LEARN TO SYNTHESIZE APPEARANCE, SHAPE AND MOTION USING SYNTHETIC DATA

by

Guilin Liu
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Jyh-Ming Lien, Dissertation Director
_____	Dr. Jana Kosecka, Committee Member
_____	Dr. Yotam Gingold, Committee Member
_____	Dr. Qi Wei, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Summer Semester 2017
George Mason University
Fairfax, VA

Learn to Synthesize Appearance, Shape and Motion Using Synthetic Data

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Guilin Liu
Bachelor of Engineering
Wuhan University, 2012

Director: Dr. Jyh-Ming Lien, Professor
Department of Department of Computer Science

Summer Semester 2017
George Mason University
Fairfax, VA

Acknowledgments

For the past few years of my PhD study, I was fortunate to work with and get help from a lot of brilliant people. I would like to thank all the people who help during my PhD study.

Firstly, I would like to thank my advisor professor Jyh-Ming Lien for making me interested in shape analysis topics and at the same time giving me a lot of freedom to explore researches in deep learning, computer vision and other applied machine learning topics. I also deeply appreciate his patience, advices and supports all the way through my PhD study. Without his help, this dissertation would not have been possible. I also would like to thank professor Yotam Gingold, professor Jana Kosecka and professor Qi Wei for being my committee members and giving me a lot of insightful comments. It is always with a lot of fun to discuss and collaborate with professor Gingold. I also learned a lot from professor Kosecka about computer vision research through her seminar course and paper reading group meeting. Professor Wei gave me a lot of advices about how to do a good academic presentation and collaborate with academic colleagues.

I also want to express my sincere thanks to my internship mentor and collaborator Dr. Duygu Ceylan from Adobe. I had a lot of fun working with her. She is always working hard, thinking for others and encouraging. I also want to thank other researchers from Adobe including Dr. Jimei Yang, Dr. Ersin Yumer, Dr. Xin Sun, Dr. Zhili Chen and Dr. Nathan Carr for providing internship mentoring and fruitful discussions.

During the past few years, I was fortunate to work with a lot of other amazing researchers including Dr. Zhonghua Xi, professor Qixing Huang (UT Austin), professor Etienne Vouga (UT Austin), Hang Chu (U Toronto), Chao Yang (USE) and Zimo Li (USC) and Dr. Christian Langevin (USGS). I also want to take this opportunity to thank my other friends who provided a lot of advices, including Chunyuan Li (Duke), Liwen Hu (USC), Eunbyung Park (UNC), Weilun Sun (UC Berkeley), Xi Chen (Microsoft), Qingming Tang (TTIC), Xing Zhou (GMU) etc.

I would like to thank all the members from MASC group and my friends at GMU as well for the support, accompanying and advices, including Zhonghua Xi, Yanyan Lu, Christopher Vo, Evan Behar, Yue Hao, Ermo Wei, Arsalan Mousavian, Huangxin Wang, Jianchan Tan, Songrun Liu, Qi Xing, Qi Li, Chuan Yan, Haoliang Wang, Ming Zhang, Mengbai Xiao, Li Liu, An Wang, Xue Yin etc.

Finally, I want to thank my parents, sister, grandfather and uncles for their love, encouragement and endless support.

Table of Contents

List of Tables

List of Figures

Abstract

LEARN TO SYNTHESIZE APPEARANCE, SHAPE AND MOTION USING SYNTHETIC DATA

Guilin Liu, PhD

George Mason University, 2017

Dissertation Director: Dr. Jyh-Ming Lien

A vast amount of 2D images and 3D meshes and points are created every day. Many applications require us to extract more semantic information beyond the original raw discrete representation of pixels, facets and points from these data. In this dissertation, I will focus on extracting several types of shape and physical properties from these data. However, estimating these property information suffers from some difficulties, including under-constrained setting, missing accurate ground truth data and expensive computation cost. I will develop methods to solve three tasks of estimating shape and physical properties with each task representing one of the difficulties. The tasks are appearance synthesis, shape synthesis and motion synthesis. For the appearance synthesis, I will develop an end-to-end deep learning framework to estimate the material (reflectance property) from 2D images, which is originally an under-constrained problem. The main ingredient of the end-to-end framework is the introduction of a rendering layer. I will show the effectiveness of framework for editing the materials in 2D images. For shape synthesis, I will discuss how to combine the inaccurate and noisy ground truth normal data and the image itself to predict fine-scale

