are easily controllable by the artist. In this figure, we demonstrate several variants of two types of operators: (top) the smooth-union from [Gourmel et al., 2013], as well as the (bottom) implicit-contact from [Vaillant et al., 2013a].

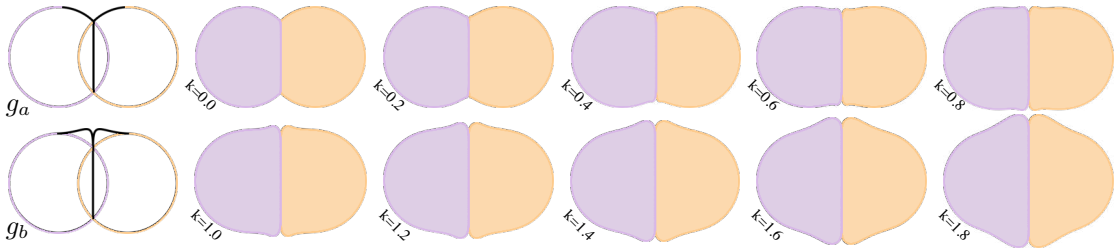


Figure 2.12: We define two symmetric contact operators through the sketches in the first column, generating g_a and g_b resulting in the composition visualized in the second column. By leveraging the consistency in the operator's parameterization, other operators can be generated as $g_k = (1-k)g_a + kg_b$; for $k \in (0,1)$ we obtain operators whose behavior is intuitively *interpolated* (top), while *extrapolation* can be obtained for values of k outside of this range (bottom).

$\{(0, *, \theta), (1, *, \theta), (*, 0, \theta), (*, 1, \theta)\}$ according to the boundary constraints in Equation 2.6.

Step 3 – Value propagation: the values assigned in the previous two sub-steps are diffused over the remaining portions of $\mathbb{D}$ by solving a bi-harmonic fairing optimization (where the values set in previous steps act as hard constraints).

Step 3* – Efficient value propagation: Step 3 is time-consuming, as its complexity is cubic in the linear resolution of the lattice. Fortunately, this is only necessary when the operator must be used in successive compositions. When only two objects need to be combined, as during the operator design process, only the lattice values surrounding the (0.5) surface are relevant, and the other ones can be safely disregarded. This observation drastically reduces the latency of the feedback loop, effectively enabling the interactive design discussed in Section 2.3.

2.7 Evaluation

We evaluate our work by verifying the *expressiveness* of our sketched operators, the *controllability* of results, as well as the possibility of *interpolating* between different operators.

Expressiveness. We empirically validate our approach by demonstrating a variety of effects. We collect all the fundamental types of implicit blending operator that have been proposed in the literature (to the best of our knowledge), and verify that our template can be optimized to express the same behavior. To this extent, Figure 2.21 reports a number of representative images from the literature, and the 2D sketch necessary to generate the desired operator. We visualize the optimized (deformed template) operator, and the result of its application on the 2D input geometry. We also show how all our custom operators extend to 3D in a completely straightforward fashion. Our sketches can be used to: (a) represent traditional CSG operations [Sabin, 1968]. (b) smoothly blend two primitives [Blinn, 1982] and [Ricci, 1973], (c) perform bulge-free blend [Gourmel et al., 2013]. (d) avoid premature-blending [Gourmel et al., 2013], (e) model bulge-on-contact between two objects [Cani, 1993], as well as (f,g) represent the partial-contact from implicit-skinning [Vaillant et al., 2013a]. In fact, the expressiveness of our operators go *beyond* the capabilities of those proposed in the literature, and allows us to extend the contact operators pioneered by [Cani, 1993], towards the representation of *asymmetric-contact*, without having to edit the input scalar fields; see Figure 2.10.

Controllability and interpolation. As our algorithm receives user sketches as input, the behavior of the operator is easily controllable. In Figure 2.11, we demonstrate how several variants of blending and contact operators can be faithfully reproduced by specifying the desired behavior with a simple 2D sketch. Another form of composition control can be obtained by *blending* two distinct composition operators. As illustrated in Figure 2.8, the topology of the template is consistent across all of our examples. This allows us to blend operators through simple linear interpolation/extrapolation of its control points; see Figure 2.12.

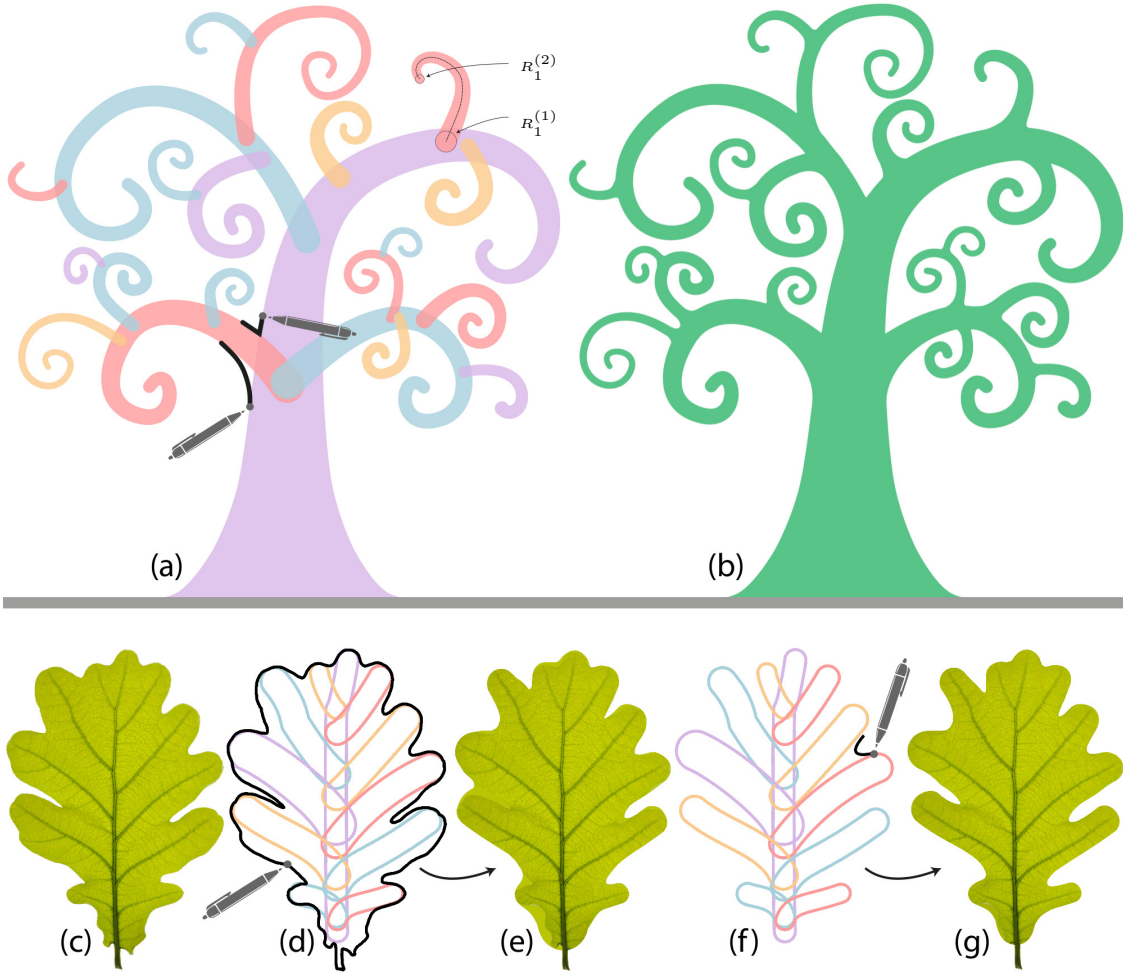


Figure 2.13: Modeling quasi-biological structures such as a procedural curly-tree (top) and an oak leaf (bottom). We show the input primitives and the artist sketches (a), as well as the composition result (b). The leaf design starts with the placement of a few skeletal branches, from which the leaf boundary (d) or a single sketch (f) can be used to design a corresponding blending (e,g).

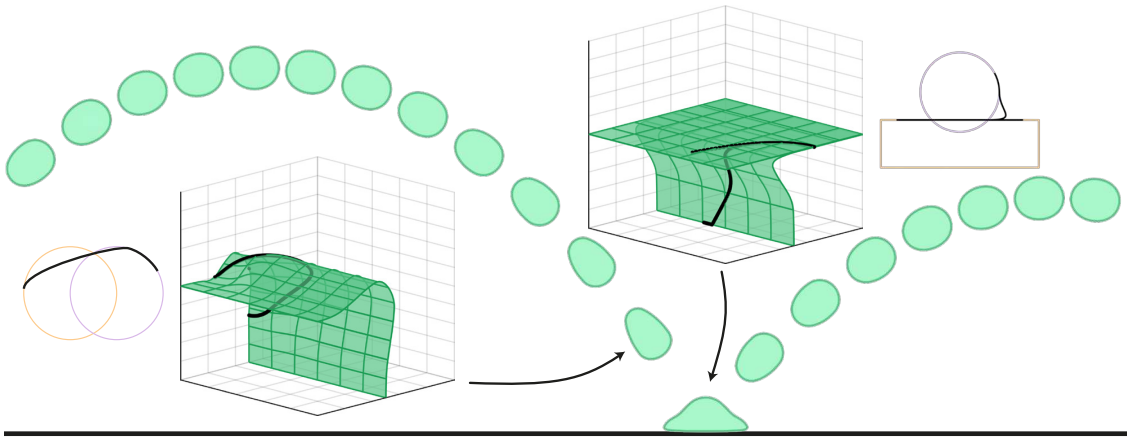


Figure 2.14: A rubber ball that elongates according to its velocity, and squishes when it comes into contact with the floor in an artist-defined fashion.

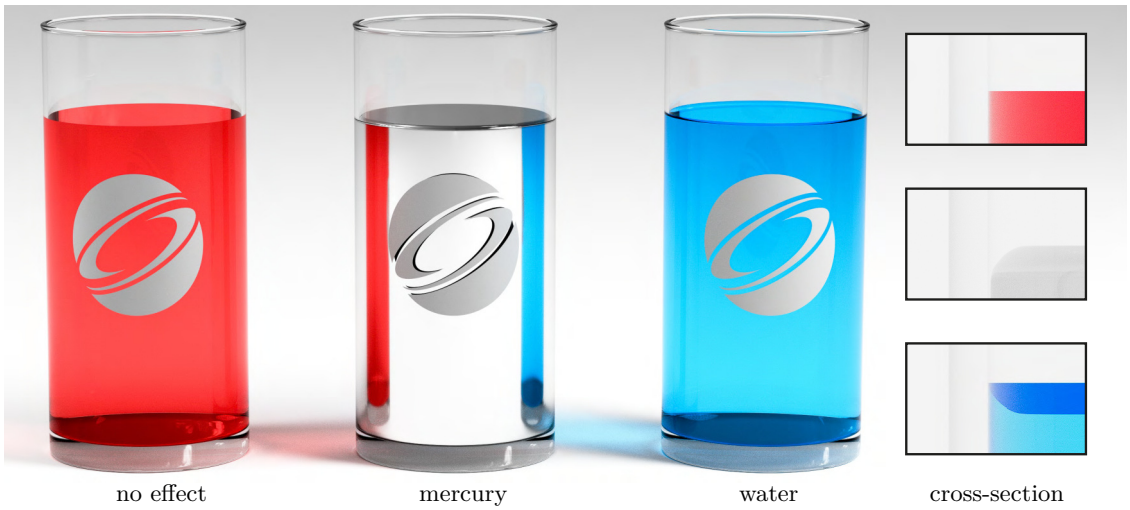


Figure 2.15: Applying implicit contact operators to the output of a fluid simulation produces a noticeable improvement in visual quality. This effect is efficient to apply, as many fluid solvers represent liquids with implicit functions. While our operator can apply to any geometry, we choose a simple and familiar scene that can be more easily appreciated.

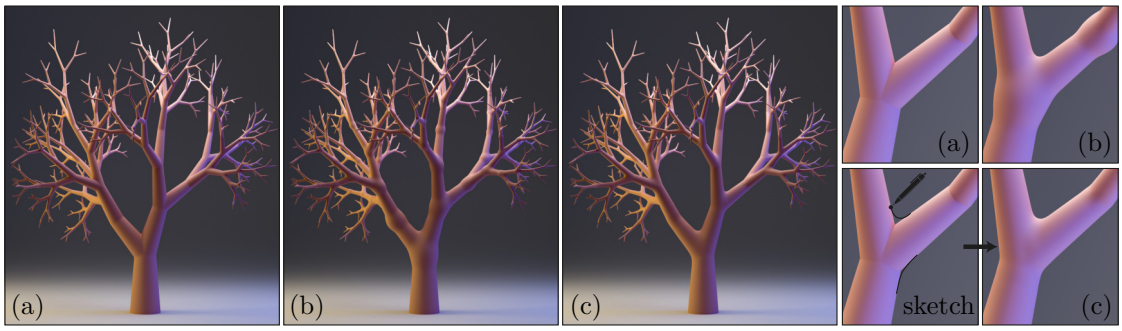
2.8 Applications

We present several example applications that benefit from the sketch-based generation of blending operators. The input sketches can either be drawn freely, or rotoscoped over annotated images.

Botanical modeling. Our curly tree shown in Figure 2.13-top is inspired by the images returned by the search query "curly tree". The placement and shape of primitives can be produced by any procedural grammar-based algorithm [Lienhard et al., 2017]. Our technique allows for the automatic generation of controllable smooth-blends connecting branches to each other. Two user strokes are sufficient to define the desired operator behavior, and the result is applied to a large quantity of compositions. In this case a single operator was used, but variability can be easily achieved by interpolating multiple variants (see Figure 2.12) with random weights; a 3D example of this process is visualized in Figure 2.17. In Figure 2.13-bottom, we apply our

simple union hydrophobic operator hydrophilic operator

Figure 2.16: The *same* sphere lying in proximity of a plane can produce contact deformations simulating both hydrophobic and hydrophilic deformations according to the type of asymmetric blending designed by the artist.



simple union *metaball* blend sketched blend

Figure 2.17: (a) A procedural tree generated by [Lienhard et al., 2017] as a collection of sphere-meshes [Tkach et al., 2016]. (b) a traditional blending operator results in unwanted bulges (e.g. Blender's metaballs), (c) our blending operator designed via a simple 2D sketch is propagated to all branches in the hierarchy.

sketch-driven operators to the procedural generation of botanical leaves. The skeletal structure of the leaf is generated via an appropriate grammar, as in the tree example. In this setup we experiment with two different sketches. In Figure 2.13d, the sketch is the entire boundary of the leaf from Figure 2.13c, while in Figure 2.13f the user has sketched the operator himself. The optimized operators result in blended geometry closely mimicking the geometry of natural leaves (we superimpose the original texture to make this more apparent).

Approximating contact behavior. In Figure 2.14, we approximate the behavior of a rubber ball in motion through the design of two operators. The first distorts the shape of the ball, modeled via a "ghost" primitive whose position is determined according to the velocity vector. The second operator captures the contact behavior between the projectile and the deformable wall. In our *stop-motion* illustration the distortion was interpolated, while the contact behavior was extrapolated. Asymmetric contact operators can also be used to approximate the fine-scale behavior of liquids when they come in contact with surfaces coated with different materials; see Figure 2.16. Obtaining such effects through fluid dynamic simulation is difficult as it requires a volume-preserving solver and careful implementation of boundary conditions; [Wang et al., 2005]. Such effects, due primarily to surface tension, can also be approximated by evaluating a curvature flow on surfaces [Thürey et al., 2010]. Our approach approximates these effects without the need for complex physical simulation, and allows their application in a lightweight fashion as a post-processing step; see Figure 2.15.

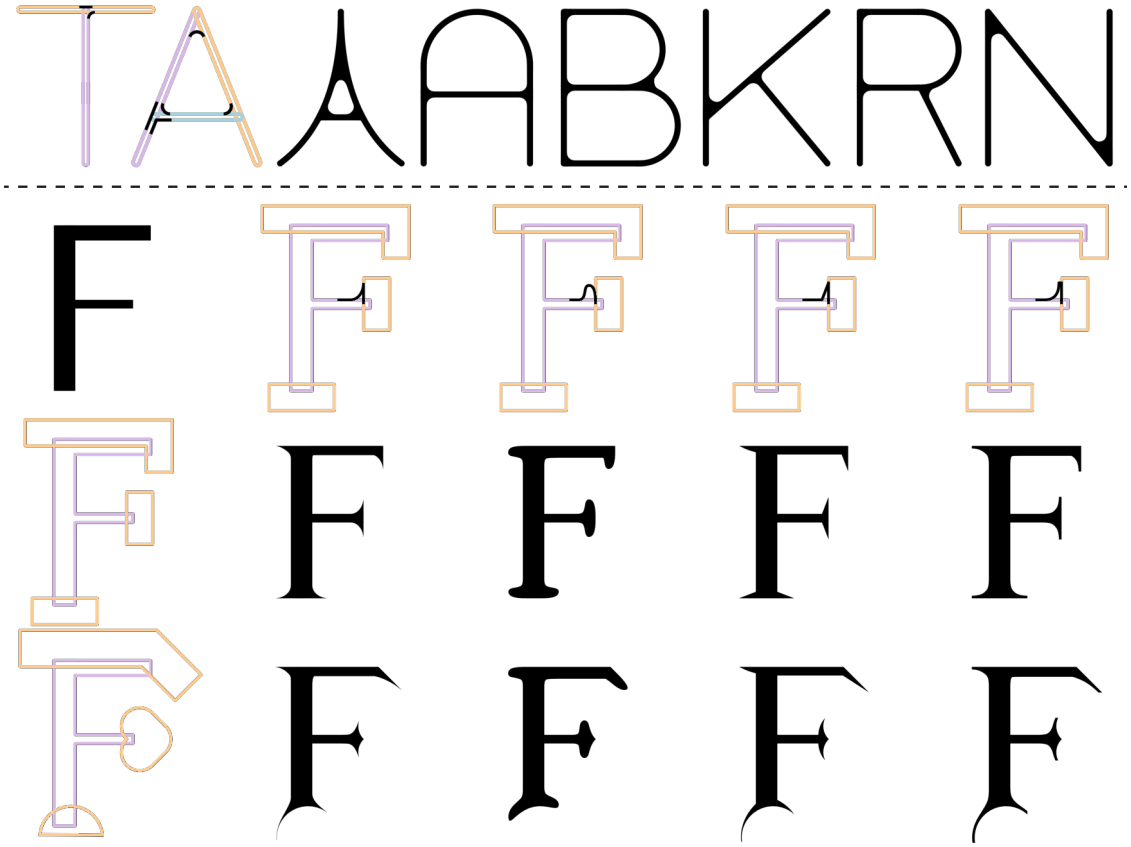


Figure 2.18: (top) Multiple sketches on different primitives are used to synthesize a blending operator that can be used to produce font variations as well as an entire "infinite resolution" font family. (bottom) Sketching negative blends to apply a "serification" effect.

Implicit vector font design. Another application is the creation of font libraries, in particular as our method allows a user to explore the design space *interactively*; see Figure 2.18. In this example, we use multiple sketches on the letter `T` and `A` to derive one operator, reproducing the desired behavior for orthogonal and non-orthogonal configurations. Again, the composition operators is derived from a few exemplars and then applied to all other cases. The user can explore different variants for a character, and export the blending operator to create a self-consistent font library. An example showcasing the extensibility of font design to three dimensions is illustrated in Figure 2.1. In typography, another common task is the application of *serifs* to characters. In this example, we apply our composition operators to this task. To achieve this, the yellow primitive *removes* ink from the pink primitive, with a difference operation (see Equation 2.4). This setup provides the user with a fast and easy method to prototype different serif styles, as illustrated. Because our operator kernel P is an algebraic surface, the operator g can be generated at any scale; hence, as long as the input primitives can also be expressed in algebraic form (as in the examples shown), our fonts can be synthesized at any desired resolution.

Figure 2.19: The leaf geometry from Figure 2.13e is lifted to 3D, and combined with a set of randomly placed spheres through the hydrophobic contact operator from Figure 2.16.

2.9 Conclusions

Implicit models have recently re-emerged as a powerful modeling technique with the introduction of gradient-based operators [Gourmel et al., 2013], solving several problems that were often regarded as intrinsic to implicit modelling. While more expressive the new operators are difficult to control. In this work we have introduced a solution based on an inverse approach which allows *sketch-based* design of new operators. The designer directly describes the desired effect of the operator on the resulting surface, without having to understand the mathematics of the operator space. We have shown a number of practical applications demonstrating that our approach enables a designer to quickly take advantage of the powerful nature of implicit modelling. In addition, our approach also serves as a conceptual tool to investigate the space of possible gradient-based operators. Using it, we were able to create a new class of non-commutative operators; see Figure 2.10.

Limitations and future work. The *expressiveness* of the gradient-based implicit operators, defined as in Equation 2.1, while being superior to the traditional 2D operators, is still insufficient for some cases. For example, regardless of how it is designed, no such operator can reproduce the shape sketched in Figure 2.20a. This situation arises when different sketches define constraints that are conflicting in the operator domain. For instance, in Figure 2.20a, the top left and top right fields are mirrored by vertical symmetry. In that case, pairs of symmetric points in the Euclidean space define the same point in the operator space. If two different sketches are defined in each side, as the sharp angle in the top left and the blend in the top right, the result of our fitting process provides an "average" shape that can be unsatisfactory when the input sketches are too different. In the future we intend to further generalize blending operators, to augment the range of achievable effects; e.g. anisotropic operators. We have shown that operators based on a 3D domain can be designed with 2D sketches; the same approach could be applied to higher dimensional operators, which have additional parameters.

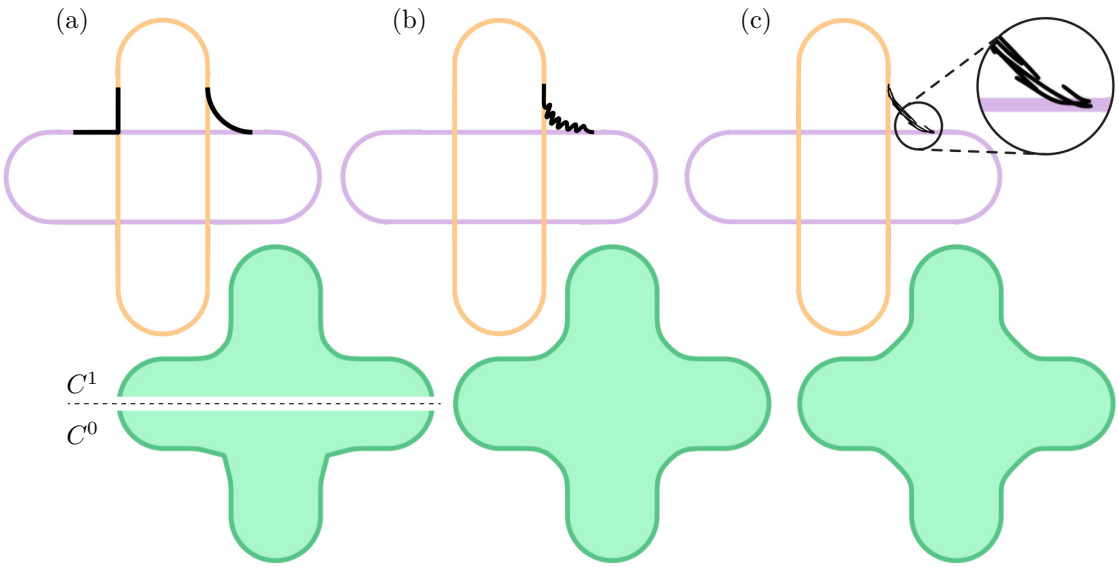


Figure 2.20: (a) The sketches result in conflicting samples in the operator domain $\mathbb{D}$, revealing a limitation in expressiveness of gradient-based operators; the user can nonetheless opt for an operator exhibiting either C^0 or C^1 continuity to approximate its input. (b) The smooth nature of the operators is unsuitable to capture dense high-frequency information, which is simply ignored as noise by the optimization. (c) On the other hand, this could be thought of as an advantage, as an artist can draw multiple *noisy* strokes, and obtain an operator fitting them well.

The *applicability* of our custom operators is not limitless either. While they can be applied to any implicit model, the fall-off functions are assumed to be roughly similar, otherwise the shape produced by the operator can diverge arbitrarily from the user's sketch; e.g. if one of the two support radii R_2 is vastly different.

Another inherited limitation of gradient-based operators is that they can generate artifacts where scalar fields exhibit gradient discontinuities. This rarely occurs in practice.

A different issue is that our operators, in general, lack associativity. Associative binary operators are desirable because they constitute a natural definition of operators working on any number of operands. An interesting open question is whether the input primitives could be inferred along with the operators. Finally, it would be interesting to handle a sketch on more than two overlapping primitives.

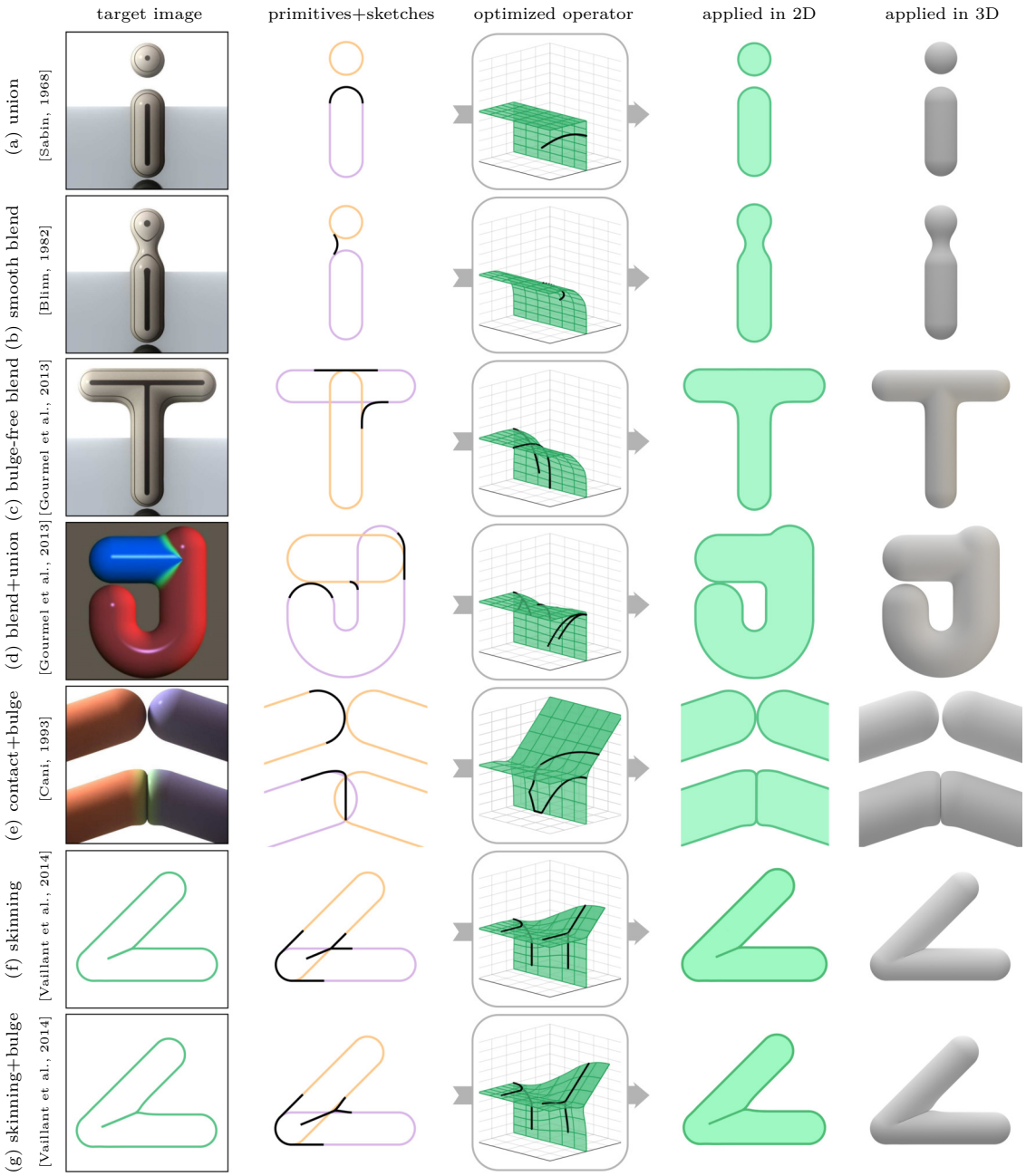


Figure 2.21: Our sketch-based construction method can be used to effortlessly produce every gradient blend operator proposed in past literature (pioneering citation is indicated). Starting from images depicting the final results of each operator (leftmost column), we simply sketched over 2D contour (second column). From these sketches, our system *reverse-engineers* the operator (central column). This can be applied (in 2D or 3D alike, rightmost columns) to reproduce the same results. Our target images are courtesy of [Gourmel et al., 2013] and [Vaillant et al., 2014].

Chapter 3

Simulation with Primitives

Figure 3.1: (left) We compactly model muscles as a collection of generalized rods, where volume conservation is expressed by a radius function defined on curve's vertices – vis sphere's radii. (middle) In the limit, this discretization represents the smooth rods illustrated here, for which physical constraints due to volume invariance can be expressed analytically. (right) The rods create a subspace on which physics is solved, and its effects later propagated to the muscle mesh via linear blend skinning; please see the animation in our **supplemental video**. The "Max" anatomical model is courtesy of ZIVA Dynamics.

Abstract

We extend the formulation of position-based rods to include elastic *volumetric* deformations. We achieve this by introducing an additional degree of freedom per vertex – isotropic scale (and its velocity). Including scale enriches the space of possible deformations, allowing the simulation of volumetric effects, such as a reduction in cross-sectional area when a rod is stretched. We rigorously derive the continuous formulation of its elastic energy potentials, and hence its associated position-based dynamics (PBD) updates to realize this model, enabling the simulation of up to 26000 DOFs at 140 Hz in our GPU implementation. We further show how rods can provide a compact alternative to tetrahedral meshes for the representation of complex muscle deformations, as well as providing a convenient representation for collision detection. This is achieved by modeling a muscle as a *bundle* of rods, for which we also introduce a technique to automatically convert a muscle surface mesh into a rods-bundle. Finally, we show how rods and/or bundles can be skinned to a surface mesh to drive its deformation, resulting in an alternative to cages for real-time

volumetric deformation. The source code of our physics engine will be openly available upon publication.

3.1 Introduction

In recent years, the computer graphics community has invested exceptional efforts in adapting the (non real-time) physical simulation algorithms at the core of cinematic special effects (e.g.: [Ziva Dynamics]) to the realm of (real-time) interactive applications (e.g.: games, AR/VR). Many of these advancements have been possible thanks to a new class of physics solver, pioneered by Müller et al. [2007], realized on top of *Verlet*-class integrators [Bender et al., 2015]. These *position-based* solvers are capable of elegantly modeling constrained Newtonian dynamics, including rigid-body, cloth, ropes, rods and fluids in a unified framework. A primary example is the NVIDIA FLEX system [Macklin et al., 2014], capable of modeling complex and varied physical phenomena in real-time by leveraging modern GPU hardware.

Rods with volume. Within this technological landscape, of particular relevance to our work is the modeling of elastic "rods" [Spillmann and Teschner, 2007; Umetani et al., 2014; Kugelstadt and Schömer, 2016]. These models extend "ropes" by augmenting each segment composing the rod with an orthogonal coordinate frame, hence allowing the modeling of *torsion* on top of stretching/bending. Our VIPER rods extend these formulations by accounting for volume preservation, a phenomena not modeled by existing position-based rod models. Many interesting phenomena require this constraint (e.g. soft-bodies, fluids). For example, water is the largest constituent of most animal tissues ($\approx 80\%$ in muscles) hence modeling quasi-incompressible phenomena is of critical importance to achieve believable motion. We address this problem by adding a *per-vertex scaling* degree of freedom – a measure of the local rod cross-section – and optimizing for this quantity within the physics solve. Our rod segments are *hybrid* surface representations, they are *explicitly* parameterized by the position and scale/radius of their vertices, but their surface is defined *implicitly*. This hybrid structure makes them particularly well suited for efficient collision detection/resolution [Green, 2010].

Anatomical modeling. Such a physical model not only satisfies our fixation in efficiently simulating rubber bands, but has immediate applications towards the modeling of muscles. As illustrated in Figure 3.2, *striated skeletal muscles*[1] in human bodies can be represented as a collection of fibers surrounded by connective tissue (i.e. fasciae). Simulating these muscle types efficiently is a fundamental problem, as they represent from 36% to 42% of the average human body mass. In this paper, we propose to efficiently model muscles as a structured collection of volume-preserving rods. This new model can also be interpreted as a generalization of the static volumetric primitives in *Implicit Skinning* [Vaillant et al., 2013b], where skin can then be efficiently modeled as a triangular mesh sliding on an implicit iso-surface defined by our fibers. The VIPER primitive is designed to be integrated with other existing components to produce a complete character representation. These include representations of the skeleton, fat, skin, etc.

[1] From now on, we will refer to "striated skeletal muscle" simply as "muscles".

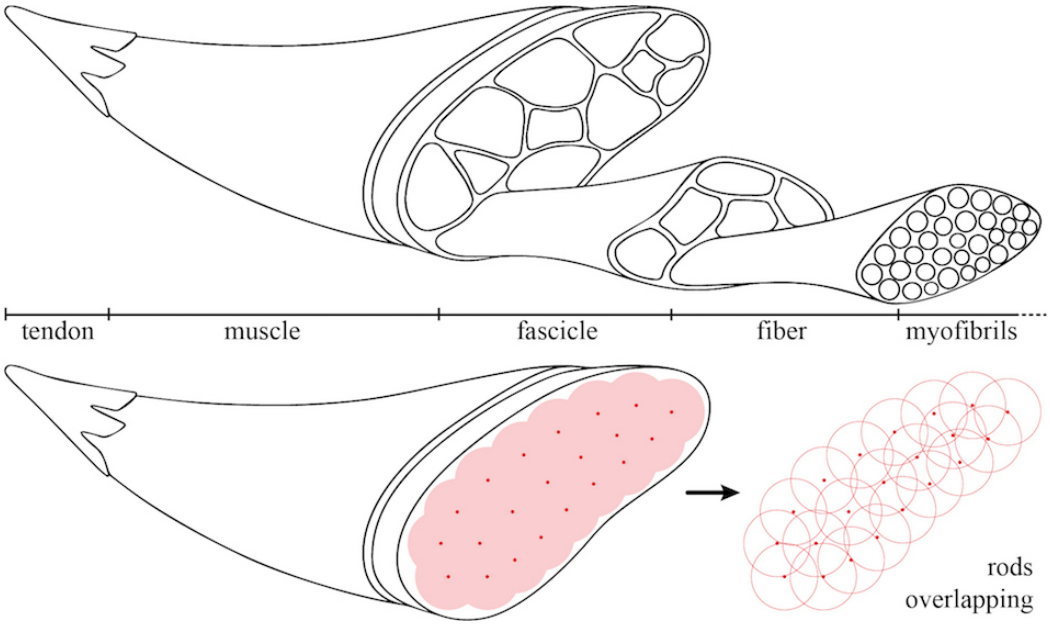


Figure 3.2: (top) Skeletal striated muscle as a collections of nested fascicles, fibres, and myofibrils; base image courtesy of [Lee et al., 2010]. (bottom) We abstract the fascicles as a collection of rods. These can be overlapping, and their rest-pose structure is controlled by *shape-matching* constraints.

Figure 3.3: Bio-mechanically accurate simulation of volumetric anatomical structure is the most effective way to simulate secondary motions (e.g. skin sliding on muscles) and deliver true realistic appearance to dynamic virtual characters; image courtesy of [Ziva Dynamics].

Volumetric simulation. In the industry, volume-preserving simulation is typically performed by discretizing the interior of the object with tetrahedra or using an approximating cage. However, to the best of our knowledge, even modestly sized models require a lengthy preprocessing (e.g. a large scale eigen-decomposition for computing the deformation modes; see [Barbič and James, 2005]) before real-time simulation becomes possible. For example, while visually striking, computing the simulation in Figure 3.3 requires a two-pass optimization (① muscle, ② skin). ① Muscles are discretized with $51k$ tets sharing $21k$ vertices, and each of their 4 steps of simulation requires 3.2 seconds. ② Skin ($78k$ vertices) is simulated as cloth layered on top of fat (having $68k$ tets sharing $18k$ vertices), where each of the necessary 4 substeps of simulation requires ≈ 35 seconds of compute. Overall, this cumulates to ≈ 2.5 minutes of compute/frame. Offline simulation can be exploited to learn sub-spaces, which then enables dynamic deformations in real-time; see [Xu and Barbič, 2016]. However, once learnt, the dynamic behavior of the model is "baked". Clearly, this is an obstacle towards the ultimate goal of truly interactive physics simulation, and, consequently, interactive modeling. The VIPER primitive not only allows the real-time volumetric simulation of complex anatomical structures, but also provides a viable alternative to cages as a compact control structure for soft-body deformations.

Automatic fiber-bundle modeling. Rod-based representations of muscle fascia are not commonly available – typical asset databases contain tetrahedral mesh models instead. While artists sometimes model main characters to the level of interior muscles, this effort is expensive and not justified for background characters. Thus we also introduce a technique to convert existing assets with minimal user intervention. Our solution builds fiber bundles by first creating a set of slices through the muscle then performing an iterative optimization to interpolate these slices with a given number of rods such that they approximate the input surface well.

Contributions. Our fundamental contribution is the design of a novel real-time physics engine for soft-body dynamics. Our system presents several sub-contributions:

- We enrich position-based solvers by introducing a new volume-preserving cosserat rods model and associated constraints.
- We demonstrate how these primitives, when assembled into fiber-bundles, are effective in efficiently modeling muscles.
- We introduce a technique for conveniently creating fiber-bundles models from existing simulation assets.
- We introduce the use of VIPER rods as efficient deformation proxies for soft-body deformation.

3.2 Related Work

We overview the literature from different angles. We recap example-based modeling frameworks that are commonly used in digital production, as well as recent efforts towards the use of simulated anthropomorphic models. We also review methods that attempt to "emulate" them via geometric processes, and finally processes to calibrate a given model to a target.

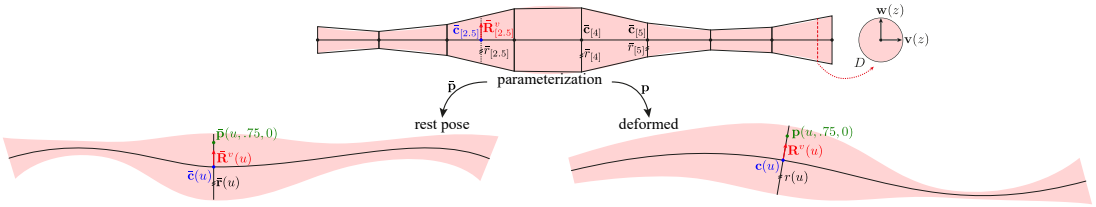


Figure 3.4: (top) The parameterization of a VIPER rod, and its discretization. (left) Its rest configuration, and (right) a deformed configuration.

Example-based deformation. Digital characters are often modeled via their skin (i.e. *skinning*), with no consideration of underlying volumetric structures, often resulting in non-physically realistic effects such as the *candy wrapper* problem of (LBS) Linear Blend Skinning [Jacobson et al., 2014a]. While these artifacts can be resolved [Kavan et al., 2007; Le and Hodgins, 2016], skinning solutions lack details such as tendons, muscle bulges, wrinkles, and volume preservation. Example-based approaches such as (PSD) *Pose-Space Deformation* [Lewis et al., 2000; Kurihara and Miyata, 2004] and *BlendShapes* [Lewis et al., 2014] interpolate artist-sculpted shapes to *emulate* all these effects. However, "*producing effects such as skin sliding over underlying structures, or the collision effects visible parts of the body press together, can require considerable skill and weeks to months of sculpting depending on the required quality*" [Yuen, 2018]. Such data-driven models can be learnt from measurements for both static [Loper et al., 2015], as well as dynamic [Pons-Moll et al., 2015] humans, but they hardly generalize outside of their corresponding training domains.

Physically-based anthropomorphic models. Physically-based simulation of characters has a long history [Terzopoulos and Waters, 1990; Scheepers et al., 1997; Sifakis et al., 2005] but, due to its high computational cost, it has only recently began to see practical use. For skeletal muscle deformation, Lee et al. [2010] provides an excellent overview of the field, in regards to which Saito et al. [2015], with its ability to reach *near-interactive runtime*, can be considered the state-of-the-art. Similarly to blendshape generation, training data can be exploited to generate efficient low-dimensional simulations [Xu and Barbič, 2016; Schumacher et al., 2012; Bouaziz et al., 2014], enabling physically-based digital characters in production settings [Clutterbuck and Jacobs, 2010; Ziva Dynamics]. A shortcoming of these methods is the requirement that the training set encompasses samples of all configurations of the object/character that that will be needed. Physically based approaches also permit a decomposition into *layers* – skin, muscle, fat, and bones – enabling appropriate algorithms to be used for each. Highly relevant to our work is the simulation of skin layered over volumetric primitives pioneered by Li et al. [2013], and its realization in commercial software [Vital Mechanics], as well as other existing research [Saito and Yuen, 2017], industry [Ziva Dynamics], and proprietary solutions [Clutterbuck and Jacobs, 2010] to this complementary problem. Recently, Romeo et al. [2018] proposed the use of PBD for muscle simulation, but its ≈40s/frame of processing time makes it unsuitable to interactive applications.

Non physically-based anthropomorphic models. Recent efforts have been made to create alternative representations for sub-skin volumetric models (i.e. representations of muscle, fat, and bones). For example, Maya Muscle [Comet, 2011] represents muscles via NURBS that drive the skin via LBS, whose volume is artist-driven in a PSD fashion. However, due to the *explicit* nature of NURBS, collisions are expensive, hence the performance of the system does not scale well in

the complexity of the model. Rather than driving skin via LBS, *Implicit Skinning* [Vaillant et al., 2013b] lets it slide on top of implicit surfaces via optimization. These surfaces are defined by blending components that abstract entire body parts (i.e. union of bone, muscle, and fat). In contrast to our work, note that this model is purely kinematic (i.e. no physics). Another relevant class of methods simplifies anthropomorphic components even further. For example representing an entire arm as two *tapered capsules* is advantageous for arm vs. cloth collision detection [Muller, 2008]. Sphere-Meshes generalize these representations, and have recently been used to approximate geometry [Thiery et al., 2013], and track its movement in real-time [Tkach et al., 2016].

Calibrating anthropomorphic (volumetric) models. Ali et al. [2013] pioneered the transfer of anatomical structures from a template to a target human via approximate deformation models of soft tissues, and Zhu et al. [2015] calibrated these models from a set of RGBD images capturing a human in motion. Analogously to these method, physically inspired models [Saito et al., 2015] can be calibrated to a set of 3D surface scans [Kadleček et al., 2016]. Of particular relevance to our method is the fiber estimation technique pioneered by Choi and Blemker [2013] employed in [Saito et al., 2015]. While Saito et al. [2015] is interested in deriving the anisotropic deformation frame, we require an explicit decomposition of the muscle in fiber bundles.