normal from 2D images using the deep learning framework. Results will show that even though the ground truth normal is inaccurate and far from detailed, the trained deep learning model can still produce detailed normal predictions. The motion synthesis part will be about approximating the medial axis of a robot's configuration space. Originally, computing the medial axis of the configuration space is highly computationally expensive. I will show how the formulation of support vector machine can be adapted to solve this task efficiently. Detailed explanation of how the difficulties are resolved and plenty of experiment results will be provided. On the other hand, the methods for these tasks require sufficient and valid training data. I generate synthetic datasets to compensate for the lack of corresponding real image datasets for appearance synthesis and shape synthesis. Semantically meaningful segmentations of 3D shapes are used to generate plausible synthetic datasets for these two tasks. To train the model to approximate the medial axis of the robot's configuration space well, the segmentation with bounded geometric constraints of the 3D models in the environment is needed. To generate the segmentations which are semantically meaningful and with bounded geometric constraints, I will propose a new part-aware shape feature and two nearly convex decomposition methods. Comparisons with human segmentation and other alternatives will validate the effectiveness of the proposed feature and methods.

Chapter 1: Introduction

1.1 Understanding Shape and Physical Properties

During the past several decades, researchers have successfully advanced the design of various sensors to capture 2D and 3D data from the real world. These sensors include digital cameras, laser scanners, stereo cameras, etc. As these sensors are becoming more and more affordable for general public, billions of 2D images and millions of 3D meshes and point cloud data are created everyday.

However, images and point clouds (meshes) are only represented with a sequence of discrete pixels and points (facets); for most machine-based operations or interactions, these low-level discrete representations are not enough. For example, image relighting has to estimate physical shapes and relative reflectance properties among pixels and preserve them before and after the relighting operation while the naive relighting usually change all the pixel values with the same ration, which gives unrealistic output; navigation through an environment using the captured 3D data also depends on a deep understanding of the spatial relationship among the objects in the scene; inserting a new object into an existing image for virtual reality applications requires the cue of global layout of object in the scene. All these tasks requires us to go beyond the discrete raw representations of the 2D and 3D data and to have a better understanding of the data at semantic level.

Two types of such semantic information are shape property and physical property. Some representative types of the shape properties are normal directions, symmetrical correspondence, medial axis, topological connectivity, etc. In computer vision and graphics, researchers have been working on two kinds of physical properties: reflectance property and dynamics property. Reflectance property captures the relationship between incoming and

Figure 1.1: Image composition example. Left: editing with wrong estimation of the ball's shape and physical properties; Right: editing with better estimation of the ball's shape and physical properties. (Image credit: Adobe Project Felix Video)

out-coming light. It determines the appearance of the images captured by sensors and seen by human. Dynamics property characterizes how objects move and deform given gravity, external forces and other physical constraints. Physical property is also known as material property. To well understand these shape and physical properties from either 2D or 3D captured data remains challenging.

Extracting and understanding those shape and physical properties from raw data lead to a lot of applications in machine-based interactions with the captured data or the physical environment itself. The machine-based interactions include image editing, image compositions, image semantic segmentation and labeling and image based reconstruction from 2D images, 3D model extraction and labeling and semantic-level manipulation and deformation for 3D data, robot localization, robot navigation, path planning etc. Those property information usually plays a vital role in the success of the interaction operations. Take image editing for example. Image editing is with the great demand ranging from common smart phone users to professionals. However, image editing usually ends up with unrealistic artifacts, like wrong details, or that the edited appearance doesn't correspond to the appearance of any object or scene. Figure 1.1 is an example of image composition without and with good estimation of the object's shape and physical properties. On the other hand, when those editing and operations are armed with better estimated shape property and

physical property information, more realistic and plausible results can be generated. For instance, knowing the reflectance property enables the objects to know better how their color (appearance) needs to change according to the illumination changes during image editing. Another example of interactions is to let the human-designed robot or machine to conduct several specific tasks in the environment that has been re-created in the virtual world using the captured data. For this kind of interaction, full understanding of the environment's shape property is also the key to the success. Motion planning is one of them. Motion planning refers to planning the path for the robot to move from the starting configuration to the target configuration without colliding with obstacles, self-intersection or violating other constraints. As an application of motion planning, delivery robot requires the geometric information of the environment to plan a safe path to the delivery destination without colliding with the obstacle where no road exists while some other information like GPS can only helps the navigation on the existing road system.

Existing methods in estimating physical and shape properties usually suffer from several difficulties. The major difficulties include but not limit to:

1) the problem setting itself is under-constrained;

2) there is no ground truth data, or the ground truth data is not accurate enough;

3) the computation is time-consuming.

Most estimation tasks contains one or several such difficulties. In this dissertation, I would like to explore three topics in estimating physical and shape properties. Each topic will be a representative task for one of those difficulties. Although each topic will focus on one difficulty, it doesn't mean that the topic has only one of those difficulties. Instead, my objective is to use one topic for each difficulty as the representative task to show how the difficulties can be addressed. Specifically, I will study the following three topics:

1) **Appearance Synthesis**: understanding more robust reflectance property (physical property) from 2D images and showing how it can be used for realistic material transfer tasks;

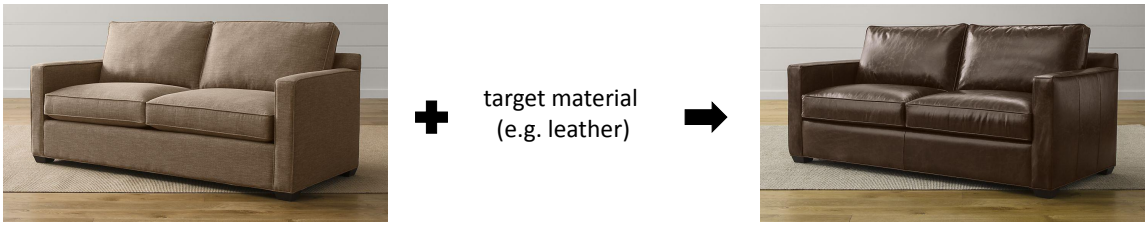


Figure 1.2: Changing material on 2D image.

2) **Shape Synthesis**: extracting more detailed normal information (shape property) from 2D images and exploring its various applications;

3) **Motion Synthesis**: sampling the medial axis (shape property) of a robot's configuration space efficiently and utilizing it for faster and safer motion planning solutions.

Each of these tasks will be the representative task for studying one difficulty with: **appearance synthesis** representing the issue of under-constrained setting, **shape synthesis** representing the issue of lack of accurate ground truth data and **motion synthesis** representing the issue of expensive computation cost. The goals of this dissertation lie in not only that I want to present new approaches to solve those three original challenging problems, but also that I hope the solutions presented in this dissertation would inspire and stimulate future research in exploring similar methods to estimate other shape and physical properties and address additional difficulties. The corresponding works are also discussed and published in papers [1, 2, 3, 4, 5].