3.3 Generalized Rod Parameterization

Our physical model of Cosserat rods consists of a smooth parametric curve in 3D space $\mathbf{c}(z) : [z_0, z_1] \rightarrow \mathbb{R}^3$. An orthogonal frame $\mathbf{D}(z) \in \mathbb{R}^{3\times3}$ is attached to every point $\mathbf{c}(z) \in \mathbb{R}^3$ on the curve. The orthogonal frame $\mathbf{D}(z) = s(z)\mathbf{R}(z)$ is a combination of a uniform scale $s(u)$, and an orthonormal matrix $\mathbf{R}(z) = [\mathbf{u}(z), \mathbf{v}(z), \mathbf{w}(z)]$; see Figure 3.4. Note that our model generalizes classical elastic rods [Spillmann and Teschner, 2007; Kugelstadt and Schömer, 2016], as in those models the scale is kept *constant* along the curve. As shown in Figure 3.4, any point in the parametrized volume of the rod can be transformed to the rest configuration by a function $\bar{\mathbf{p}}(x, y, z) : \mathbb{R}^3 \rightarrow \mathbb{R}^3$:

$$\bar{\mathbf{p}}(x, y, z) \equiv \bar{\mathbf{p}}(\mathbf{q}, z) = \bar{\mathbf{c}}(z) + \bar{s}(z)\bar{\mathbf{R}}(z)\mathbf{q} \tag{3.1}$$

where $\mathbf{q} = [x, y, 0]^T$ is a point on a disc $D(z)$ of radius $r(z)$, aligned with the xy plane, and centered at its center of mass. Similarly, a second function maps the rod from parameterization to its deformed configuration:

$$\mathbf{p}(x, y, z) \equiv \mathbf{p}(\mathbf{q}, z) = \mathbf{c}(z) + s(z)\mathbf{R}(z)\mathbf{q} \tag{3.2}$$

3.4 Variational Implicit Euler Solver

Our solver is based on the variational form of implicit Euler integration [Martin et al., 2011]. The physical model evolves through a discrete set of time samples, with simulation step size h. At time t the deformed volume is defined as $\mathbf{p}_t(x, y, z)$ and the velocity as $\dot{\mathbf{p}}_t(x, y, z)$. The rest pose is defined as $\bar{\mathbf{p}}(x, y, z) = \mathbf{p}_0(x, y, z)$. The mass m is assumed to be uniform over the rod. The sum of the external forces is denoted as $\mathbf{f}_{\text{ext}}(x, y, z)$. We will now drop the indexing (x, y, z) whenever

possible to improve readability. We consider position dependent internal forces such that the sum of the internal forces is

$$\mathbf{f}_{\text{int}}(x,y,z) = -\tfrac{1}{2}\textstyle\sum_i \nabla_{\mathbf{p}} \|\mathbf{W}_i(\mathbf{p},\bar{\mathbf{p}})\|^2_{\mathbf{K}_i}, \tag{3.3}$$

where $\mathbf{W}_i(\mathbf{p},\bar{\mathbf{p}})$ is a potential energy function, and $\mathbf{K}_i$ is a stiffness matrix which we assume to be uniform over the rod, and the notation $\|\mathbf{x}\|^2_{\mathbf{A}}$ means $\mathbf{x}^T\mathbf{A}\mathbf{x}$. We can then write implicit Euler as an optimization describing the compromise between an inertia potential and the elastic potentials:

$$\min_{\{\mathbf{c}_t, s_t, \mathbf{R}_t\}} \int_{z_0}^{z_1} \iint_{D(z)} \underbrace{\tfrac{m}{2h^2}\|\mathbf{p}_t - \hat{\mathbf{p}}_t\|_2^2}_{inertia} + \underbrace{\tfrac{1}{2}\textstyle\sum_i \|\mathbf{W}_i(\mathbf{p}_t,\bar{\mathbf{p}})\|^2_{\mathbf{K}_i}}_{elastic} dx\, dy\, dz \tag{3.4}$$

With $\hat{\mathbf{p}}_t$ we indicate the *inertial prediction* for $\mathbf{p}_t$, i.e., its next position in absence of internal forces:

$$\hat{\mathbf{p}}_t = \mathbf{p}_{t-1} + h\dot{\mathbf{p}}_{t-1} + \tfrac{h^2}{m}\mathbf{f}_{\text{ext}}, \tag{3.5}$$

where $\dot{\mathbf{p}}(\mathbf{q},z) = \dot{\mathbf{c}}(z) + \left(\dot{s}(z)\mathbf{R}(z) + s(z)\dot{\mathbf{R}}(z)\right)\mathbf{q}$.

Discretization. We discretize the curve in the parametrized domain using a set of $m+1$ points $\{z_{[0]},\ldots,z_{[m]}\}$ connected using m *piecewise linear elements* of length $\{l_{[1]},\ldots,l_{[m]}\}$; see Figure 3.4. We can approximate the curve integral by integrating over these piecewise linear elements using the midpoint rule. For the integration we also define a set of m midpoints $\{z_{[.5]},\ldots,z_{[m-.5]}\}$. A point on a midpoint cross section is then parametrized as:

$$\mathbf{p}(\mathbf{q}, z_{[j-.5]}) = \mathbf{c}(z_{[j-.5]}) + s(z_{[j-.5]})\mathbf{R}(z_{[j-.5]})\mathbf{q} \tag{3.6}$$

$$\mathbf{c}(z_{[j-.5]}) \equiv \tfrac{1}{2}\left[\mathbf{c}(z_{[j-1]}) + \mathbf{c}(z_{[j]})\right] \tag{3.7}$$

$$s(z_{[j-.5]}) \equiv \tfrac{1}{2}\left[s(z_{[j-1]}) + s(z_{[j]})\right] \tag{3.8}$$

This is similar to the staggered grid discretization of previous work [Spillmann and Teschner, 2007; Grégoire and Schömer, 2006], where the frames $\mathbf{R}$ are stored at the *midpoints*. Contrary to previous approaches, our model also has a scale that we store at the *endpoints* of the linear elements. We can now rewrite Equation 3.4 as:

$$\min_{\{\mathbf{c}_t, s_t, \mathbf{R}_t\}} \sum_{j=1}^{m} l_{[j]} \iint_{D(z_{[j-.5]})} \tfrac{m}{2h^2}\|\mathbf{p}_t - \hat{\mathbf{p}}_t\|_2^2 + \tfrac{1}{2}\textstyle\sum_i \|\mathbf{W}_i(\mathbf{p}_t,\bar{\mathbf{p}})\|^2_{\mathbf{K}_i} dx\, dy \tag{3.9}$$

3.5 Elastic Potentials

In this section we detail the elastic potentials used to simulate our volume preserving rods.

Strain potential. We define the strain at a midpoint as

$$\mathrm{E}_{\text{strain}}(z_{[j-.5]}) = \iint_{D(z_{[j-.5]})} \|\mathbf{R}^T\nabla\mathbf{p} - \bar{\mathbf{R}}^T\nabla\bar{\mathbf{p}}\|^2_{\mathbf{K}_s} dx\, dy, \tag{3.10}$$

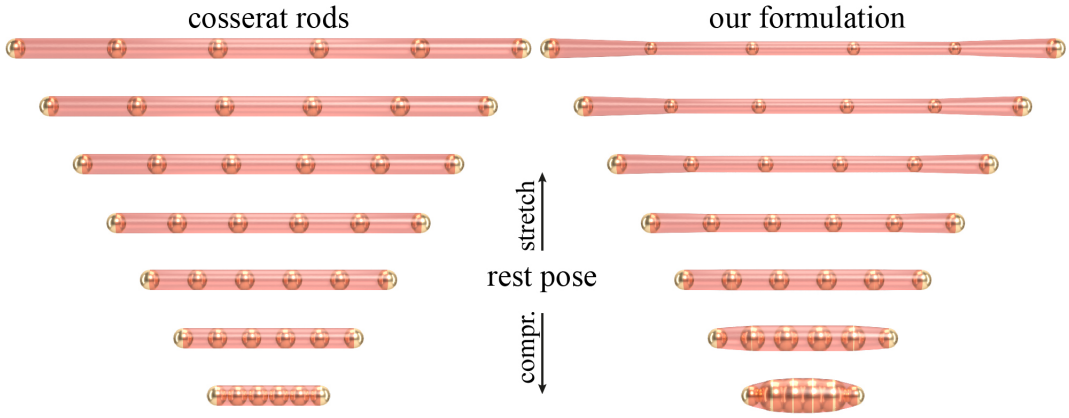


Figure 3.5: **Static behavior** – We compare the deformation of a standard cosserat rod to our volumetric invariant version. Our additional degrees of freedom allow us to model the buckling (resp. bulging) caused by the stretching (resp. compression) of the rod.

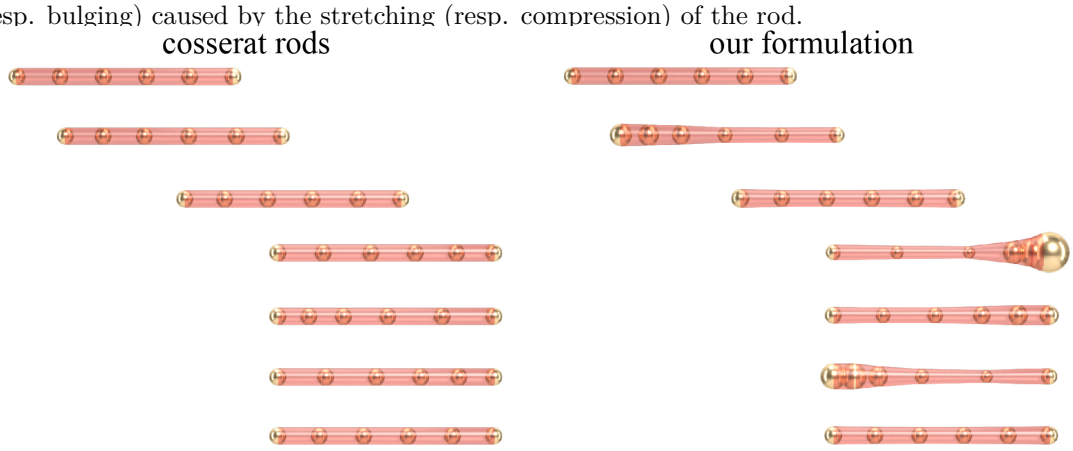


Figure 3.6: **Dynamic behavior** – Our generalized rods do not only capture static volumetric deformations, but the scale's velocity allows us to model volume dynamics. In this example, note the rod length is unchanged, but our formulation models a volumetric shockwave travelling through the rod.

where $\mathbf{K}_s = \left[k_s^x \mathbf{e}^x \; k_s^y \mathbf{e}^y \; k_s^z \mathbf{e}^z \right]$ is a diagonal stiffness matrix and $[\mathbf{e}^x \mathbf{e}^y \mathbf{e}^z]$ is the standard basis. $\nabla \mathbf{p}$ and $\nabla \bar{\mathbf{p}}$ denote the deformation gradients, i.e., the Jacobian matrices of the deformation functions [Sifakis and Barbic, 2012]. As $\mathbf{p}$ and $\bar{\mathbf{p}}$ map $\mathbb{R}^3$ to $\mathbb{R}^3$, the Jacobian matrices are 3×3. The Jacobians are not rotational invariant so we rotate them back to the parametrization domain to be able to compare them on a common ground. As explained in Appendix A.1 integrating the strain energy (3.10) leads to

$$\begin{aligned} \mathrm{E}_{\mathrm{strain}}(z_{[j-.5]}) &= \pi r^2 k_s^z \|\nabla_z \mathbf{c} - \mathbf{w}\bar{\mathbf{w}}^T \nabla_z \bar{\mathbf{c}}\|_2^2 && (3.11) \\ &+ \pi r^2 (k_s^x + k_s^y)(s - \bar{s})^2 && (3.12) \\ &+ \tfrac{1}{4}\pi r^4 (k_s^x + k_s^y)(\nabla_z s - \nabla_z \bar{s})^2 && (3.13) \\ &+ \|s\boldsymbol{\Omega} - \bar{s}\bar{\boldsymbol{\Omega}}\|_{\mathbf{H}_s}^2, && (3.14) \end{aligned}$$

where $\mathbf{H}_s = \left[\pi r^4 k_s^z \mathbf{e}^x \; \pi r^4 k_s^z \mathbf{e}^y \; \pi r^4 (k_s^x + k_s^y) \mathbf{e}^z \right]$ is the second moment of area of a disc scaled by the stiffness, and the Darboux vector [Kugelstadt and Schömer, 2016] is denoted by $\boldsymbol{\Omega} = [\Omega^u, \Omega^v, \Omega^w]^T$. Note that we retrieved similar energies in previous works augmented by our additional scale degree of freedom. The energies (3.11) and (3.12) respectively measure the stretch along the curve and the cross section, while (3.13) measures the variation of scale across sections, and (3.14) measures bending/twisting. Interestingly, Equation 3.13 can also be interpreted as a measure of surface stretch.

We use an additional energy measuring the second order variation of scale complementing (3.13) with a measure of surface *bending*

$$\mathrm{E}_{\mathrm{bending}}(z_{[j-.5]}) = \tfrac{1}{4}\pi r^4 \left(k_b^x + k_b^y\right) \left(\nabla_z^2 s - \nabla_z^2 \bar{s}\right)^2 \tag{3.15}$$

where $\nabla^2 = \nabla \cdot \nabla = \Delta$ is the Laplacian operator. Note that this energy can also been seen as an *approximation* of:

$$\iint\limits_{D(z_{[j-.5]})} \|\mathbf{R}^T \nabla^2 \mathbf{p} - \bar{\mathbf{R}}^T \nabla^2 \bar{\mathbf{p}}\|_{\mathbf{K}_b}^2 \, dxdy, \tag{3.16}$$

where $\mathbf{K}_b = \left[k_b^x \mathbf{e}^x \; k_b^y \mathbf{e}^y \; k_b^z \mathbf{e}^z \right]$. This energy compares the Laplacians of the deformation functions giving a second order measure of the deformation.

Volume potential. By denoting the determinant with $|\cdot|$, and stiffness by k_v, we define the volume preservation at midpoints as:

$$\mathrm{E}_{\mathrm{vol}}(z_{[j-.5]}) = \iint\limits_{D(z_{[j-.5]})} k_v \left(|\nabla \mathbf{p}| - |\nabla \bar{\mathbf{p}}|\right)^2 dx \, dy \tag{3.17}$$

Note that the determinant is rotational invariant, hence it is not necessary to rotate the Jacobians. As explained in Appendix A.1 integrating the volume energy (3.17) leads to:

$$\begin{aligned} \mathrm{E}_{\mathrm{vol}}(z_{[j-.5]}) &= \pi r^2 k_v \|s^2 \nabla_z \mathbf{c} - \bar{s}^2 \mathbf{w}\bar{\mathbf{w}}^T \nabla_z \bar{\mathbf{c}}\|_2^2 && (3.18) \\ &+ \tfrac{1}{2}\pi r^4 k_v \left(s^3 \Omega^u - \bar{s}^3 \bar{\Omega}^u\right)^2 && (3.19) \\ &+ \tfrac{1}{2}\pi r^4 k_v \left(s^3 \Omega^v - \bar{s}^3 \bar{\Omega}^v\right)^2. && (3.20) \end{aligned}$$

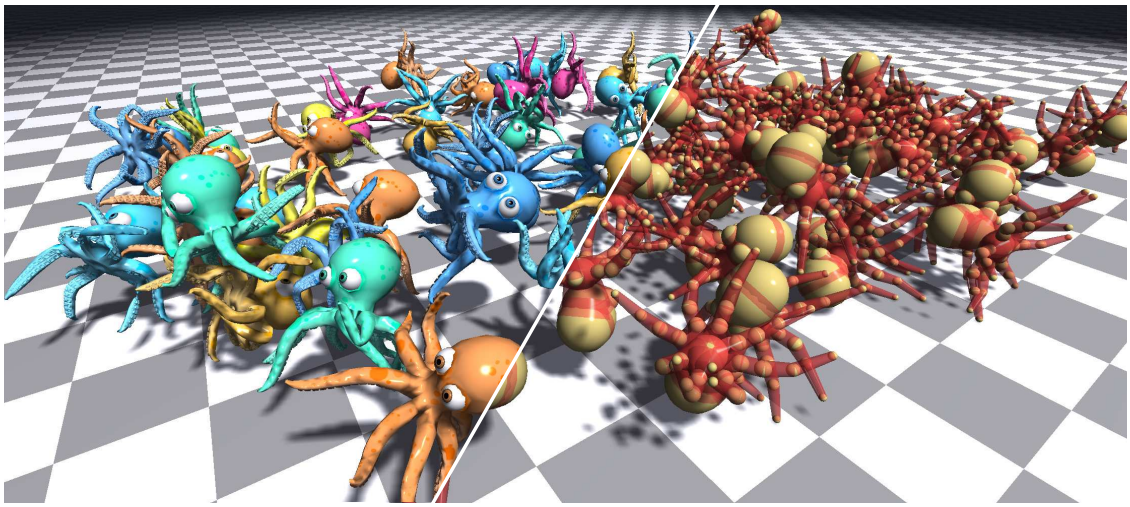


Figure 3.7: Being built on VIPER primitives, our physics engine can simulate soft-body deformation and dynamic interactions between hundreds of models in *real-time*. A peculiarity of our engine is that *both* collision and simulation are executed on the *same* geometry; see our video in the **additional material**.

3.6 Optimization

To approach the non-linear optimization problem in (3.9), we linearize the inertia and non-linear elastic terms, and then solve the optimization iteratively in a Gauss-Newton fashion. To warm start the optimization, we first compute a *prediction step* by ignoring the elastic potentials and by solely minimizing the inertia term – this simply provides an initial guess. We then compute a (set of) *correction steps* that also include the elastic potentials.

Prediction step. By denoting by $\boldsymbol{\theta}$ the angles parametrizing the rotation matrix and $\mathcal{I}$ is the second moment of area of a disc in world-space, the predictions for the different degrees of freedom of our model are computed as:

$$\hat{\mathbf{c}} = \mathbf{c} + h\dot{\mathbf{c}} + \tfrac{h^2}{\pi r^2 m}\boldsymbol{\xi}_{\text{ext}}, \qquad \text{(center prediction)} \tag{3.21}$$

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta} + h\dot{\boldsymbol{\theta}} + \tfrac{\mathcal{I}^{-1}h^2}{s^2 m}\boldsymbol{\tau}_{\text{ext}}, \qquad \text{(frame prediction)} \tag{3.22}$$

$$\hat{s} = s + h\dot{s} + \tfrac{2h^2}{\pi r^4 m}\boldsymbol{\gamma}_{\text{ext}}, \qquad \text{(scale prediction)} \tag{3.23}$$

where $\boldsymbol{\xi}_{\text{ext}}$ is the sum of the external forces which act on the disc, $\boldsymbol{\tau}_{\text{ext}}$ is the sum of the external torques, and $\boldsymbol{\gamma}_{\text{ext}}$ is a quantity which can be seen as the counterpart of the total external torque measuring the sum of the external forces projected on the position vectors; see Appendix A.2 for more details. Note that the center and frame predictions are similar to the rigid-body equations of motion for a stretched disc. On top of these equations, we get a scale prediction describing how the scale of the disc is affected by the velocity and the external forces.

Correction steps. We then compute a set of correction steps including both inertial/elastic terms. We define

$$\mathbf{X} = [\mathbf{c}_{[0]}^T, s_{[0]}, \boldsymbol{\theta}_{[.5]}^T, \mathbf{c}_{[1]}^T, s_{[1]}, \boldsymbol{\theta}_{[1.5]}^T, \cdots]^T \tag{3.24}$$

as the vector containing all the degrees of freedom, and $\boldsymbol{\lambda}$ as the vector of Lagrange multipliers. $\mathbf{K}$ is a block diagonal matrix containing the stiffness parameters multiplied by the length of the piecewise elements, $\mathbf{A}$ is a block diagonal matrix stacking the inertia weights multiplied by the length of the piecewise elements, and

$$\mathbf{W}(\mathbf{X}) = [\mathbf{W}_1(\mathbf{X}), \mathbf{W}_2(\mathbf{X}), \cdots]^T \tag{3.25}$$

stacks the potential energy functions. Denoting the iteration number with k, the the state is then updated as $\mathbf{X}^k = \mathbf{X}^{k-1} + \Delta\mathbf{X}$ and $\boldsymbol{\lambda}^k = \boldsymbol{\lambda}^{k-1} + \Delta\boldsymbol{\lambda}$, where, as derived in Appendix A.3:

$$\Delta\mathbf{X} = -h^2\mathbf{A}^{-1}\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\Delta\boldsymbol{\lambda}, \tag{3.26}$$

$$\Delta\boldsymbol{\lambda} = \left(h^2\|\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\|^2_{\mathbf{A}^{-1}} + \mathbf{K}^{-1}\right)^{-1}\mathbf{W}(\mathbf{X}^{k-1}) - \mathbf{K}^{-1}\boldsymbol{\lambda}^{k-1}. \tag{3.27}$$

For realtime performance we opt for using an iterative linear system solver such as block Jacobi or Gauss-Seidel. The update for the i-th constraint is:

$$\Delta\boldsymbol{\lambda}_i = \beta\left(h^2\|\mathbf{W}_i(\mathbf{X})^T\|^2_{\mathbf{A}^{-1}} + \mathbf{K}_i^{-1}\right)^{-1}\left(\mathbf{W}_i(\mathbf{X}) - \mathbf{K}_i^{-1}\boldsymbol{\lambda}_i\right) \tag{3.28}$$

where we dropped the superscripts to improve readability, and β is a relaxation parameter. Note that $\mathbf{A}$ being a block diagonal matrix with block of size at most 3×3, $\mathbf{A}^{-1}$ can be efficiently computed. Note that Equation 3.28 is a generalization of the XPBD update [Macklin et al., 2016] derived for our volume preserving rod model.

3.7 Real-time physics engine

We implemented our real-time solver on a GPU by leveraging the *Thrust* framework provided by the CUDA library, which we execute on a single NVIDIA GTX 1080 graphics card. In our engine, any volumetric object is modeled as a collection of tapered capsule primitives, or "pills" (for brevity), while the floor is represented as a simple halfplane constraint. During simulation, at each time step, we start by first animating kinematic objects (e.g. bones). We then compute the inertial predictions, and perform collision detection (Sec. 3.7.1) to generate collision resolution constraints (Sec. 3.7.2). We then solve for all constraints using a Jacobi solver: constraints are computed in parallel, and the resulting positional displacements are averaged out by a reduction. The transformations of VIPERs can then be skinned to any surface mesh model (Sec. 3.7.4).

Our model provides a viable alternative to cages as a compact control structure for soft-body deformations. We demonstrate this in Figure 3.7, where we rigged a simple octopus character using rods, and solve for collisions and soft-body deformations in real time. As shown in the accompanying video, our *non-optimized* prototype achieves real-time performance ($\approx 7ms$ sim, $\approx 6ms$ render) for scenes containing up to 100 octopuses, each rigged with 37 pills, whose deformation is skinned to a triangular surface mesh of $\approx 13k$ faces. Note how for robust collision detection we need far fewer pills than the volumetric particles used in Macklin et al. [2014].

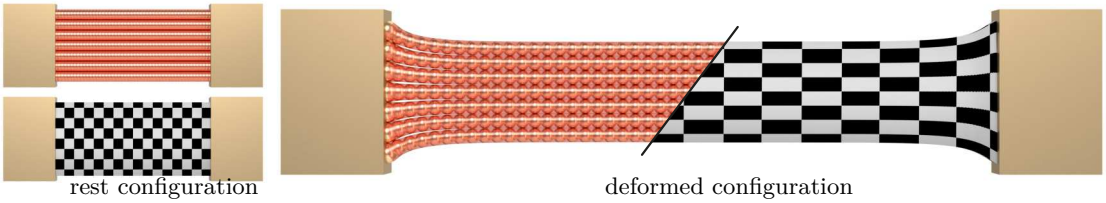


Figure 3.8: (left) An elastic band mesh and its VIPER discretization. (right) The VIPER simulation and their deformation "skinned" to the rest-pose mesh.

3.7.1 Collision detection

To detect collisions between physical primitives, we adopt the approach presented by Green [2010] popularized in the context of GPU particle fluid simulation. Towards this goal, we approximate each pill by its bounding sphere, and (conservatively) detect collisions to be later handled in the resolution loop detailed in Section 3.7.2. Collisions are detected with the assistance of a uniform grid with cell width chosen as the diameter of the largest bounding sphere, such that collisions between spheres centered in non-neighboring cells are impossible. Extending [Green, 2010], we also count the spheres in the neighboring cells of each particle, and use this information to construct *in parallel* a list of all potential collisions. This is in contrast to the original algorithm which for each sphere processes neighbors in series, and therefore degrades to a partially serial algorithm in cases where many particles fall into a single grid cell. Further details regarding this process are provided together with an executable 1D example in the form of a *Jupyter notebook* in our additional material.

3.7.2 Collision handling

In a generic physics engine one would implement collision detection/response between any pair of available proxies. For the sake of efficiency, in our framework we only tackle two collision proxies: (kinematic) *half-planes* and (dynamic) *pills*. Within the combinatorial set of collision pairs, the main challenge is pill-to-pill collisions. In what follows, we first compute the meta-parameters of the collisions, that are then resolved in the optimization via PBD constraints [Müller et al., 2007].

Collision metadata. Given two pills $\mathbf{P}_a$ and $\mathbf{P}_b$, each modeled as a pair of spheres, for example, $\mathbf{P}_a = \{(\mathbf{c}_1^a, r_1^a), (\mathbf{c}_2^a, r_2^a)\}$, the fundamental queries we need to answer are: ① is there a collision? ② what is the collision point/normal? ③ what is the inter-penetration amount? As typical in efficient collision resolution, we introduce a *single* PBD constraint modeling collision forces corresponding to the *largest* pill-to-pill inter-penetration. In our solution, we leverage the geometric structure of the problem: ① a pill can be interpreted as the union of infinitely many spheres whose position and radii are linearly interpolated between its endpoints; ② the largest inter-penetration corresponds to the inter-penetration between any pair of spheres, one in pill $\mathbf{P}_a$ and one in pill $\mathbf{P}_b$. By first defining the LERP operator as $\mathcal{L}(\mathbf{x}_1, \mathbf{x}_2, \gamma) \equiv (1-\gamma)\,\mathbf{x}_1 + \gamma\,\mathbf{x}_2$, the interpolated sphere $(\mathbf{c}(\gamma), r(\gamma))$ is derived by LERP'ing the endpoint quantities as $\mathbf{c}(\gamma) = \mathcal{L}(\mathbf{c}_1, \mathbf{c}_2, \gamma)$ and $r(\gamma) = \mathcal{L}(r_1, r_2, \gamma)$. The largest inter-penetration is then given by the solution of the *bivariate*

optimization problem:

$$\arg\min_{\alpha,\beta} \ \|\mathbf{c}_a(\alpha) - \mathbf{c}_b(\beta)\|_2 - (r_a(\alpha) + r_b(\beta)) \tag{3.29}$$

Because a closed-form operator $\Pi_b(\alpha)$ providing the barycentric coordinate of the closest-point projection of a point $\mathbf{c}_a(\alpha)$ onto the pill $\mathbf{P}_b$ is available (see Appendix A.4) we can further simplify this problem into a *scalar* optimization problem:

$$\arg\min_{\alpha} \ \|\mathbf{c}_a(\alpha) - \mathbf{c}_b(\Pi_b(\alpha))\|_2 - (r_a(\alpha) + r_b(\Pi_b(\alpha))) \tag{3.30}$$

which we solve by Dichotomous Search [Antoniou and Lu, 2007] with a fixed number of iterations (set to 10 in our engine).

Collision constraints. Having detected a collision between two pills $\mathbf{P}_a$ and $\mathbf{P}_b$, and having computed α^* (and hence $\beta^* = \Pi_b(\alpha^*)$) by solving (3.30), we can express a constraint that correlates radii and positions on the two pills in order to resolve the collision in a least squares sense:

$$E_{\text{collision}} = \left(\|\mathbf{c}_a(\alpha^*) - \mathbf{c}_b(\beta^*)\|_2 - r_a(\alpha^*) - r_b(\beta^*)\right)^2. \tag{3.31}$$

3.7.3 Scale-invariant shape matching – "Bundling"

To represent more complex geometry than individual rods, such as that in Figure 3.2, we can gather a collection of rods in a cross-section, and introduce a constraint to explain their joint deformation. We employ the assumption that muscle fibers in a muscle cross-section contract isotropically. Indexing the rods in a cross-section by i, our rod deformation model can be expressed as a similarity transform

$$\mathbf{T}_i = \begin{bmatrix} s_i\mathbf{R}_i & \mathbf{c}_i \\ \mathbf{0} & 1 \end{bmatrix}. \tag{3.32}$$

As illustrated in Figure 3.9, for each muscle cross-section we define a scale invariant shape-matching energy measuring the deviation of the current rod deformations $\mathbf{T}_i$ from a global similarity transform $\mathbf{T}^*$ of the rest deformations $\bar{\mathbf{T}}_i$ as

$$E_{\text{shape}} = \sum_i \|\mathbf{T}^*\bar{\mathbf{T}}_i - \mathbf{T}_i\|_2^2. \tag{3.33}$$

We treat this energy as a hard constraint by finding the optimal $\mathbf{T}^*$ and setting $\mathbf{T}_i = \mathbf{T}^*\bar{\mathbf{T}}_i$ as a post-processing step after few iterations. The optimal $\mathbf{T}^*$ can be computed following the derivation in [Umeyama, 1991]. The optimal rotation $\mathbf{R}^*$ can be found by solving

$$\mathbf{R}^* = \arg\min_{\mathbf{R}\in\text{SO}(3)} \ \|\mathbf{R} - \Sigma_i[s_i\mathbf{R}_i \quad \mathbf{c}_i - \boldsymbol{\mu}][\bar{s}_i\bar{\mathbf{R}}_i \quad \bar{\mathbf{c}}_i - \bar{\boldsymbol{\mu}}]^T\|_2^2, \tag{3.34}$$

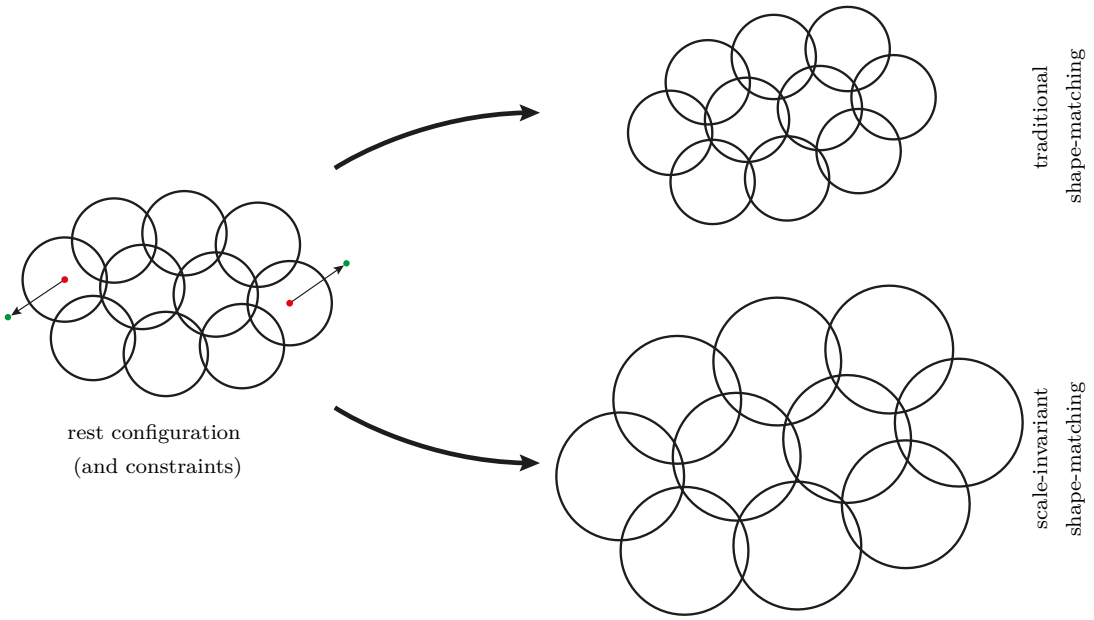


Figure 3.9: **Scale-invariant shape matching** – Shape matching can recover a rigidly transformed configuration, while our model allows for a null-space that includes uniform scale. We employ this model as we work under the assumption that muscle fibers in a muscle cross-section contract isotropically.

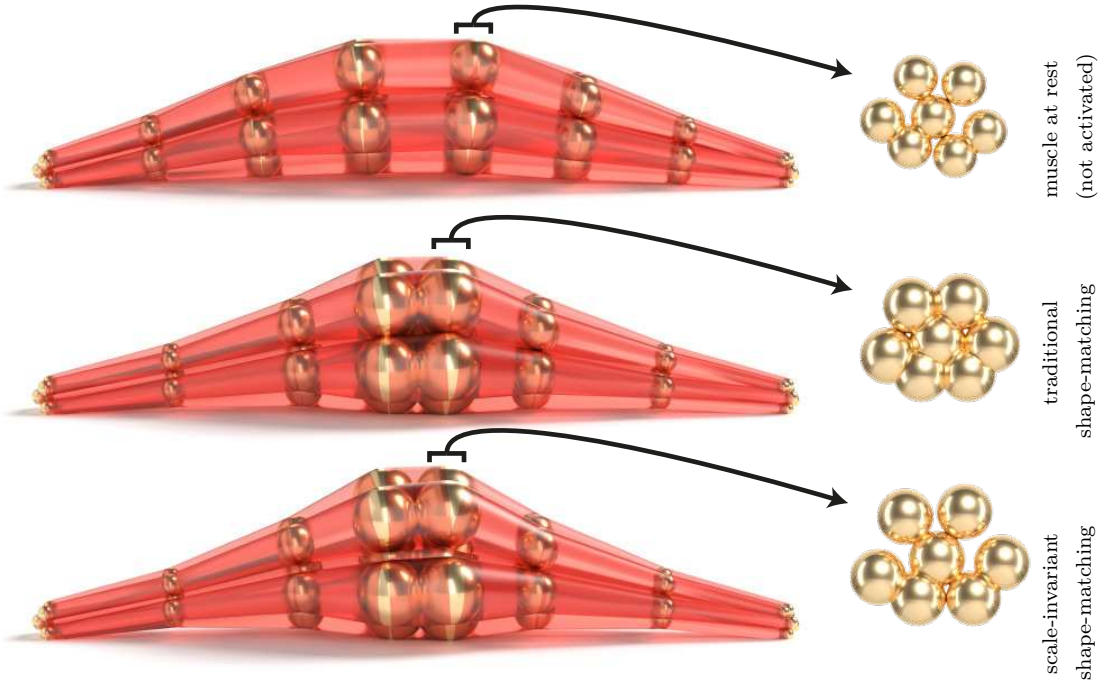


Figure 3.10: **Muscle contraction** – (top) Muscle at rest and its excitation (force shortening edge lengths in an area) with traditional shape-matching constraints (middle) vs our novel scaled shape-matching constraints (bottom).

where $\boldsymbol{\mu} = \frac{1}{n}\Sigma_i \mathbf{c}_i$. We compute the optimal rotation $\mathbf{R}^*$ using the robust approach presented in [Müller et al., 2016]. The optimal scale can be computed as

$$s^* = \frac{\sum_i \text{sum}(\mathbf{R}^*[\bar{s}_i\bar{\mathbf{R}}_i \quad \bar{\mathbf{c}}_i - \bar{\boldsymbol{\mu}}] \circ [s_i\mathbf{R}_i \quad \mathbf{c}_i - \boldsymbol{\mu}])}{\sum_i \text{sum}([\bar{s}_i\bar{\mathbf{R}}_i \quad \bar{\mathbf{c}}_i - \bar{\boldsymbol{\mu}}] \circ [\bar{s}_i\bar{\mathbf{R}}_i \quad \bar{\mathbf{c}}_i - \bar{\boldsymbol{\mu}}])}, \tag{3.35}$$

where sum$(\cdot)$ adds all entries of the matrix and $\circ$ is the Hadamard product. Finally, the optimal translation can be derived as

$$\mathbf{c}^* = \boldsymbol{\mu} - s^*\mathbf{R}^*\bar{\boldsymbol{\mu}}. \tag{3.36}$$

3.7.4 Skinning VIPERs to surface deformation

Our VIPER rods can also be employed as a volumetric *proxy* driving the deformation of a surface mesh; see Figure 3.1 and Figure 3.8. In particular, we blend the relative transformations between deformed and rest pose configurations via *linear blend skinning*. To achieve this, we utilize a modified version of the LBS weight computation by Thiery et al. [2013]. In that paper, weights were computed by fairing an initial assignment where each vertex was *fully* attached to the nearest element. Instead, we modify this assignment to begin with weights for each vertex that are proportional to the inverse square-distance from the vertex to the surface of each pill. This resolves cases where multiple surfaces are at equal distance, and the nearest neighbor is multiply-defined.

3.7.5 Energy implementation

The energies derived in Section 3.5 have been derived using continuous operators. To implement these energies we approximate $\nabla_z \mathbf{c}(z_{[j+.5]})$, $\nabla_z s(z_{[j+.5]})$, and $\nabla_z^2 s(z_{[j+.5]})$ using finite difference such that

$$\nabla_z \mathbf{c}(z_{[j+.5]}) = l_{[j]}^{-1}\left(\mathbf{c}(z_{[j+1]}) - \mathbf{c}(z_{[j]})\right), \tag{3.37}$$

$$\nabla_z s(z_{[j+.5]}) = l_{[j]}^{-1}\left(s(z_{[j+1]}) - s(z_{[j]})\right), \tag{3.38}$$

$$\begin{aligned}\nabla_z^2 s(z_{[j]}) = {} & l_{[j]}^{-1}\left(s(z_{[j+1]}) - s(z_{[j]})\right) \\ & - l_{[j-1]}^{-1}\left(s(z_{[j]}) - s(z_{[j-1]})\right).\end{aligned} \tag{3.39}$$

For simplicity of the derivations we used angles $\boldsymbol{\theta}$ to parametrize the rotation $\mathbf{R}$. However, our implementation uses quaternions. For small angles the corresponding quaternion is $\mathbf{Q} = [\frac{\boldsymbol{\theta}}{2}^T, 1]^T$. We also use quaternions to represent rotations and approximate the Darboux vector using

$$\boldsymbol{\Omega}(z_{[j]}) = 4(l_{[j-1]} + l_{[j]})^{-1}\,\text{Im}(\bar{\mathbf{Q}}_{[j-.5]}\mathbf{Q}_{[j+.5]}), \tag{3.40}$$

where Im$(\cdot)$ gives the imaginary part of a quaternion.

3.8 Anatomical modeling and simulation

We now describe how our rods can be used to model complex anatomical structures such as *bones* and *muscles*; see Figure 3.1. For the former, we use a simplified version of Thiery et al. [2013] where only pill primitives are used – this primarily allows us to reduce the complexity of the collision detection/resolution codebase. We then detail the conversion of digital models of muscles into VIPERs in Section 3.8.1, and describe a few nuances about the simulation of their motion in Section 3.8.2.

3.8.1 Muscles to rods conversion – "Viperization"

We developed a (weakly-assisted) technique to convert a traditional muscle model into a collection of K rods. As outlined in Figure 3.11, our process involves several phases. We begin by computing a volumetric discretization of the muscle's surface. We then ask the artist to paint annotations on the surface marking the start (sources) and the end (sinks) of the muscle. A harmonic solve alike the one describe in Choi and Blemker [2013] is the executed to compute a field that smoothly varies in the $[0, 1]$ range along the muscle's length. We then sample M iso-levels of this field to be surfaces containing the M vertices of *each* of the K generated VIPERs. Within each iso-level we require: ① VIPERs to have the same radii, and ② to be distributed uniformly on the slice (iso-surface). To obtain this, we perform a *restricted centroidal Voronoi diagram* of K points on each slice [Botsch et al., 2010], while simultaneously penalizing the length of each rod. We alternate this variational optimization with a discrete one that re-assigns spheres to different rods in order to minimize the sum of rod lengths. This process, which we refer to as "combing", starts from one end of the muscle, and executes in order $M-1$ instances of minimum-cost bipartite matching (which we solve in polynomial time via the Hungarian algorithm), where the pairwise costs are the K^2 euclidean distances across rod nodes in two adjacent slices. A few examples of the results of this process are illustrated in Figure 3.1 and Figure 3.12.

3.8.2 Muscle simulation

Once a muscle is *viperized* as described in Section 3.8.1, rod centers within the same cross-section are connected via the *bundling* constraints in Equation 3.32. Every rod also obeys the constraints described in Section 3.5. The endpoints of rods are kinematically attached to bones, and intra-muscle collisions are disabled, while inter-muscle collisions are detected and resolved as described respectively in Section 3.7.1 and Section 3.7.2. Muscles can also be *activated* (fiber contractions generating a stronger force, producing a change of shape at constant muscle length and volume) by inserting internal forces, or even simply shortening the length of fibers resulting in the bulging effects illustrated in Figure 3.10. We also speed-up the solver convergence during fast motion by exploiting the availability of bone transformations. In more detail, each muscle particle has two skinning weights corresponding to the two bones the muscle is attached to. At the beginning of each frame we use the transform of each bone relative to their last frame's transforms to initialize the displacement of the particle using LBS, and later refine this via simulation.

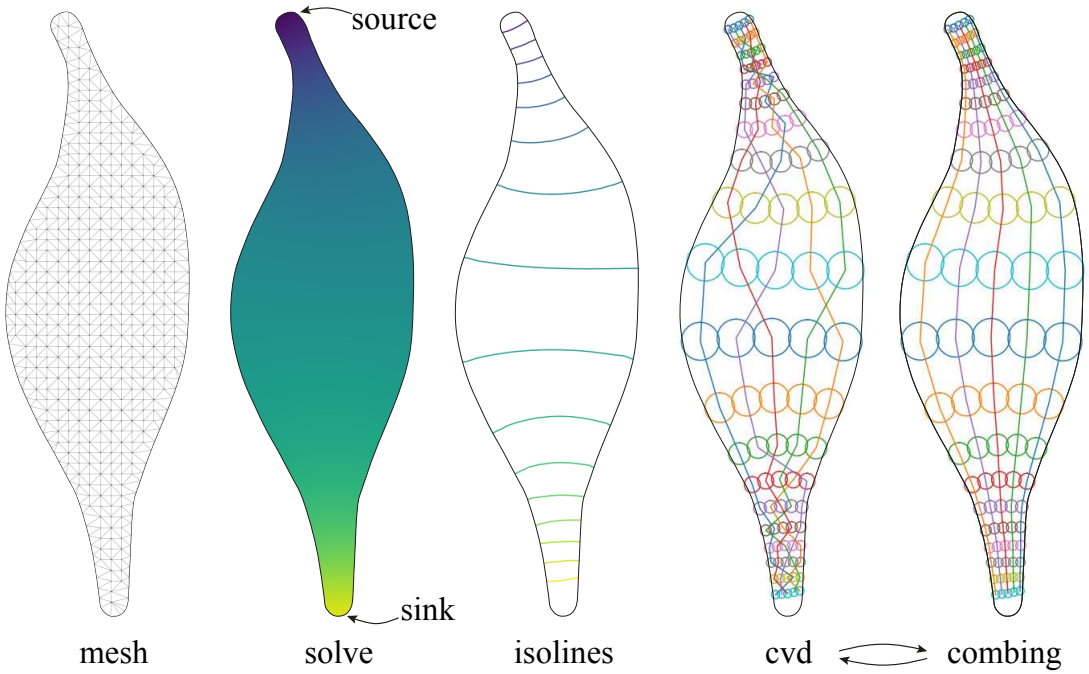


Figure 3.11: The mesh-to-VIPER conversion process. Given source/sink constraints, we compute a harmonic function in the volume, and extract a few discrete iso-levels. Within each of these, we execute a restricted CVD to place 5 elements of the same radii on these surfaces. We then execute a combinatorial optimization that connects samples across layers to produce minimal length curves.