1.2 Appearance Synthesis

Problem Setup. One type of an object's physical properties that contributes to its appearance is reflectance property. When light reaches an object, reflectance property will determine the directions and intensities of the outgoing light. The final images seen by

human eyes or cameras are the results of interaction among the objects' geometry, lighting and reflectance property. Reflectance property is also called material. In the rest of this thesis, the term reflectance property and the term material will be used interchangeably. Many designers want to change such property of the objects on 2D images or 3D models. In the case of 2D images, there is such a common demand for designers: given an image, for instance a fabric sofa, shown in Fig 1.2, how could a new sofa image be synthesized such that the geometry and lighting are consistent with the input image while the material becomes leather? I will mainly focus on the material property analysis and material editing tasks fron a single 2D image.

Difficulty. Those two tasks are very challenging as usually only the images of the object itself are provided and there is no information about the objects' geometry, lighting conditions or the original material property. As the first step, the material property needs to be inferred from images. As the images are the results of interaction among the geometry, lighting and material property, typically the first step needs to infer those information from the images. However, this problem is highly ambiguous since multiple combinations of those components may result in the same image. The quality of the estimation of each component is also highly correlated to that of other components. Those issues make the material property estimation from a single image be a highly under-constrained problem. Many previous works have tackled this ambiguity either by assuming that at least one of the intrinsic properties is known [6] or devising priors about each of these properties [7]. The recent success of deep learning methods, has stimulated research to learn such priors directly from data.

Proposed Method Overview. To apply deep learning approaches for the material estimation problem, a feasible approach is to predict each of the intrinsic properties independently from an input image, i.e. using individual neural networks to predict normals, material, and lighting. However, the interaction of these properties during the image formation process inspires us to devise a joint prediction framework. An alternative common practice of deep learning methods is to represent the intrinsic properties in a latent feature

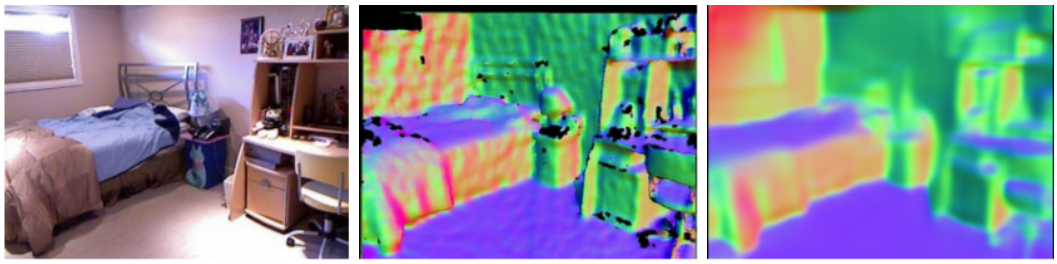


Figure 1.3: Left: input 2D image; Middle: noisy normal from ground truth depth data; Right: blurry normal from using the method from [9].

space [8] where an image can be *decoded* from implicit feature representations. Directly editing such feature representations, however, is not trivial since they do not correspond to any true physical property. On the contrary, the physics of image formation process is well understood and motivates us to replace this *black-box decoder* with a *physical decoder* and thus enables the network to learn the physics of image formation.

I will present an end-to-end network architecture that replicates the image formation process in a physically based rendering layer. The network is composed of prediction modules that infer each of the intrinsic properties from a single image of an object. These predictions are then provided to a rendering layer which re-synthesizes the input image. The loss function is defined as a weighted sum of the error over the individual predictions and perceptual error over the synthesize images.

1.3 Shape Synthesis

Problem Setup and Difficulty. Normal direction is one type of shape property, which captures the local surface details of objects. It plays a vital role in geometry reconstruction, visual understanding, virtual reality etc. Normal information is usually obtained through either laser range sensors or estimated from 2D images. However, normal information from those sources is usually noisy or blurry. In the case of range sensors as source, normal is computed from the capture depth data or point cloud data. Nevertheless, there are two