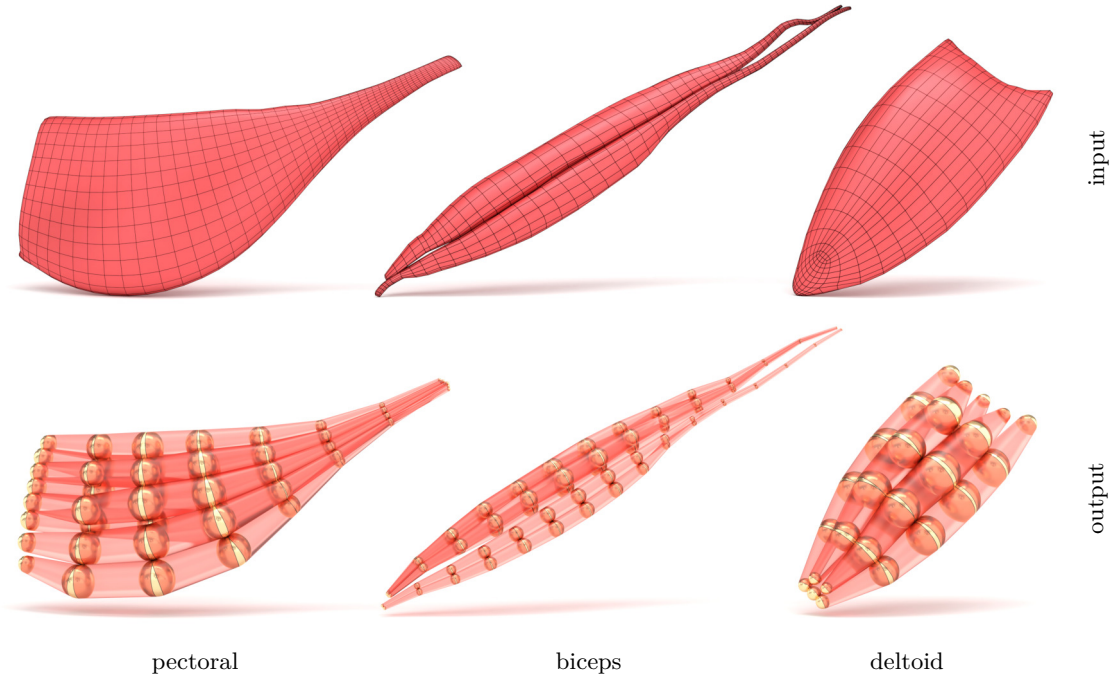


Figure 3.12: We illustrate several examples of the VIPERs extracted by our automated process. For the *pectoral*, our VIPER model employs 8 rods, each discretized by 8 elements. In comparison, the surface meshes by ZIVA contain ($|\partial V| = 1584, |\partial F| = 3164$) on the boundary and its (volumetric) simulation mesh contains ($|V| = 350, |T| = 1089$) tetrahedral elements. Where $\partial \equiv$ "boundary", $F \equiv$ "faces", $V \equiv$ "vertices" and $T \equiv$ "tetrahedra".

3.9 Conclusions & future work

In this paper, we introduced a novel formulation of cosserat rods that considers local volume, and optimizes for its local conservation. The resulting position-based simulation is highly efficient, and is straightforward to implement on graphics hardware. We demonstrated how rod-bundling is a powerful representation for the modeling of volumetric deformation – and in particular for skeletal muscles. Rather than requiring artists to model from scratch, we also introduced an algorithm to procedurally generate VIPERs with minimal user interaction. Finally, by coupling the rod simulation to a surface mesh via skinning, our model can be thought of as a direct alternative to tetrahedral meshes and cages for real-time non-rigid deformation. Most importantly, our generalized rods formulation opens up a number of venues for future work, which we classify in three broad areas, as elaborated below.

Model generalization. While in our rods we discretized the skeletal curve with piecewise linear elements, it would be interesting to investigate whether the use of *continuous* curve models such as splines would be tractable – from both mathematical, as well as implementation standpoints. Similarly to [Müller and Chentanez, 2011; Müller and Chentanez, 2011], our model could also be extended to model *anisotropic* volume deformations, that is, both the rest pose and deformed rod could have a non-circular cross section. Further, while in this paper we treated the modeling of non-circular cross-sections via bundling, the theory of medial axis [Tagliasacchi et al., 2016] tells us how any geometric object can be approximated via primitives formed as the convex-hulls of three-spheres – what Tkach et al. [2016] called "wedges". Extending our volume-invariant rods models to volume-invariant *wedges* would provide an elegant generalization of our modeling paradigm.

Anatomical modeling. As highlighted by our supplementary video, rod primitives can be exploited for the efficient approximate modeling and simulation of complex structures. Nonetheless, the dynamics of the human body are the result of the complex *interplay* between muscle, fat, and the consequent deformation of skin. Enriching our model to also account for these factors would be an interesting extension. For example, rather than driving the muscle surface via skinning as in Section 3.7.4, one could represent a muscle as a controllable *implicit blend* [Angles et al., 2017], approximate fat as an elastic offset between muscles and skin, and *simulate skin* as an elastic surface whose vertices lie on a (potentially) user-controlled iso-level of the implicit function. Further, while artistical editing of physically driven anatomical systems can be a difficult due to the complexity of simulation, our framework could immediately enable *interactive modeling*, in a similar fashion to what ZBrush/ZSphere currently provides for authoring static geometry. By extending the works in [Tkach et al., 2016, 2017], an efficient anatomical model for a particular user could also be constructed by fitting to RGBD data.

Optimization. Our solver has not *yet* directly leveraged the straightforward multi-resolution structure of rod geometry. More specifically, the curve parameterization of rods offers a domain over which designing prolongation/restriction operators needed for a *geometric multi-grid* implementation becomes straightforward. An orthogonal dimension for optimization would be to consider the existence of multi-resolution structures within the *cross-sectional* domain; see Figure 3.2. This could both be exploited in offering multi-scale interaction for artists in editing our

rod models, as well as to produce level-of-details models for efficient simulation at scale. Finally, while we employed out-of-the-box geometry processing tools to convert a triangular surface mesh into a rod model, we believe fitting a fiber-bundle model to a given solid could be achieved without the (often finicky) conversion to volume/tetrahedral mesh, but rather as a direct optimization over fiber placements.

Chapter 4

Learnable Primitives

Abstract

We introduce a deep network that can be trained to tackle image reconstruction and classification problems that involve detection of multiple object instances, without any supervision regarding their whereabouts. The network learns to extract the most significant top-K patches, and feeds these patches to a task-specific network – e.g., auto-encoder or classifier – to solve a domain specific problem. The challenge in training such a network is the non-differentiable top-K selection process. To address this issue, we lift the training optimization problem by treating the result of top-K selection as a slack variable, resulting in a simple, yet effective, multi-stage training. Our method is able to learn to detect recurrent structures in the training dataset by learning to reconstruct images. It can also learn to localize structures when only knowledge on the occurrence of the object is provided, and in doing so it outperforms the state-of-the-art.

4.1 Introduction

The ability to find multiple instances of characteristic entities in a scene is core to many computer vision applications. For example, finding people [Sewart and Andriluka, 2016; Zhang et al., 2018a], detecting arbitrary number of classes and objects [Ren et al., 2015; He et al., 2017; Redmon and Farhadi, 2017], and detecting local features [Lowe, 2004; Bay et al., 2008] all rely on this ability. In traditional vision pipelines, selecting the top-K responses in a heat-map and using their locations is the typical way to approach the problem [Lowe, 2004; Bay et al., 2008; Felzenszwalb et al., 2010]. However, due to the non-differentiable nature of this operation, it has not found immediate application in deep learning based solutions.

Circumventing top-K in end-to-end learning. To overcome this challenge, researchers proposed to use grids [Redmon et al., 2016; He et al., 2017; Detone et al., 2018], to simplify the formulation by isolating each instance [Yi et al., 2016], or to provide alternative supervision by optimizing over multiple branches [Ono et al., 2018]. While effective, they do not generalize well outside the application domain for which they were designed. Other formulations, such as the use

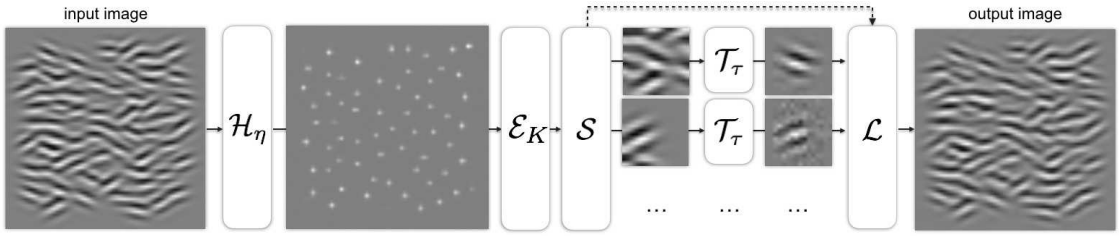


Figure 4.1: **An example of MIST architecture** – A network $\mathcal{H}_\eta$ estimates locations and scales of patches encoded in a heatmap $\mathbf{h}$. The locations of its top-k responses are extracted by $\mathcal{E}_K$. Patches are then extracted via a sampler $\mathcal{S}$, and then fed to a task-specific network $\mathcal{T}_\tau$. The results are then fed to a task-specific loss $\mathcal{L}$. In this example, the specific task is to re-synthesize the image as a super-position of (unknown, local) basis functions.

of sequential detection [Eslami et al., 2015] or channel-wise approaches [Zhang et al., 2018c] are problematic to apply when the number of instances of the same object is large.

Introducing MIST architectures. Therefore, we introduce a new deep framework which we name *Multiple Instance Spatial Transformer* or *MIST* for brevity. From a high level, the MIST framework first decomposes the image into a finite collection of patches, and then processes these patches to perform a given task. As illustrated in Figure 4.1 for the image synthesis task, given an image we first compute a heatmap via a deep network whose local maxima correspond to locations of interest. From this heatmap, we gather the parameters of the top-K local maxima, and then extract the corresponding collection of image patches via a re-sampling process. With the collection of patches, we execute the same task-specific network whose output is aggregated to finally evaluate a task-specific loss. We then optimize this task loss to train the entire framework.

Training MISTs by lifting. Training a pipeline that includes a non-differentiable selection/gather operation is non-trivial. To solve this problem we to lift the problem to a higher dimensional one by treating the parameters defining the interest points as slack variables, and introduce a hard constraint that they must correspond to the output that the heatmap network gives. This constraint is realized by introducing an auxiliary function that creates a heatmap given set of interest point parameters. We then solve for the relaxed version of this problem, where the hard constraint is turned into a soft one, and the slack variables are also optimized within the training process. Critically, our training strategy allows us to have an optimizable version of ① non-maximum suppression, and ② top-K selection, thus creating a network architecture resembling compute strategies that were dominant in pre deep-learning computer vision.

Applications. To demonstrate the capabilities of MISTs, we evaluate our network on a variety of weakly-supervised multi-instance problems. Note how in some of these applications, the value of K is the *only* supervision signal we provide. We consider ① the problem of recovering the basis functions that created a given texture, ② the classification of numbers in cluttered scenes where the only supervision is the occurrence of these numbers.

In summary, in this paper:

- we introduce the MIST framework for weakly-supervised multi-instance visual learning;
- we introduce a training method that allows the use of top-K approaches for end-to-end

trainable architectures;

- we show that our framework can reconstruct images as parts, as well as detect/classify instances without any location supervision.

4.2 Related works

Attention models and the use of localized information have been actively investigated in the literature. Some examples include discriminative tasks such as fine-grained classification [Sun et al., 2018], and pedestrian detection [Zhang et al., 2018a], and generative ones such as image synthesis from natural language [Johnson et al., 2018]. We now discuss a selection of representative works, and classify them according to how they deal with multiple instances.

Grid-based methods. Since the introduction of Region Proposal Networks (RPN) [Ren et al., 2015], grid-based strategies have been used for dense image captioning [Johnson et al., 2016], instance segmentation [He et al., 2017], keypoint detection [Georgakis et al., 2018], multi-instance object detection [Redmon and Farhadi, 2017]. Recent improvements to RPNs attempt to learn the concept of a generic object covering multiple classes [Singh et al., 2018], and to model multi-scale information [Chao et al., 2018]. The multiple transformation corresponding to separate instances can also be densely regressed via Instance Spatial Transformers [Wang et al., 2018], which removes the need to identify discrete instance early in the network. Unfortunately, all these method are fully supervised, as they require both class *labels* and object *locations* for training.

Heatmap-based methods. Heatmap-based methods have recently gained interest to detect features [Yi et al., 2016; Ono et al., 2018; Detone et al., 2018], find landmarks [Zhang et al., 2018c; Merget et al., 2018], and regress human body keypoint [Tekin et al., 2017; Newell et al., 2016]. While it is possible to output one heatmaps per type of point [Zhang et al., 2018c; Tekin et al., 2017], this still restricts the number of instances to one. Yi *et al.* [Yi et al., 2016] re-formulates the problem based on each instance, but in doing so it introduces a non-ideal difference between training and testing regimes. Grids can also be used in combination to heatmaps [Detone et al., 2018], but this results in an unrealistic underlying assumption of uniformly distributed detections in the image. Overall, heatmap-based methods excel when the "final" task of the network is generate a heatmap [Merget et al., 2018], but are problematic to use as an intermediate layer in the presence of multiple instances.

Sequential inference methods. Another way to approach multi-instance problems is to attend to one instance at a time in a sequential way. Gregor *et al.* [Gregor et al., 2015] proposes a recurrent network that processes only a small area at a time for both discriminative and generative tasks. These sequential models have then been extended to localize and recognize MNIST digits in a cluttered image [Ba et al., 2015; Eslami et al., 2015]. Overall, RNNs often struggle to generalize to sequences longer than the ones encountered during training, and while recent results on inductive reasoning are promising [Gupta et al., 2018], their performance does not scale well when the number of instances is large.

Knowledge transfer. To overcome the acquisition cost of labelled training data, one can transfer knowledge from labeled to unlabeled dataset. For example, Inoue *et al.* [Inoue et al., 2018] train on a single instance dataset, and then attempt to generalize to multi-instance domains, while Uijlings *et al.* [Uijlings et al., 2018] attempts to also transfer a multi-class proposal generator to the new domain. While knowledge transfer can be effective, it is highly desirable to devise unsupervised methods such as ours that do not depend on an additional dataset.

Weakly supervised methods. To further reduce the labeling effort, weakly supervised methods have also been proposed. Wan *et al.* [Wan et al., 2018] learns how to detect multiple instances of a single object via region proposals and ROI pooling, while Tang *et al.* [Tang et al., 2018b] proposes to use a hierarchical setup to refine their estimates. Gao *et al.* [Gao et al., 2018] provides an additional supervision by specifying the number of instances in *each* class, while Zhang *et al.* [Zhang et al., 2018b] localizes objects by looking at the network activation maps [Zhou et al., 2016; Selvaraju et al., 2017]. However, all these method still rely on region proposals from an existing method, or define them via a hand-tuned process.

4.3 MIST Framework

A prototypical MIST architecture is composed of two trainable components: ① the first module receives an image as input and extracts a collection of patches, at image locations and scales that are computed by a trainable heatmap network $\mathcal{H}_\eta$ with weights η; see Section 4.4. ② the second module processes each extracted patch with a task-specific network $\mathcal{T}_\tau$ whose weights τ are shared across patches, and further manipulates these signals to express a task-specific loss $\mathcal{L}_{task}$; see Section 4.5. The two modules are connected through non-maximum suppression on the scale-space heatmap output of $\mathcal{H}_\eta$, followed by a top-K selection process to extract the parameters defining the patches, which we denote as $\mathcal{E}_K$. We then sample patches at these locations through bilinear sampling $\mathcal{S}$ and feed them the second module.

The defining characteristic of MIST architectures is that they are *quasi-unsupervised*: the only strictly required supervision is the number K of patches to extract. The training of MIST architectures is summarized by the optimization:

$$\arg\min_{\tau,\eta} \ \mathcal{L}_{\text{task}}(\mathcal{T}_\tau(\mathcal{S}(\mathcal{E}_K(\mathcal{H}_\eta(I))))) \tag{4.1}$$

where τ, η are the network trainable parameters. Note how in the expression above $\mathcal{E}_K$ is non-differentiable, thus making (4.1) unapproachable by back-propagation for training.

Example task. In Figure 4.1, we illustrate an example of a MIST architecture for image reconstruction. In more details, the task is to understand how to re-synthesize the image as a super-position of K spatially localized basis functions. Note that, while this task is quasi-unsupervised, it presents several joint challenges as it needs to estimate: ① an unknown shared low-dimensional set of latent bases; ② where to place instances from this latent space; ③ the latent coefficients representing each instance. Further details and additional example tasks are described in Section 4.5.

Training MISTs. While our MISTs enable us to approach several vision tasks, such as the ones above, with minimal supervision the true challenge lies in the definition of an effective training strategy. Back-propagation through a selection process is possible, alike to what is performed for max-pooling. In Section 4.6, we argue why this is highly detrimental, and propose an effective multi-stage training solution.

Evaluation and implementation. We qualitative and quantitatively evaluate MISTs on a number of tasks in Section 4.7, and provide further implementation details in Section 4.8.

4.4 Patch extraction

We extracts a set of K (square) patches that correspond to "important" locations in the image – where importance is a direct consequence of $\mathcal{L}_{\text{task}}$. The localization of such patches can be computed by regressing a 2D heatmap whose top-K peaks correspond to the patch centers. However, as we do not assume these patches to be equal in size, we regress to a collection of heatmaps at different scales. To limit the number of necessary scales, we use a discrete and sparse scale-space, while resolving for intermediate scales by weighted interpolation.

Multiscale heatmap network – $\mathcal{H}_\eta$. Our multiscale heatmap network is inspired by LF-Net [Ono et al., 2018]. We employ a fully convolutional network with (shared) weights η at multiple scales, indexed by $s = 1 \dots S$, on the input image I. The weights η across scales are shared so that the network cannot implicitly favor a particular scale. To do so, we first downsample the image to each scale I_s, execute the network $\mathcal{H}_\eta$ on it, and finally upsample to the original resolution. This process generates a multiscale heatmap tensor $\mathbf{h} = \{\mathbf{h}_s\}$ of size $H \times W \times S$ where $\mathbf{h}_s = \mathcal{H}_\eta(I_s)$, and H is the height of the image and W is the width. For the convolutional network we use 4 ResNet blocks [He et al., 2015], where each block is composed of two 3×3 convolutions with 32 channels and relu activations without any downsampling. We then perform a *local spatial softmax* operator [Ono et al., 2018] with spatial extent of 15×15 to sharpen the responses.

Extracting patch location and scale. We first normalize the heatmap tensor so that it has a unit sum along the scale dimension:

$$\mathbf{h}_s = \frac{\mathcal{H}_\eta(I_s)}{\sum_s \mathcal{H}_\eta(I_s) + \epsilon}, \tag{4.2}$$

where $\epsilon = 10^{-6}$ is added to prevent divisions by zero. We then compute the top-K spatial locations across all scales, to obtain the image-space coordinates of patch centers:

$$\{(x_k, y_k)\} = \text{TOPK}(\max_s \mathbf{h}_s). \tag{4.3}$$

Note that a direct extraction of K maxima is possible because the aforementioned local spatial softmax performs a localized non-maximal suppression. The corresponding scale is computed by weighted first order moments [Suwajanakorn et al., 2018], where the weights are the responses in the corresponding heatmaps:

$$s_k = \sum_s \underbrace{\mathbf{h}_s(x_k, y_k)}_{\text{weights}} s. \tag{4.4}$$

Note we do not need to normalize here, as (4.2) has unit sum along the s dimension. These two operations are abstracted by our top-K extractor $\{\mathbf{x}_k\} = \mathcal{E}_K(\mathbf{h})$ in (4.1).

Note also that our extraction process uses a single heatmap for all instances that we extract. By contrast, existing heatmap-based methods [Eslami et al., 2015; Zhang et al., 2018c] typically rely on heatmaps dedicated to each instance, which is problematic when an image contains two instances of the same class. Conversely, we restrict the role of the heatmap network $\mathcal{H}_\eta$ to find the "important" areas in a given image, without having to distinguishing between classes, hence simplifying learning.

Patch sampling. As a patch is uniquely parameterized its location and scale $\mathbf{x}_k = (x_k, y_k, s_k)$, we can then proceed to sample its corresponding tensor via bilinear interpolation [Jaderberg et al., 2015]:

$$\{\mathbf{P}_k\} = \mathcal{S}\left(I, \{\mathbf{x}_k\}\right). \tag{4.5}$$

Comparison to LF-Net. Note that differently from LF-Net [Ono et al., 2018], we do not perform a softmax along the scale dimension. The scale-wise softmax in LF-Net is problematic as the computation for a softmax function relies on the input to the softmax being *unbounded*. For example, in order for the softmax function to behave as a max function, due to exponentiation, it is necessary that one of the input value reaches infinity (i.e. the value that will correspond to the max), or that all other values to reach negative infinity. However, at the network stage where softmax is applied in [Ono et al., 2018], the score range from zero to one, effectively making the softmax behave similarly to averaging. Our formulation does not suffer from this drawback.

4.5 Task-specific networks

We now introduce two instances of the MIST architectures corresponding to different applications. We keep the heatmap network and the extractor architectures the same, and only change the task-specific network, as well as the loss used for supervision. In particular, we consider a reconstruction problem in Section 4.5.1, and a classification problem in Section 4.5.2. Further implementation details can be found in Section 4.8.

4.5.1 Image reconstruction

As illustrated in Fig. 4.1, for image reconstruction we append our patch extraction network with a *shared* auto-encoder for each extracted patch. We can then train this network to *reconstruct* the original image by inverting the patch extraction process, and forming the task specific loss to be the ℓ_2 norm between the input image and the reconstructed image. Specifically, we introduce the inverse sampling operation $\mathcal{S}^{-1}(\mathbf{P}_i, \mathbf{x}_i)$, which starts with an image of all zeros, and places the patch $\mathbf{P}_i$ at $\mathbf{x}_i$. We then add all the images together to obtain the reconstructed image, overall expressing the following task loss:

$$\mathcal{L}_{task} = \left\| \mathbf{I} - \sum_i \mathcal{S}^{-1}\left(\mathbf{P}_i, \mathbf{x}_i\right) \right\|_2^2. \tag{4.6}$$

Algorithm 1 Multi-stage optimization for MISTs

Require: K : number of patches to extract, $\mathcal{L}_{\text{task}}$: task specific loss, I : input image, η : parameters of the heatmap network, τ : parameters of the task network.

function TRAINMIST(I, $\mathcal{L}_{\text{task}}$)
  for each training batch **do**
    $\tau \leftarrow$ Optimize $\mathcal{T}_\tau$ with $\mathcal{L}_{\text{task}}$
    for $n = 1$ to N **do**
      $\{\mathbf{x}_k^*\} \leftarrow$ Optimize $\{\mathbf{x}_k\}$ with $\mathcal{L}_{\text{task}}$
    end for
    $\bar{\mathbf{h}} \leftarrow \mathcal{E}_K^{-1}(\{\mathbf{x}_k^*\})$
    $\eta \leftarrow$ Optimize $\mathcal{H}_\eta$ with $\mathcal{L}_{\text{lift}}$
  end for
end function

Overall, the network is designed to *jointly* model and localize repeating structures in the input signal. Regressing shared basis functions can be related to non-local mean processes [Buades et al., 2005], as a model for the input signal is created by agglomerating the information in scattered spatial instances. Our task architecture is also related to transforming auto-encoders [Hinton et al., 2011], where the difference is that in this previous work a *single* instance is present in the image, and that they provide the ground truth transformations as a supervision.

4.5.2 Multiple instances classification

By appending the patch extraction module with a classification network we can realize an architecture for multiple instance learning. For each extracted patch $\mathbf{P}_k$ we apply a shared classifier network to output $\hat{\mathbf{y}}_k \in \mathbb{R}^C$, where C is the number of classes. In turn, these are then converted into probability estimates by the transformation $\hat{\mathbf{p}}_k = \text{softmax}(\hat{\mathbf{y}}_k)$. By denoting the one-hot ground-truth labels of instance l in the image as $\mathbf{y}_l$, we define the multi-instance classification loss as

$$\mathcal{L}_{task} = H\left(\frac{1}{L}\sum_{l=1}^{L}\mathbf{y}_l, \frac{1}{K}\sum_{k=1}^{K}\hat{\mathbf{p}}_k\right) \quad , \tag{4.7}$$

where $H(\cdot)$ denotes cross entropy and L is the number of instances in the image. Note here that we *do not* provide supervision about where each class instances are, yet the detector network will automatically learn how to localize the content with minimal supervision.

4.6 Training MISTs

While it is technically possible to back-propagate through MIST architectures, the gradients would only flow through the spatial regions corresponding to the K selected keypoints – this results in a training process that *ignores* locations away from the selection. This is a fundamental issue, as, in order for the network to learn to *only* respond to the desired locations, we need negative examples just as much as we need positive examples. To circumvent this problem, we propose a multi-stage training optimization.

Differentiable top-K via lifting. The introduction of auxiliary variables (i.e. lifting) to simplify the structure of an optimization problem has proven effective in a range of domains ranging from registration via ICP [Taylor et al., 2016], to efficient deformation models [Sorkine and Alexa, 2007], and robust optimization [Zach and Bournaoud, 2018]. To simplify our training optimization, we start by decoupling the heatmap tensor from the optimization (4.1) by introducing the corresponding auxiliary variables $\bar{\mathbf{h}}$, as well as the patch parameterization variables $\{\mathbf{x}_k\}$ that are extracted by the top-K extractor:

$$\underset{\eta,\tau,\bar{\mathbf{h}},\{\mathbf{x}_k\}}{\arg\min} \ \mathcal{L}_{\text{task}}(\mathcal{T}_\tau(\mathcal{S}(\{\mathbf{x}_k\}))) \tag{4.8}$$

$$\text{s.t.} \quad \{\mathbf{x}_k\} = \mathcal{E}_K(\bar{\mathbf{h}}) \tag{4.9}$$

$$\bar{\mathbf{h}} = \mathcal{H}_\eta(I) \tag{4.10}$$

We then relax (4.10) to a least-squares penalty:

$$\underset{\eta,\tau,\bar{\mathbf{h}},\{\mathbf{x}_k\}}{\arg\min} \ \mathcal{L}_{\text{task}}(\mathcal{T}_\tau(\mathcal{S}(\{\mathbf{x}_k\}))) + \|\bar{\mathbf{h}} - \mathcal{H}_\eta(I)\|_2^2 \tag{4.11}$$

$$\text{s.t.} \quad \{\mathbf{x}_k\} = \mathcal{E}_K(\bar{\mathbf{h}}) \tag{4.12}$$

and finally approach it by alternating optimization:

$$\underset{\tau,\{\mathbf{x}_k\}}{\arg\min} \ \mathcal{L}_{\text{task}}(\mathcal{T}_\tau(\mathcal{S}(\{\mathbf{x}_k\}))) \tag{4.13}$$

$$\underset{\eta}{\arg\min} \ \|\bar{\mathbf{h}} - \mathcal{H}_\eta(I)\|_2^2 \tag{4.14}$$

where $\bar{\mathbf{h}}$ has been dropped as it is not a free parameter: it can be computed as $\bar{\mathbf{h}} = \mathcal{E}_K^{-1}(\{\mathbf{x}_k\})$ after the $\{\mathbf{x}_k\}$ have been optimized by (4.13), and as $\bar{\mathbf{h}} = \mathcal{H}_\eta(I)$ after η have been optimized by (4.14). To accelerate training, we further split (4.13) into two stages, and alternate between optimizing τ and $\{\mathbf{x}_k\}$. In particular, multiple optimization iterations of $\{\mathbf{x}_k\}$ are executed to allow keypoints to displace faster during training. The summary for the three stage optimization procedure is outlined in Alg. 1: ① we optimize the parameters τ with the loss $\mathcal{L}_{task}$; ② we then fix τ, and refine the positions of the patches $\{\mathbf{x}_k\}$ for T iterations with $\mathcal{L}_{task}$. ③ with the optimized patch positions $\{\mathbf{x}_k^*\}$, we invert the top-K operation by creating a target heatmap $\bar{\mathbf{h}}$, and optimize the parameters η of our heatmap network $\mathcal{H}$ using ℓ_2 distance between the two heatmaps, $\mathcal{L}_{\text{lift}} = \|\bar{\mathbf{h}} - \mathcal{H}_\eta(I)\|_2^2$. Notice that we are not introducing *any* additional supervision signal that is tangent to the given task.

Generating the target heatmap – $\mathcal{E}_K^{-1}(\{\mathbf{x}_k\})$. For creating the target heatmap $\bar{\mathbf{h}}$, we create a tensor that has zeros everywhere except for the positions corresponding to the optimized positions. However, as the optimized patch parameters are no longer integer values, we need to quantize them with care. For the spatial locations we simply round to the nearest pixel, which at most creates a quantization error of half a pixel, which does not cause problems in practice. For scale however, simple nearest-neighbor assignment causes too much quantization error as our scale-space is sparsely sampled. We therefore assign values to the two nearest neighboring scales in a way that the center of mass would be the optimized scale value. That is, we create a heatmap tensor

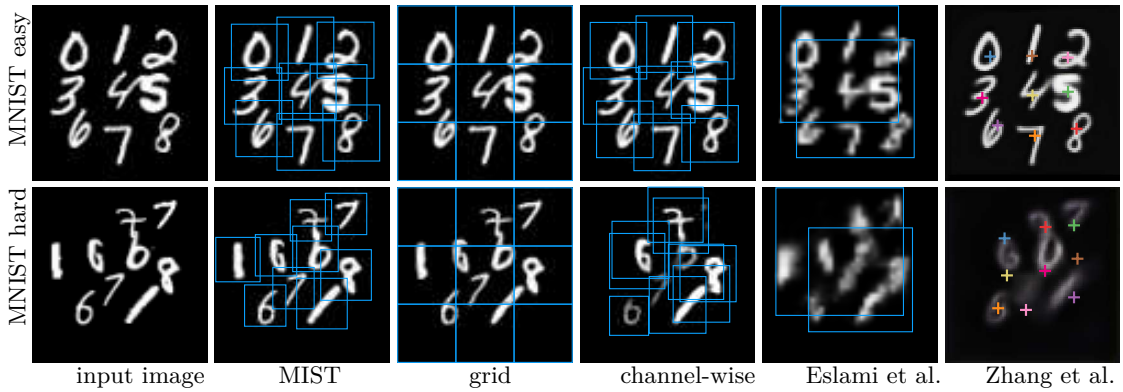


Figure 4.2: MNIST character synthesis examples for (top) the "easy" single instance setup and (bottom) the hard multi-instance setup. We compare the output of MISTs to grid, channel-wise, Eslami *et al.* [Eslami et al., 2015] and Zhang *et al.* [Zhang et al., 2018c].

that would result in the optimized patch locations when used in forward inference.

4.7 Results and evaluation

To demonstrate the effectiveness of our framework we evaluate two different tasks. We first perform a quasi-unsupervised image reconstruction task, where *only* the total number of instances in the scene is provided. We then show that our method can also be applied to weakly supervised multi-instance classification, where only image-level supervision is provided. Note that, unlike region proposal based methods, our localization network only relies on cues from the classifier, and *both* networks are trained from scratch.

4.7.1 Image reconstruction

From the MNIST dataset, we derive two different scenarios. In the "MNIST easy" dataset, we consider a simple setup where the *sorted* digits are confined to a perturbed *grid* layout; see Figure 4.2 (top). Specifically, we perturb the digits with a Gaussian noise centered at each grid center, with a standard deviation that is equal to one-eighths of the grid width/height. In the "MNIST hard" dataset, the positions are randomized through a Poisson distribution [Bridson, 2007], as is the identity, and cardinality of each digit. Note how we allow multiple instances of the same digit to appear in this variant. As expected, both these datasets contain a training and testing subsets, and the testing portion is never seen at training time.

Comparison baselines. We compare out method against four baselines: ① *grid* we setup a grid of keypoints, and apply the same auto-encoder architecture as MIST to reconstruct the input image; ② in the *channel-wise* variant we use the same heatmap network, except for the last convolutional layer giving K channels as output, where each channel is dedicated to an interest point. Their locations are obtained through a channel-wise soft argmax as in [Zhang et al., 2018c]. We also use the same architecture for the auto-encoder as MIST; ③ the method of *Eslami* et al. *[Eslami et al., 2015]* is a sequential generative model. To generate nine digits, it is required for the

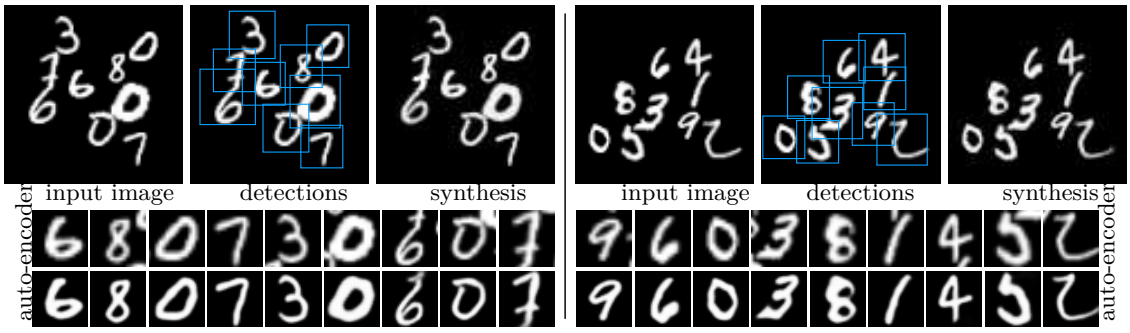


Figure 4.3: Two auto-encoding examples learnt from MNIST-hard. In the top row, for each example we visualize input, patch detections, and synthesis. In the bottom row we visualize each of the extracted patch, and how it is modified by the learnt auto-encoder. Notice the full process is self-supervised, with the exception that we know every image contains 9 numbers.

Table 4.1: Reconstruction error in terms of Root Mean Square Error (RMSE) across the various baselines. Best results in bold, and second best underlined. Our method performs best on MNIST easy, and second best on MNIST hard. Note however, that the grid method is not able to learn any notion of individual digits.

	MIST	Grid	Ch.-wise	[Eslami et al., 2015]	[Zhang et al., 2018c]
MNIST easy	**.038**	<u>.039</u>	.042	.100	.169
MNIST hard	<u>.089</u>	**.047**	.128	.154	.191
Gabor	**.095**	N/A	N/A	N/A	N/A

method to be trained with also examples where various number of total digits exist (images with only 1 digit, 2 digits, etc.). We make a special exception for this method, and populate the training set with all of these cases; ④ we finally compare to the state-of-the-art method by *Zhang* et al. *[Zhang et al., 2018c]* that provides a heatmap-based method with channel-wise strategy for unsupervised learning of landmarks.

Results for "MNIST easy". As shown in Figure 4.2 (top) all methods successfully re-synthesize the image, with the exception of [Eslami et al., 2015]. As this method is sequential, with nine digits the sequential implementation simply becomes too difficult to optimize through. Note how this method only learns to describe the scene with a few large regions. Quantitative results are summarized in Table 4.1.

Results for "MNIST hard". As shown in Figure 4.2 (bottom), all methods except ours fail to properly represent the image, where not only it was able to reconstruct the image, but also learnt how to localize the digits. Note that while it might *look* like the grid method succeeded, its trained auto-encoder simply failed in capturing the concept of individual digits. Conversely, as shown in Figure 4.3, our method is able to learn this, demonstrated by the auto-encoder successfully separating the existing overlaps. For quantitative results, please see Table 4.1.

Finding the basis of a procedural texture. We further demonstrate that our methods can be used to find the basis function of a procedural texture. For this experiment we synthesize textures with procedural Gabor noise [Lagae et al., 2009]. Gabor noise is obtained by convolving oriented Gabor wavelets with a Poisson impulse process. Hence, given exemplars of noise, our

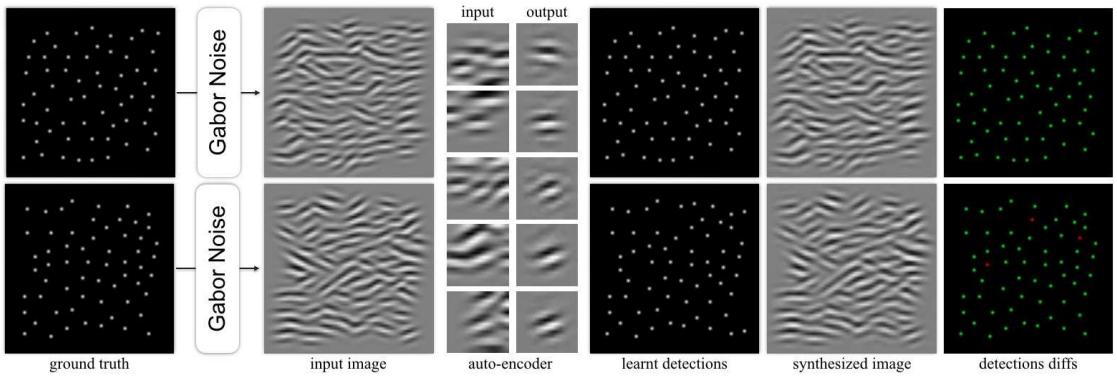


Figure 4.4: Unsupervised inverse rendering of procedural Gabor noise. In the last column we highlight localization mistakes.

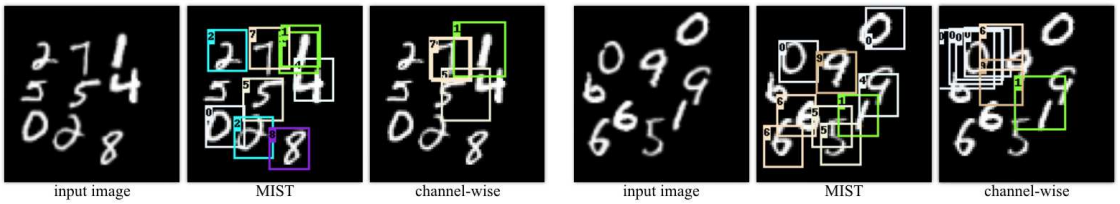


Figure 4.5: Two qualitative examples for detection and classification on our Multi-MNIST dataset.

framework is tasked to regress the underlying impulse process, and reconstruct the Gabor kernels so that when the two are combined, we can reconstruct the original image. Figure 4.4 illustrates the results of our experiment. Note how the learnt auto-encoder learnt very well to reconstruct Gabor kernels, even though in the training images they are heavily overlapped. Further, note that the number of instances detected is significantly larger than that possible with other methods.

4.7.2 Multiple instance classification

To test our method in a multiple instance classification setup, we rely on the *MNIST hard* dataset. We compare our method to *channel-wise*, as other baselines are designed for purely generative tasks. To evaluate the accuracy of the models, we compute the intersection over union (IoU) between the ground-truth bounding box and the detection results, and assign it as a match if the IoU score is over 50%. We report the number of correctly classified matches in Table 4.2. Our method clearly outperforms the *channel-wise* strategy. A few qualitative results are illustrated in Figure 4.5. Note that even without direct supervision on the digits locations, our method correctly localizes them. Conversely, the *channel-wise* strategy fails to learn. This is because *multiple instances* of the

Table 4.2: Instance level detection and classification performance on the MNIST hard dataset.

	MIST	channel-wise
IOU 50%	**84.6%**	25.4%
Classification	**95.6%**	75.5%
Both	**83.5%**	24.8%

same digits are present in the image. For example, in the example Figure 4.5 (right), we have two number sizes, zeros, and nines. This prevents any of these digits from being detected/classified properly by a channel-wise approach.

4.8 Implementation details

Auto-encoder network. The input layer of the autoencoder is 32x32xC where C is the number of color channels. We use 5 up/down-sampling levels. Each level is made of 3 resnet blocks and each resnet block uses a number of channels that doubles after each downsampling step. Resnet blocks uses 3x3 convolutions of stride 1 with relu activation. For downsampling we use 2D max pooling with 2x2 stride and kernel. For upsampling we use 2D transposed convolutions with 2x2 stride and kernel. The output layer uses a sigmoid function. We use layer normalization before each convolution layer.

Classification network. We re-use the same architecture as encoder for first the task and append a dense layer to map the latent space to the score vector of our 10 digit classes.

4.9 Conclusion

In this paper, we introduced the MIST framework for multi-instance image reconstruction and classification. Both these tasks are based on localized analysis of the image, yet we train the network without providing any localization supervision. The network learns how to extract patches on its own, and these patches are then fed to a task-specific network to realize an end goal. While at first glance the MIST framework might appear non-differentiable, we show how via lifting they can be effectively trained in an end-to-end fashion. We demonstrated the effectiveness of MNIST by introducing a variant of the MIST dataset, and demonstrating compelling performance in both reconstruction and classification. We also show how the network can be trained to reverse engineer a procedural texture synthesis process.

Chapter 5

Conclusion

5.1 Summary

In Chapter 2, we have presented a method to reverse-engineer how primitives blend together. We have introduced a template for blending operators that can mimic any operator from previous work but also completely new ones. This template can be optimized to fit sample points in the operator domain. This enables the interactive design of operators that can be applied by users in composition modeling without necessitating any technical knowledge.

In Chapter 3, we have introduced a new volume-preserving primitive for physical simulations. In order to efficiently simulate it, we have extended the position-based dynamics simulation framework by adding an extra degree of freedom. To do so, we have carefully derived the equations of the updates and constraints. We have demonstrated that this primitive is an effective collision proxy. Additionally, this primitive can be used as a volumetric proxy for surface deformation. Finally, we have shown that this simple primitive can be useful to simulate complex objects such as muscles.

In Chapter 4, we have introduced a novel unsupervised Deep Learning framework to learn and detect recurring primitives. At the core of our contribution is our multi-steps training process that is required to train our network despite the non-differentiable top-k operation. We have successfully applied our method on multi-instance reconstruction and classification tasks, outperforming the state-of-the-art unsupervised methods.

5.2 Future Work

Joint optimization of primitives and composition functions. In Chapter 2, a method to extract the function by which some primitives are combined was presented. This assumes that the shape of the primitives and their pose are known. But this information is rarely available. Most three-dimensional models are just specified in terms of their final shape, not by the primitives that compose them. Of course, there has been some work on extracting primitives but it would be interesting to jointly optimize for the primitives and the composition functions.

Simulation of primitives. In Chapter 3, the use the "tapered capsule" as a useful primitive for simulation was introduced. One might wonder whether there are more primitives that are well suited for simulating physical objects. An interesting example to investigate is the "wedge" primitive that is the two-dimensional counterpart of the tapered capsule. Its skeleton is a triangle (as opposed to a segment for a capsule).

MIST for unsupervised object detection. Presented in Chapter 4, MISTs are a first step towards the definition of optimizable image-decomposition networks that could be extended to a number of exciting *unsupervised* learning tasks. Amongst these, we intend to explore the applicability of MISTs to unsupervised detection/localization of objects, facial landmarks, and local feature learning.

Learning three-dimensional primitives. Another interesting venue for research would be to apply MISTs to three-dimensional data. This would have applications in scene compression, scene understanding, object detection, and keypoints detection.

Animation decomposition. So far, in Chapter 4, we have only used fixed images to learn to primitives. But in an animated sequence such as a video or a three-dimensional animation, primitives have a temporal coherency. This could be leveraged to learn more efficiency and solve ambiguous situations. During an animation primitives may also deform, which should also be taken into account.