issues for this source: 1). the resolution is usually low compared to the resolution of the images, which sometimes would miss some important local details; 2). for the regions with challenging materials, like mirror, the depth data or point cloud data of those regions are either missing or not correct. In the case of estimating normal from 2D images, estimating normal information from 2D images relies on 3D model fitting or learning based reasoning approaches. The performance of model fitting highly depends on whether the accurate models can be retrieved from the database or generated through deformation. Both of these two options are challenging problems. Moreover, those model fitting approaches work poorly for scene images. The current methods of predicting normal from 2D images ofter produce blurry output, like the deep neural network based approaches. For example, in Figure 1.3, the left image is the 2D image captured by camera and the middle image show the noisy ground truth normal computed from the depth data. Both the image and depth data are from [10]; the right image shows the current state-of-art normal estimation, which is blurry, using the approach described in [9]. Other photometric stereo based methods needs multi-view images of the same scene or objects as input, which doesn't hold for most scenarios. Those noisy and blurry issues have limited a lot of applications which prefers fine-scale accurate normal information.

Propose Method Overview. In this dissertation, I will present a framework to predict accurate fine-scale normal information from a single 2D image using deep neural network. Different from previous deep learning based normal prediction works, this framework would not only penalize the difference between the predicted normal and ground truth normal, but also add the regularization of whether the images generated using the estimated normal are matched with the original input images. I will use the similar formulation as Section 1.2, but will focus more on detailed normal prediction, show results on real scene images as well and discuss how to train such frameworks efficiently.

1.4 Motion Synthesis

Problem Setup and Difficulty. One main goal of robotics research is to design methods to conduct some tasks through movements and other interaction operations with the environment. The motion planning is about the process of breaking down the desired movement task into discrete motions satisfying movement constraints or sometimes with the optimization in some aspect. Two well-known approaches towards solving this problem are PRM [11] and RRT [12]. Medial axis, one type of shape properties, is a set of points that locally maximize the distance to the boundary or objects. Medial axis is originally referred to as the topological skeleton. It has a lot of applications in shape recognition, shape segmentation, topology based transform, other shape analysis and topology analysis tasks. Medial axis is also an important structure in robotic motion planning [11, 13]. Computing the medial axis of robot's configuration space and sampling more configurations on the medial axis will encourage the maximum clearance and distance from the obstacle. Having such high clearance often creates safer paths and offers higher possibility to find a path in complex environment where the feasible configuration space only occupies a small proportion of the whole space. However, computing the medial axis can be very time-consuming in high dimensional space, which is common for robot's motion planning problems where the joints and articulation usually cause the problem to be with high degree of freedom. Hence, instead of computing the exact medial axis, people try to compute the approximate medial axis.

One type of approximation is sampling. Wilmarth et al. [14] investigated a method, called Medial Axis Probabilistic Road Map (MAPRM), which retracts configurations, free of collision or not, to the medial axis of free configuration space. Because configurations in collision with obstacles can also be pushed to the medial axis, MAPRM provably increases the probability of generating samples in the narrow passage. Lien et al. [15] extended MAPRM to high dimensional spaces by approximating the clearance and penetration depth with a large number of random query rays. Both methods do not provide any guarantee regarding

the distribution of samples on the MA. Recently, Yeh et al. [16] presented a method named Uniform Medial-Axis PRM (UMAPRM) that can uniformly samples the MA, thus provide better coverage than MAPRM.

Although these methods effectively addressed the issue of the "narrow passage problem," which means only a very small proportion of the whole space is valid for the final task, they all rely on expensive geometric computation to approximate clearance and penetration depth, particularly in high dimensional space. These expensive proximity queries hinders further improvement of these MA-based sampling strategy. For example, it is highly desirable that the edges connecting the sample can also be on or near the MA to increase both roadmap connectivity and path clearance. However, the edges by these MAPRM-variants are usually not on the MA because making a connection on the MA between a pair of configurations requires a local planner that retracts every discretization of the connection to the MA. This local planner is prohibitively slow for most motion planning problems.