Appendices

Appendix A

Appendix for Chapter 3

A.1 Elastic Potentials

Strain. As defined in Section 3.5 we can write the strain energy as

$$\mathrm{E}_{\text{strain}} = \iint_D \|\mathbf{R}^T\nabla\mathbf{p} - \bar{\mathbf{R}}^T\nabla\bar{\mathbf{p}}\|^2_{\mathbf{K}} dxdy, \tag{A.1}$$

where we drop the subscript from $\mathbf{K}$ for brevity. This equation can be extended as

$$\iint_D \|\mathbf{R}^T\nabla(\mathbf{c} + s\mathbf{R}\mathbf{q}) - \bar{\mathbf{R}}^T\nabla(\bar{\mathbf{c}} + \bar{s}\bar{\mathbf{R}}\mathbf{q})\|^2_{\mathbf{K}} dxdy, \tag{A.2}$$

where

$$\begin{aligned}\mathbf{R}^T\nabla(\mathbf{c} + s\mathbf{R}\mathbf{q}) &= \mathbf{R}^T(\nabla\mathbf{c} + \nabla s\mathbf{R}\mathbf{q} + s\nabla\mathbf{R}\mathbf{q} + s\mathbf{R}\nabla\mathbf{q}), &\text{(A.3)}\\ &= \mathbf{R}^T\nabla\mathbf{c} + \nabla s\mathbf{q} + s\mathbf{R}^T\nabla\mathbf{R}\mathbf{q} + s\nabla\mathbf{q}. &\text{(A.4)}\end{aligned}$$

We can derive the gradient operator for each part of this summation leading to

$$\mathbf{R}^T\nabla\mathbf{c} = \begin{bmatrix}\mathbf{0} & \mathbf{0} & \mathbf{R}^T\nabla_z\mathbf{c}\end{bmatrix} = \begin{bmatrix}0&0&0\\0&0&0\\0&0&\mathbf{w}^T\nabla_z\mathbf{c}\end{bmatrix}, \tag{A.5}$$

$$\nabla s\mathbf{q} = \begin{bmatrix}\mathbf{0} & \mathbf{0} & \nabla_z s\mathbf{q}\end{bmatrix} = \begin{bmatrix}0&0&\nabla_z sx\\0&0&\nabla_z sy\\0&0&0\end{bmatrix}, \tag{A.6}$$

$$s\mathbf{R}^T\nabla\mathbf{R}\mathbf{q} = \begin{bmatrix}\mathbf{0} & \mathbf{0} & s\boldsymbol{\Omega}\times\mathbf{q}\end{bmatrix} = \begin{bmatrix}0&0&-s\Omega^w y\\0&0&s\Omega^w x\\0&0&s\Omega^u y - s\Omega^v x\end{bmatrix}, \tag{A.7}$$

$$s\nabla\mathbf{q} = \begin{bmatrix}s\mathbf{e}^x & s\mathbf{e}^y & \mathbf{0}\end{bmatrix} = \begin{bmatrix}s&0&0\\0&s&0\\0&0&0\end{bmatrix}, \tag{A.8}$$

where $\boldsymbol{\Omega} = [\Omega^u\Omega^v\Omega^w]^T$ is the Darboux vector; and analogous expressions can be derived for $\nabla\bar{\mathbf{p}}$.

We can now observe than (A.1) can be broken up into the sum of distinct energies

$$\mathrm{E}_{\text{strain}} = \iint_D k^z(\mathbf{w}^T\nabla_z\mathbf{c} - \bar{\mathbf{w}}^T\nabla_z\bar{\mathbf{c}})^2 \tag{A.9}$$

$$+\|(\nabla_z s - \nabla_z\bar{s})\mathbf{q}\|^2_{\mathbf{K}} \tag{A.10}$$

$$+\|(s\boldsymbol{\Omega} - \bar{s}\bar{\boldsymbol{\Omega}}) \times \mathbf{q}\|^2_{\mathbf{K}} \tag{A.11}$$

$$+(k^x + k^y)(s - \bar{s})^2 dxdy, \tag{A.12}$$

as the cross terms evaluate to $\mathbf{0}$. After integrating over the disc we can reformulate (A.9) as

$$\mathrm{E}_{\text{strain}} = \pi r^2 k^z \|\nabla_z\mathbf{c} - \mathbf{w}\bar{\mathbf{w}}^T\nabla_z\bar{\mathbf{c}}\|_2^2 \tag{A.13}$$

$$+ \tfrac{\pi r^4(k^x+k^y)}{4}(\nabla_z s - \nabla_z\bar{s})^2 \tag{A.14}$$

$$+ \|s\boldsymbol{\Omega} - \bar{s}\bar{\boldsymbol{\Omega}}\|^2_{\mathbf{H}} \tag{A.15}$$

$$+ \pi r^2(k^x + k^y)(s - \bar{s})^2, \tag{A.16}$$

where $\mathbf{H} = \left[\begin{array}{ccc} \frac{\pi r^4 k^z \mathbf{e}^x}{4} & \frac{\pi r^4 k^z \mathbf{e}^y}{4} & \frac{\pi r^4 (k^x+k^y)\mathbf{e}^z}{4} \end{array}\right]$ is the second moment of the area of a disc scaled by the stiffness.

Volume. As defined in Section 3.5 we can write the volume energy as

$$\mathrm{E}_{\text{vol}} = \iint_D k(|\nabla\mathbf{p}| - |\nabla\bar{\mathbf{p}}|)^2 dxdy \tag{A.17}$$

$$= \iint_D k(|\mathbf{R}^T\nabla\mathbf{p}| - |\bar{\mathbf{R}}^T\nabla\bar{\mathbf{p}}|)^2 dxdy, \tag{A.18}$$

Based on the strain derivation, the determinant can be computed as

$$|\mathbf{R}^T\nabla(\mathbf{c} + s\mathbf{R}\mathbf{q})| = \left|\begin{bmatrix} s & 0 & \nabla_z sx - s\Omega^w y \\ 0 & s & \nabla_z sy + s\Omega^w x \\ 0 & 0 & \mathbf{w}^T\nabla_z\mathbf{c} + s\Omega^u y - s\Omega^v x \end{bmatrix}\right| \tag{A.19}$$

$$= s^2\mathbf{w}^T\nabla_z\mathbf{c} + s^3(\Omega^u y - \Omega^v x). \tag{A.20}$$

We can now integrate over the disc leading to

$$\mathrm{E}_{\text{vol}} = \pi r^2 k\|s^2\nabla_z\mathbf{c} - s^2_{\bar{\mathbf{p}}}\mathbf{w}\bar{\mathbf{w}}^T\nabla_z\bar{\mathbf{c}}\|_2^2 \tag{A.21}$$

$$+ \tfrac{\pi r^4 k}{2}(s^3\Omega^u - \bar{s}^3\bar{\Omega}^u)^2 \tag{A.22}$$

$$+ \tfrac{\pi r^4 k}{2}(s^3\Omega^v - \bar{s}^3\bar{\Omega}^v)^2. \tag{A.23}$$

A.2 Prediction Step

As described in (3.9), the inertial potential over the disc is of the form

$$\mathrm{E}_{\text{inertia}} = \iint_D \tfrac{m}{2h^2}\|\mathbf{p}_t - \hat{\mathbf{p}}_t\|_2^2 dxdy. \tag{A.24}$$

We aim at finding the unknowns $\mathbf{x}_t = [\, \mathbf{c}_t^T \;\; s_t \;\; \boldsymbol{\theta}_t^T \,]^T$ that minimize (A.24) giving us the prediction update. We denote by $\boldsymbol{\theta}$ the angles parametrizing the rotation matrix. Because the deformation function $\mathbf{p}_t$ is non linear, i.e., due to the rotational degrees of freedom, we rely on a Gauss-Newton iterative scheme for the minimization. We linearize (A.24) at $\mathbf{x}_t^k$ leading to

$$\underset{\Delta \mathbf{x}_t^k}{\arg\min} \iint\limits_D \tfrac{m}{2h^2} \|\mathbf{A}\Delta\mathbf{x}_t^k - \mathbf{b}\|_2^2 dxdy, \tag{A.25}$$

where k is the iteration number, and $\Delta\mathbf{x}_t = [\, \Delta\mathbf{c}_t^T \;\; \Delta s_t \;\; \Delta\boldsymbol{\theta}_t^T \,]^T$. At each iteration we minimize (A.25), and then apply the update $\mathbf{x}_t^{k+1} = \mathbf{x}_t^k + \Delta\mathbf{x}_t^k$, where we initialize $\mathbf{x}_t^0 = \mathbf{x}_{t-1}$. The matrix $\mathbf{A}$ can be written as $\mathbf{A} = [\, \mathbf{I}_{3\times 3} \;\; \mathbf{R}_t^k\mathbf{q} \;\; -s_t^k[\mathbf{R}_t^k\mathbf{q}]_\times \,]$, where $[\cdot]_\times$ is a cross product skew-symmetric matrix. The vector $\mathbf{b}$ is defined as $\mathbf{b} = \mathbf{p}_{t-1} + h\dot{\mathbf{p}}_{t-1} + \frac{h^2}{m}\mathbf{f}_{\text{ext}} - \mathbf{p}_t^k$. To compute the prediction updates we will use a single iteration of Gauss-Newton so $\mathbf{b} = h\dot{\mathbf{p}}_{t-1} + \frac{h^2}{m}\mathbf{f}_{\text{ext}}$ as $\mathbf{p}_t^0 = \mathbf{p}_{t-1}$. We will now drop the superscripts and the subscripts to improve readability. As Equation A.25 is quadratic we can be find its minimum by solving the normal equation

$$\left(\iint\limits_D \tfrac{m}{h^2} \mathbf{A}^T\mathbf{A} dxdy \right) \Delta\mathbf{x} = \iint\limits_D \tfrac{m}{2h^2} \mathbf{A}^T\mathbf{b} dxdy. \tag{A.26}$$

Interestingly, the left hand side $\iint\limits_D \frac{m}{2h^2}\mathbf{A}^T\mathbf{A}dxdy$ can be simplified to a block diagonal matrix of the form

$$\begin{bmatrix} \iint\limits_D \frac{m}{2h^2}\mathbf{I}_{3\times 3} dxdy & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \iint\limits_D \frac{m}{2h^2}(\mathbf{Rq})^T(\mathbf{Rq}) dxdy & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \iint\limits_D \frac{ms^2}{2h^2}[\mathbf{Rq}]_\times^T[\mathbf{Rq}]_\times dxdy \end{bmatrix}, \tag{A.27}$$

by noticing that $[\mathbf{Rq}]_\times^T(\mathbf{Rq}) = \mathbf{0}$. Moreover, as the center of mass of the disc is placed at the origin $\iint\limits_D m\mathbf{Rq}dxdy = \mathbf{0}$ and $\iint\limits_D m[\mathbf{Rq}]_\times = \mathbf{0}$. We can now integrate the diagonal elements leading to

$$\begin{bmatrix} \frac{\pi r^2 m}{h^2}\mathbf{I}_{3\times 3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\pi r^4 m}{2h^2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{ms^2}{h^2}\mathcal{I} \end{bmatrix}, \tag{A.28}$$

where $\mathcal{I} = \mathbf{R}\begin{bmatrix} \frac{\pi r^4}{4}\mathbf{e}^x & \frac{\pi r^4}{4}\mathbf{e}^y & \frac{\pi r^4}{2}\mathbf{e}^z \end{bmatrix}\mathbf{R}^T$ is the second moment of area of a disc in world-space. The right hand side $\iint\limits_D \frac{m}{h^2}\mathbf{A}^T\mathbf{b}dxdy$ can be simplified as

$$\begin{bmatrix} \frac{\pi r^2 m}{h}\dot{\mathbf{c}} + \boldsymbol{\xi}_{\text{ext}} \\ \frac{\pi r^4 m}{2h}\dot{s} + \boldsymbol{\tau}_{\text{ext}} \\ \frac{ms^2\mathcal{I}}{h}\dot{\boldsymbol{\theta}} + \boldsymbol{\gamma}_{\text{ext}} \end{bmatrix}, \tag{A.29}$$

where $\boldsymbol{\xi}_{\text{ext}} = \iint\limits_D \mathbf{f}_{\text{ext}} dxdy$ is the sum of the external forces which act on the disc, $\boldsymbol{\tau}_{\text{ext}} = s\iint\limits_D (\mathbf{Rq}) \times \mathbf{f}_{\text{ext}} dxdy$ is the sum of the external torques and $\boldsymbol{\gamma}_{\text{ext}} = \iint\limits_D (\mathbf{Rq}) \cdot \mathbf{f}_{\text{ext}} dxdy$ is a quantity which can

be seen as the counterpart of the external torque by measuring the external forces applied along the center direction.

Center update. By solving the linear system (A.26) for $\Delta\mathbf{c}$ we find the prediction update for the center

$$\Delta\mathbf{c} = h\dot{\mathbf{c}} + \tfrac{h^2}{\pi r^2 m}\boldsymbol{\xi}_{\text{ext}}, \tag{A.30}$$

As we integrate on a disc located at a midpoint this update is valid for the center of the disc located at this point. We approximate the update over the end points by using the same update rule.

Scale update. Similarly, the scale update can be computed solving the linear system for Δs leading to

$$\Delta\mathbf{s} = h\dot{s} + \tfrac{2h^2}{\pi r^4 m}\boldsymbol{\gamma}_{\text{ext}}, \tag{A.31}$$

We also approximate the update over the end points using the same update rule

Frame update. The frame update can be computed by solving the linear system for $\Delta\boldsymbol{\theta}$ leading to

$$\Delta\boldsymbol{\theta} = h\dot{\boldsymbol{\theta}} + \tfrac{\mathcal{I}^{-1}h^2}{s^2 m}\boldsymbol{\tau}_{\text{ext}}. \tag{A.32}$$

A.3 Correction Step

Inertia approximation. From the derivation in Appendix A.2 we can obtain an approximation of the inertia term as

$$\mathrm{E}_{\text{inertia}} \approx \tfrac{\pi r^2 m}{2h^2}\|\mathbf{c}_t - \hat{\mathbf{c}}_t\|_2^2 + \tfrac{\pi r^4 m}{4h^2}(s_t - \hat{s}_t)^2 + \tfrac{s^2 m}{2h^2}\|\boldsymbol{\theta}_t - \hat{\boldsymbol{\theta}}_t\|_{\mathcal{I}}^2, \tag{A.33}$$

where

$$\hat{\mathbf{c}}_t \;= \mathbf{c}_{t-1} + h\dot{\mathbf{c}}_{t-1} + \tfrac{h^2}{\pi r^2 m}\boldsymbol{\xi}_{\text{ext}}, \tag{A.34}$$

$$\hat{s}_t = s_{t-1} + h\dot{s}_{t-1} + \tfrac{2h^2}{\pi r^4 m}\boldsymbol{\gamma}_{\text{ext}}, \tag{A.35}$$

$$\hat{\boldsymbol{\theta}}_t = \boldsymbol{\theta}_{t-1} + h\dot{\boldsymbol{\theta}}_{t-1} + \tfrac{\mathcal{I}^{-1}h^2}{s^2 m}\boldsymbol{\tau}_{\text{ext}}, \tag{A.36}$$

are the inertial predictions for the different degrees of freedom. The variational form of implicit Euler (3.9) can then be written in the form

$$\min_{\mathbf{X}} \tfrac{1}{2h^2}\|\mathbf{X} - \hat{\mathbf{X}}\|_{\mathbf{A}}^2 + \tfrac{1}{2}\|\mathbf{W}(\mathbf{X})\|_{\mathbf{K}}^2, \tag{A.37}$$

where $\mathbf{X} = [\,\mathbf{c}_{[0]}^T, s_{[0]}, \boldsymbol{\theta}_{[.5]}^T, \mathbf{c}_{[1]}^T, s_{[1]}, \boldsymbol{\theta}_{[1.5]}^T, \cdots\,]^T$ and $\hat{\mathbf{X}}$ are vectors containing all the degree of freedoms and their predictions, $\mathbf{K}$ is a block diagonal matrix stacking the stiffness parameters scaled by the length of the piecewise elements, $\mathbf{A}$ is a block diagonal matrix stacking the inertia weights scaled by the length of the piecewise elements, and $\mathbf{W}(\mathbf{X}) = [\,\mathbf{w}_1(\mathbf{x})\ \mathbf{w}_2(\mathbf{x})\ \ldots\,]^T$ stacks the potential energy functions.

Variational Solver. To solve this optimization we can linearize the elastic potentials and write

an iterative Gauss-Newton optimization

$$\min_{\Delta\mathbf{X}} \tfrac{1}{2h^2}\|\mathbf{X}^{k-1}+\Delta\mathbf{X}-\hat{\mathbf{X}}\|^2_{\mathbf{A}}+\tfrac{1}{2}\|\mathbf{W}(\mathbf{X}^{k-1})+\nabla\mathbf{W}(\mathbf{X}^{k-1})\Delta\mathbf{X}\|^2_{\mathbf{K}}, \tag{A.38}$$

where k is the iteration number, $\mathbf{X}^k=\mathbf{X}^{k-1}+\Delta\mathbf{X}$, and we initialize $\mathbf{X}^0=\hat{\mathbf{X}}$. Since Equation A.38 is quadratic in the unknown $\Delta\mathbf{X}$, we can minimize it with a single linear solve

$$\tfrac{\mathbf{A}}{h^2}(\mathbf{X}^{k-1}+\Delta\mathbf{X}-\hat{\mathbf{X}})+\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\mathbf{K}\left(\mathbf{W}(\mathbf{X}^{k-1})+\nabla\mathbf{W}(\mathbf{X}^{k-1})\Delta\mathbf{X}\right)=\mathbf{0}.$$

However, the conditioning of this linear system is greatly dependent on how stiff are the elastic potentials. Following the optimization trick presented in [Gould, 1986], for elastic potentials with large stiffness a better option is to split the equation above as

$$\begin{cases} \tfrac{\mathbf{A}}{h^2}(\mathbf{X}^{k-1}+\Delta\mathbf{X}-\hat{\mathbf{X}})+\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\boldsymbol{\lambda}^k=\mathbf{0}, & \text{(A.39)}\\ \mathbf{K}^{-1}\boldsymbol{\lambda}^k=\mathbf{W}(\mathbf{X}^{k-1})+\nabla\mathbf{W}(\mathbf{X}^{k-1})\Delta\mathbf{X}. & \text{(A.40)}\end{cases}$$

Note that when the elastic potentials are infinitively stiff $\mathbf{K}^{-1}$ vanishes $\boldsymbol{\lambda}=[\lambda_1,\ldots,\lambda_n]^T$ becomes the vector of Lagrange multipliers. We can now reformulate (A.39) as

$$\begin{aligned}\Delta\mathbf{X}&=-h^2\mathbf{A}^{-1}\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\boldsymbol{\lambda}^k-(\mathbf{X}^{k-1}-\hat{\mathbf{X}}) && \text{(A.41)}\\ &\approx -h^2\mathbf{A}^{-1}\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\Delta\boldsymbol{\lambda}, && \text{(A.42)}\end{aligned}$$

by assuming $\nabla\mathbf{W}(\mathbf{X}^k)\approx\nabla\mathbf{W}(\mathbf{X}^{k-1})$, and where $\boldsymbol{\lambda}^k=\boldsymbol{\lambda}^{k-1}+\Delta\boldsymbol{\lambda}$ and we initialize $\boldsymbol{\lambda}^0=\mathbf{0}$. This can be proven by induction knowing that $\mathbf{X}^0=\hat{\mathbf{X}}$ and $\boldsymbol{\lambda}^0=\mathbf{0}$. By substituting $\Delta\mathbf{X}$ in the Equation A.40, we can rewrite the system of equations as

$$\begin{cases}\Delta\mathbf{X}=-h^2\mathbf{A}^{-1}\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\Delta\boldsymbol{\lambda},\\ (h^2\|\nabla\mathbf{W}(\mathbf{X}^{k-1})^T\|^2_{\mathbf{A}^{-1}}+\mathbf{K}^{-1})\Delta\boldsymbol{\lambda}=\mathbf{W}(\mathbf{X}^{k-1})-\mathbf{K}^{-1}\boldsymbol{\lambda}^{k-1}.\end{cases}$$

Therefore, $\Delta\mathbf{X}$ and $\Delta\boldsymbol{\lambda}$ can be found with a single linear solve.

A.4 Closed form pill projection

Given a pill $\mathbf{P}=\{(\mathbf{c}_1,r_1),(\mathbf{c}_2,r_2)\}$, we can compute the closest point distance of a point $\mathbf{x}$ onto $\mathbf{P}$ in close form as

$$d=\|\mathbf{x}-\mathbf{c}_1+t\hat{\mathbf{j}}l\|_2-((1-t)r_1+tr_2) \tag{A.43}$$

where $l=\|\mathbf{c}_1-\mathbf{c}_2\|_2$ is the pill length, $\hat{\mathbf{j}}=l^{-1}(\mathbf{c}_1-\mathbf{c}_2)$ is the pill versor, $\mathbf{p}_s=\mathbf{c}_1-\hat{\mathbf{j}}\left((\mathbf{x}-\mathbf{c}_1)\cdot\hat{\mathbf{j}}\right)$ the orthogonal projection onto the pill segment, $\theta=\arcsin(l^{-1}(r_1-r_2))$ is the pill slope angle, and $t=\min(\max(-l^{-1}(\mathbf{p}_s+o\hat{\mathbf{j}}-\mathbf{c}_1)\cdot\hat{\mathbf{j}},0),1)$ is the barycentric coordinate of the projection, where $o=\|\mathbf{x}-\mathbf{p}_s\|_2^2\tan(\theta)$.

Bibliography

Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2018.

A. Alexe, L. Barthe, M.P. Cani, and V. Gaildrat. Shape modelling by sketching using convolution surfaces. In *Pacific Graphics (Short Papers)*, 2005.

Dicko Hamadi Ali, Tiantian Liu, Benjamin Gilles, Ladislav Kavan, François Faure, Olivier Palombi, and Marie-Paule Cani. Anatomy transfer. *ACM Trans. on Graphics*, 2013.

Baptiste Angles, Marco Tarini, Loic Barthe, Brian Wyvill, and Andrea Tagliasacchi. Sketch-based implicit blending. *ACM TOG (Proc. SIGGRAPH Asia)*, 2017.

Andreas Antoniou and Wu-Sheng Lu. *Practical Optimization: Algorithms and Engineering Applications*. 2007.

J. L. Ba, V. Mnih, and K. Kavukcuoglu. Multiple Object Recognition With Visual Attention. In *ICLR*, 2015.

Jernej Barbič and Doug L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Trans. on Graphics*, 2005.

L. Barthe, N. A. Dodgson, M. A. Sabin, B. Wyvill, and V. Gaildrat. Two-dimensional potential fields for advanced implicit modeling operators. *Computer Graphics Forum*, 22(1):23–33, 2003. ISSN 1467-8659. doi: 10.1111/1467-8659.t01-1-00643. URL `http://dx.doi.org/10.1111/1467-8659.t01-1-00643`.

Loïc Barthe, Brian Wyvill, and Erwin De Groot. Controllable binary csg operators for soft objects. *International Journal of Shape Modeling*, 2004.

H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. *CVIU*, 10 (3):346–359, 2008.

Jan Bender, Matthias Müller, and Miles Macklin. Position-based simulation methods in computer graphics. In *Proc. Eurographics (Technical Course Notes)*, 2015.

A. Bernhardt, A. Pihuit, M. P. Cani, and L. Barthe. Matisse: Painting 2d regions for modeling free-form shapes. In *Proc. of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, pages 57–64. Eurographics Association, 2008. ISBN 978-3-905674-07-1.

Adrien Bernhardt, Loic Barthe, Marie-Paule Cani, and Brian Wyvill. Implicit blending revisited. volume 29, pages 367–376, 2010.

James F Blinn. A generalization of algebraic surface drawing. *ACM transactions on graphics (TOG)*, 1(3):235–256, 1982.

Jules Bloomenthal. *Skeletal Design of Natural Forms*. PhD thesis, University of Calgary, 1995.

Jules Bloomenthal and Brian Wyvill, editors. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997.

Mario Botsch and Leif Kobbelt. A remeshing approach to multiresolution modeling. In *Proc. of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2004.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and B Lévy. *Polygon Mesh Processing*. A K Peters, 2010.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. on Graphics*, 2014.

Sofien Bouaziz, Andrea Tagliasacchi, Hao Li, and Mark Pauly. Modern techniques and applications for real-time non-rigid registration. *Proc. SIGGRAPH Asia (Technical Course Notes)*, 2016.

R. Bridson. Fast Poisson Disk Sampling in Arbitrary Dimensions. In *SIGGRAPH sketches*, 2007.

Rodney A Brooks, Russell Creiner, and Thomas O Binford. The acronym model-based vision system. In *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1*, pages 105–113. Morgan Kaufmann Publishers Inc., 1979.

A. Buades, B. Coll, and J.-M. Morel. A Non-Local Algorithm for Image Denoising. In *CVPR*, 2005.

Neill D. F. Campbell and Jan Kautz. Learning a manifold of fonts. *ACM Trans. on Graphics (TOG)*, 2014.

Marie-Paule Cani. An implicit formulation for precise contact modeling between flexible solids. In *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 1993.

Y-W. Chao, S. Vijayanarasimhan, B. Seybold, D. A. Ross, J. Deng, and R. Sukthankar. Rethinking the Faster R-CNN Architecture for Temporal Action Localization. In *CVPR*, 2018.

Hon Fai Choi and Silvia S Blemker. Skeletal muscle fascicle arrangements can be reconstructed using a laplacian vector field simulation. *PloS one*, 2013.

Simon Clutterbuck and James Jacobs. A physically based approach to virtual character deformations. In ACM SIGGRAPH Talk sessions, 2010.

Harper Collins. Collins english dictionary. *Glassgow: HarperCollins*, 2004.

Michael Comet. Maya muscle. `http://download.autodesk.com/us/support/files/muscle.pdf`, 2011. (Accessed on Aug. 8th, 2018).

Hugh B Cott. *Adaptive coloration in animals*. Methuen; London, 1940.

Bruno de Araújo and Joaquim Jorge. Blobmaker: Free-form modelling with variational implicit surfaces. In *12th Encontro Português de Computação Gráfica*, 2003.

D. Detone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-Supervised Interest Point Detection and Description. *CVPR Workshop on Deep Learning for Visual SLAM*, 2018.

John A Endler. Natural selection on color patterns in poecilia reticulata. *Evolution*, 34(1):76–91, 1980.

Christer Ericson. *Real-time collision detection*. CRC Press, 2004.

S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, K. Kavukcuoglu, and G. E. Hinton. Attend, Infer, Repeat: Fast Scene Understating with Generative Models. In *NIPS*, 2015.

P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. *PAMI*, 32(9):1627–1645, 2010.

Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, 100(1):67–92, 1973.

Callum Galbraith, Lars Muendermann, and Brian Wyvill. Implicit Visualization and Inverse Modeling of Growing Trees. 2004.

M. Gao, A. Li, V. I. Morariu, and L. S. Davis. C-WSL: Coung-guided Weakly Supervised Localization. In *ECCV*, 2018.

G. Georgakis, S. Karanam, Z. Yu, J. Ernst, and J. Košecká. End-to-end Learning of Keypoint Detector and Descriptor for Pose Invariant 3D Matching. In *CVPR*, 2018.

Nicholas Ian Mark Gould. On the accurate determination of search directions for simple differentiable penalty functions. *IMA Journal of Numerical Analysis*, 1986.

Olivier Gourmel, Loic Barthe, Marie-Paule Cani, Brian Wyvill, Adrien Bernhardt, Mathias Paulin, and Herbert Grasberger. A gradient-based implicit blend. *ACM Trans. on Graphics (TOG)*, 2013.

Simon Green. Particle simulation using CUDA. *NVIDIA whitepaper*, 2010.

Mireille Grégoire and Elmar Schömer. Interactive simulation of one-dimensional flexible parts. In *Proc. of ACM Symposium on Solid and Physical Modeling*, 2006.

K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. DRAW: A Recurrent Neural Network For Image Generation. In *ICML*, 2015.

A. Gupta, A. Vedaldi, and A. Zisserman. Inductive Visual Localization: Factorised Training for Superior Generalization. In *BMVC*, 2018.

Thomas C Hales. The honeycomb conjecture. *Discrete & Computational Geometry*, 25(1):1–22, 2001.

John C Hart and Brent Baker. Implicit modeling of tree surfaces. In *Proc. EG workshop on implicit surfaces*, 1996.

K. He, X. Zhang, R. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. In *ICCV*, 2015.

K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *ICCV*, 2017.

G.E. Hinton, A. Krizhevsky, and S.D. Wang. Transforming Auto-Encoders. In *International Conference on Artificial Neural Networks*, pages 44–51, 2011.

Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. 2018.

Christoph Hoffmann and John Hopcroft. Automatic surface generation in computer aided design. *The Visual Computer*, 1(2):92–100, 1985.

P. C. Hsu and C. Lee. Field functions for blending range controls on soft objects. *Proc. of Eurographics, Computer Graphics Forum*, 22(3):233–242, 2003.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 1999.

Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH Fluids in Computer Graphics. In *Eurographics 2014 - State of the Art Reports*, 2014.

N. Inoue, R. Furuta, T. Yamasaki, and K. Aizawa. Cross-Domain Weakly-Supervised Object Detection through Progressive Domain Adaptation. In *ECCV*, 2018.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and J.P. Lewis. Skinning: Real-time shape deformation. SIGGRAPH Course, http://skinning.org/direct-methods.pdf, 2014a.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014b.

M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial Transformer Networks. In *NIPS*, pages 2017–2025, 2015.

J. Johnson, A. Karpathy, and L. Fei-fei. Densecap: Fully Convolutional Localization Networks for Dense Captioning. In *CVPR*, 2016.

J. Johnson, A. Gupta, and L. Fei-fei. Image Generation from Scene Graphs. In *CVPR*, 2018.

Petr Kadleček, Alexandru-Eugen Ichim, Tiantian Liu, Jaroslav Křivánek, and Ladislav Kavan. Reconstructing personalized anatomical models for physics-based body animation. *ACM TOG (Proc. SIGGRAPH Asia)*, 2016.

Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. A survey of simple geometric primitives detection methods for captured 3d data. In *Computer Graphics Forum*. Wiley Online Library, 2018.

Michael R Kaplan. The use of spatial coherence in ray tracing. In *ACM SIGGRAPH*, volume 85, pages 22–26, 1985.

Olga Karpenko, John F Hughes, and Ramesh Raskar. Free-form sketching with variational implicit surfaces. In *Computer Graphics Forum*, 2002.

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, 2007.

Timothy L Kay and James T Kajiya. Ray tracing complex scenes. In *ACM SIGGRAPH computer graphics*, volume 20, pages 269–278. ACM, 1986.

Shigeru Kondo. The reaction-diffusion system: a mechanism for autonomous pattern formation in the animal skin. *Genes to Cells*, 7(6):535–541, 2002.

Tassilo Kugelstadt and Elmar Schömer. Position and orientation based cosserat rods. In *Proc. of the Symposium on Computer Animation (SCA)*, 2016.

Tsuneya Kurihara and Natsuki Miyata. Modeling deformable human hands from medical images. In *Proceedings of the 2004 ACM SIGGRAPH Symposium on Computer Animation (SCA-04)*, 2004.

Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2009)*, July 2009.

Eric Larsen, Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Fast distance queries with rectangular swept sphere volumes. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 4, pages 3719–3726. IEEE, 2000.

Binh Huy Le and Jessica K. Hodgins. Real-time skeletal skinning with optimized centers of rotation. *ACM Trans. Graph.*, 2016.

Yann LeCun and M Ranzato. Deep learning tutorial. In *Tutorials in International Conference on Machine Learning (ICML'13)*, pages 1–29. Citeseer, 2013.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Dongwoon Lee, Michael Glueck, Azam Khan, Eugene Fiume, and Ken Jackson. A survey of modeling and simulation of skeletal muscle. *ACM Trans. on Graphics*, 2010.

J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proc. ACM SIGGRAPH*, 2000.

J.P. Lewis, Ken Anjyo, Taehyun Rhee, Mengjie Zhang, Fred Pighin, and Zhigang Deng. Star: Practice and theory of blendshape facial models. In *Eurographics*, 2014.

Duo Li, Shinjiro Sueda, Debanga R Neog, and Dinesh K Pai. Thin skin elastodynamics. *ACM TOG (Proc. SIGGRAPH)*, 2013.

Hao Li, Robert W Sumner, and Mark Pauly. Global correspondence optimization for non-rigid registration of depth scans. In *Computer graphics forum*, volume 27, pages 1421–1430. Wiley Online Library, 2008.

S. Lienhard, C. Lau, P. Müller, P. Wonka, and M. Pauly. In *Design Transformations for Rule-based Procedural Modeling*, 2017.

Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *arXiv preprint arXiv:1809.02165*, 2018.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM Trans. on Graphics*, 2015.

D.G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 20(2), 2004.

Ives Macedo, Joao Paulo Gois, and Luiz Velho. Hermite radial basis functions implicits. In *Computer Graphics Forum*, volume 30, pages 27–42. Wiley Online Library, 2011.

Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM TOG (Proc. SIGGRAPH)*, 2014.

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. XPBD: Position-based Simulation of Compliant Constrained Dynamics. In *Proc. of the International Conference on Motion in Games*, 2016.

Benoit B Mandelbrot. *The fractal geometry of nature*, volume 1. WH freeman New York, 1982.

Steve Marschner and Peter Shirley. *Fundamentals of computer graphics (4th ed.) Chapter 22 Implicit Modeling*. A. K. Peters/CRC Press, Ltd., 2015.

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. *ACM Trans. on Graphics*, 2011.

D. Merget, M. Rock, and G. Rigoll. Robust Facial Landmark Detection via a Fully-Conlolutional Local-Global Context Network. In *CVPR*, 2018.

Matthias Muller. NVIDIA PhysX SDK 3.4.0 documentation. `https://docs.nvidia.com/gameworks/#gameworkslibrary/physx/physx.htm`, 2008. (Accessed on Aug. 9th, 2018).

Matthias Müller and Nuttapong Chentanez. Solid simulation with oriented particles. *ACM Trans. on Graphics*, 2011.

Matthias Müller and Nuttapong Chentanez. Adding physics to animated characters with oriented particles. 2011.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 2007.

Matthias Müller, Jan Bender, Nuttapong Chentanez, and Miles Macklin. A robust method to extract the rotational part of deformations. In *Proceedings of the 9th International Conference on Motion in Games*, MIG '16, 2016.

A. Newell, K. Yang, and J. Deng. Stacked Hourglass Networks for Human Pose Estimation. In *ECCV*, 2016.

NVIDIA. Nvidia flex, Aug 2018. URL `https://developer.nvidia.com/flex`.

Y. Ono, E. Trulls, P. Fua, and K. M. Yi. Lf-Net: Learning Local Features from Images. In *NIPS*, 2018.

Joseph O'Rourke. Finding minimal enclosing boxes. *International journal of computer & information sciences*, 14(3):183–199, 1985.

Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

Alexander Pasko and Valery Adzhiev. Function-based shape modeling: mathematical framework and specialized language. In *Automated Deduction in Geometry, Lecture Notes in Artificial Intelligence 2930*, 2004.

Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 1995.

Alexander A Pasko and Vladimir V Savchenko. Blending operations for the functionally based constructive geometry. In *Set-theoretic Solid Modeling: Techniques and Applications*, 1994.

Galina I. Pasko, Alexander A. Pasko, and Tosiyasu L. Kunii. Bounded blending for Function-Based shape modeling. *IEEE Comput. Graph. Appl.*, 25(2):36–45, 2005.

Gerard Pons-Moll, Javier Romero, Naureen Mahmood, and Michael J. Black. Dyna: A model of dynamic human shape in motion. *ACM TOG (Proc. SIGGRAPH)*, 2015.

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomir Mech. L-systems: from the theory to visual models of plants. In *Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, volume 3, pages 1–32. CSIRO Publishing Melbourne, 1996.

J. Redmon and A. Farhadi. YOLO 9000: Better, Faster, Stronger. In *CVPR*, 2017.

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *CVPR*, 2016.

S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 2015.

A.A.G. Requicha and H.B. Voelcker. *Constructive Solid Geometry*. TM (Rochester, PAP). Production Automation Project, University of Rochester, 1977.

A Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973.

Elon Rimon and Stephen P Boyd. Efficient distance computation using best ellipsoid fit. *margin*, 1(2):1, 1992.

Jack Ritter. An efficient bounding sphere. *Graphics gems*, 1:301–303, 1990.

Lawrence G Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

A. P. Rockwood. The displacement method for implicit blending surfaces in solid models. *ACM Trans. on Graphics (TOG)*, 1989.

Marco Romeo, Carlos Monteagudo, and Daniel Sánchez-Quirós. Muscle Simulation with Extended Position Based Dynamics. In *Spanish Computer Graphics Conference (CEIG)*, 2018.

M.-A. Sabin. The use of potential surfaces for numerical geometry. *British Aircraft Corporation, Weybridge, UK, Technical Report No. VTO/MS/153*, 1968.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

Jun Saito and Simon Yuen. Efficient and robust skin slide simulation. In *Proceedings of the ACM SIGGRAPH Digital Production Symposium*, DigiPro '17, 2017.

Shunsuke Saito, Zi-Ye Zhou, and Ladislav Kavan. Computational bodybuilding: Anatomically-based modeling of human bodies. *ACM TOG (Proc. SIGGRAPH)*, 2015.

Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. SIGGRAPH '97, 1997.

Ryan Schmidt, Brian Wyvill, Mario Costa-Sousa, and Joaquim A. Jorge. Shapeshop: Sketch-based solid modeling with the blobtree. In *Proc. 2nd Eurographics Workshop on Sketch-based Interfaces and Modeling*. Eurographics, 2005.

Christian Schumacher, Bernhard Thomaszewski, Stelian Coros, Sebastian Martin, Robert Sumner, and Markus Gross. Efficient simulation of example-based materials. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*, 2012.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. In *ICCV*, 2017.

R. Sewart and M. Andriluka. End-to-End People Detection in Crowded Scenes. In *CVPR*, 2016.

Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. Example-based dynamic skinning in real time. In *ACM Trans. on Graphics (TOG)*, 2008.

Dave Shreiner, Graham Sellers, John Kessenich, and Bill Licea-Kane. *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley, 2013.

Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, 2012.

Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. In *ACM TOG (Proc. SIGGRAPH)*, 2005.

B. Singh, H. Li, A. Sharma, and L. S. Davis. R-FCN-3000 at 30fps: Decoupling Detection and Classification. In *CVPR*, 2018.

Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In *Proc. of SIGGRAPH'98*, pages 405–414, New York, NY, USA, 1998. ACM. ISBN 0-89791-999-8.

Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, 2007.

J. Spillmann and M. Teschner. Corde: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. In *Proc. of the Symposium on Computer Animation (SCA)*, 2007.

M. Sun, Y. Yuan, F. Zhou, and E. Ding. Multi-Attention Multi-Class Constraint for Fine-grained Image Recognition. In *ECCV*, 2018.

Rapee Suveeranont and Takeo Igarashi. Example-based automatic font generation. In *Proceedings of the 10th International Conference on Smart Graphics*, SG'10, pages 127–138, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13543-9, 978-3-642-13543-9. URL `http://dl.acm.org/citation.cfm?id=1894345.1894361`.

S. Suwajanakorn, N. Snavely, J. Tompson, and M. Norouzi. Discovery of Latent 3D Keypoints via End-To-End Geometric Reasoning. In *NIPS*, 2018.

Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. *Proc. Eurographics (State of the Art Reports)*, 2016.

C. Tai, H. Zhang, and J. Fong. Prototype modeling from sketched silhouettes based on convolution surfaces. In *Computer Graphics Forum*, volume 23, pages 71–83, 2004.

Danhang Tang, Mingsong Dou, Peter Lincoln, Philip Davidson, Kaiwen Guo, Jonathan Taylor, Sean Fanello, Cem Keskin, Adarsh Kowdle, Sofien Bouaziz, et al. Real-time compression and streaming of 4d performances. In *SIGGRAPH Asia 2018 Technical Papers*, page 256. ACM, 2018a.

P. Tang, X. Wang, A. Wang, Y. Yan, W. Liu, J. Huang, and A. Yuille. Weakly Supervised Region Proposal Network and Object Detection. In *ECCV*, 2018b.

Jonathan Taylor, Lucas Bordeaux, Thomas Cashman, Bob Corish, Cem Keskin, Eduardo Soto, David Sweeney, Julien Valentin, Benjamin Luff, Arran Topalian, Erroll Wood, Sameh Khamis, Pushmeet Kohli, Toby Sharp, Shahram Izadi, Richard Banks, Andrew Fitzgibbon, and Jamie Shotton. Efficient and precise interactive hand tracking through joint, continuous optimization of pose and correspondences. *TOG*, 2016.

B. Tekin, P. Marquez-neila, M. Salzmann, and P. Fua. Learning to Fuse 2D and 3D Image Cues for Monocular Body Pose Estimation. In *ICCV*, 2017.

Demetri Terzopoulos and Keith Waters. Physically-based facial modeling, analysis, and animation. *Journal of Visualization and Computer Animation*, 1990.

Jean-Marc Thiery, Émilie Guy, and Tamy Boubekeur. Sphere-meshes: shape approximation using spherical quadric error metrics. *ACM Trans. on Graphics*, 2013.

Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. In *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 2010.

Anastasia Tkach, Mark Pauly, and Andrea Tagliasacchi. Sphere-meshes for real-time hand modeling and tracking. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*, 2016.

Anastasia Tkach, Andrea Tagliasacchi, Edoardo Remelli, Mark Pauly, and Andrew Fitzgibbon. Online generative model personalization for hand tracking. *ACM TOG (Proc. SIGGRAPH Asia)*, 2017.

J. R. R Uijlings, S. Popov, and V. Ferrari. Revisiting Knowledge Transfer for Training Object Class Detectors. In *CVPR*, 2018.

Nobuyuki Umetani, Ryan Schmidt, and Jos Stam. Position-based elastic rods. In *Proc. of the Symposium on Computer Animation (SCA)*, 2014.

Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1991.

Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 2013a.

Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. Implicit skinning: Real-time skin deformation with contact modeling. *ACM TOG (Proc. SIGGRAPH)*, 2013b.

Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. Robust iso-surface tracking for interactive character skinning. *ACM Trans. on Graphics (Proc. of SIGGRAPH Asia)*, 2014.

Vital Mechanics. http://www.vital.com, 2018.

F. Wan, P. Wei, J. Jiao, Z. Han, and Q. Ye. Min-Entropy Latent Model for Weakly Supervised Object Detection. In *CVPR*, 2018.

F. Wang, L. Zhao, X. Li, X. Wang, and D. Tao. Geometry-Aware Scene Text Detection with Instance Transformation Network. In *CVPR*, 2018.

Huamin Wang, Peter J Mucha, and Greg Turk. Water drops on surfaces. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 921–929. ACM, 2005.

Denis Weaire and Robert Phelan. A counter-example to kelvin's conjecture on minimal surfaces. *Philosophical Magazine Letters*, 69(2):107–110, 1994.

Brian Wyvill, Andrew Guy, and Eric Galin. Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. In *Computer Graphics Forum*, volume 18, pages 149–158. Wiley Online Library, 1999.

Hongyi Xu and Jernej Barbič. Pose-space subspace dynamics. *ACM Trans. on Graphics*, 2016.

K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. LIFT: Learned Invariant Feature Transform. In *ECCV*, 2016.

Simon Yuen. *Personal communication.* Head of Creatures, Method Studios, 2018.

C. Zach and G. Bournaoud. Descending, Lifting or Smoothing: Secrets of Robust Cost Opimization. In *ECCV*, 2018.

S. Zhang, J. Yang, and B. Schiele. Occluded Pedestrian Detection Through Guided Attention in CNNs. In *CVPR*, 2018a.

X. Zhang, Y. Wei, G. Kang, Y. Wang, and T. Huang. Self-produced Guidance for Weakly-supervised Object Localization. In *ECCV*, 2018b.

Y. Zhang, Y. Gui, Y. Jin, Y. Luo, Z. He, and H. Lee. Unsupervised Discovery of Object Landmarks as Structural Representations. In *CVPR*, 2018c.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. In *CVPR*, 2016.

Lifeng Zhu, Xiaoyan Hu, and Ladislav Kavan. Adaptable anatomical models for realistic bone motion reconstruction. *Computer Graphics Forum (Proc. EuroGraphics)*, 2015.

Olgierd Cecil Zienkiewicz, Robert Leroy Taylor, Olgierd Cecil Zienkiewicz, and Robert Lee Taylor. *The finite element method*, volume 36. McGraw-hill London, 1977.

Ziva Dynamics. Ziva dynamics. https://ziva.com, 2018.