Proposed Method Overview. In this dissertation, I will propose a solution that allows a configuration, free of collision or not, to be efficiently retracted to the MA. The proposed method achieves this without excessively expensive geometric proximity queries. The main idea is based on an observation that constructing the MA from a set of obstacles can be viewed as a multi-class training and classification problem. Each obstacle is regarded as a label and the labels of samples in the configuration space (C-space) is determined according to which obstacle is the closest. Under this setting, the MA of the free C-space can be viewed as the classification boundary. Therefore, approximating the MA can be transformed to finding the hyperplane that has the largest distance to the nearest training data point of any-class (known as the functional margin).

1.5 Machine Learning Approaches and Data Preparation

As discussed briefly above, machine learning models will be used as the main tools to address the difficulties and solve those tasks. Specifically, *Support Vector Machine (SVM)* and *Deep*

Neural Network are the good fits to tackle the challenges involved in those tasks.

1.5.1 Support Vector Machine and Deep Neural Network

Support Vector Machine [17], especially when armed with kernel tricks, have been widely used for the past few decades. The success and popularity of SVM largely owe to the solid but intuitive mathematical background of its formulation function: the maximum margin idea which represents extracting the plane which can separate the positive dataset and negative dataset while maximizing the closest distance between the plane and any data point from the positive set and the closest distance between the plane and any data point from the negative set. This is similar to extract a corresponding hyper-plane with specific geometrical properties in the training data's geometrical space. This connection motivates us to explore the possibilities in adapting the solid mathematical formulation and efficient optimization technique of Support Vector Machine to geometric problems which shares some similar properties or formulations. Especially, I made the observation that the maximum margin hyper-plane shares some similar spirits with the medial axis, which also defines the region having the maximum distance to the two closest objects or elements.

Recently neural network has regained its popularity due to explosion of the available training data, fast-development computation speed and effective training strategies. It has achieved great breakthrough in image classification [18], object detection [19], semantic segmentation [20], speech recognition [21] etc. These successes have a common setting: the problem originally is under-constrained problem and needs a lot of priors while those priors are difficult to express explicitly. Taking the image recognition as example, asking computer to recognize the object in the image with a single image itself without looking at other images is usually impossible. In this scenario, priors are needed to fuel the system. Nevertheless, encoding these priors by hand is difficult as there are many varieties in object color, rotation, translation, size, species etc. The same scenario applies to the task of extracting physical properties from images. Extracting physical property from a single image (view), or even several images (views), is under-constrained as well due to the fact

that physical property captures the global distribution of how the appearance or deformation state changes according to the changes in other ingredients, like light intensity, light direction, surface orientation, external force strength, etc. While, a single image is just single discrete appearance shot (view) of such distribution given the fixed lighting, surface orientation and other surrounding environment. Priors are required for this type of problem. Similarly, estimating normal directions, the 2.5D structure, from a single 2D image is also an ambiguous problem. Priors will be need to build the mapping from pixels in the images to normal directions. From a series of recent advance in different fields, deep neural network, with huge amount of training data to optimize each layer's parameter, has been shown to be able to capture those priors that are hidden in the training dataset robustly.

One prerequisite of applying machine learning approaches is to obtain *feasible and sufficient training data*. In other words, enough valid and feasible training data needs to be collected for the selected machine learning models. Few training data will lead to premature convergence and over-fitting. Some data-augmenting tricks like random cropping [18] only increase a small amount of translation invariance. The large number of parameters in machine learning system requires more varieties in the training data. AlexNet [22] has 60 million+ parameters; VGGNet [23] has 130 million+ parameters. The starvation of training data becomes more serious as people try to apply machine learning to more complex problems. One way which has been shown to be successful to obtain sufficient data to reduce over-fitting is to first synthesize and simulate the training data and later, if necessary and possible, fine-tune the trained model using real training data, which usually has smaller size. For example, for vision tasks, regardless of some possible artifacts, the data generated by rendering the 3D models has shown to be successful in acting the role of training data [24, 25, 26].

1.5.2 Data Preparation

Obtaining feasible and sufficient training data plays a vital role in the success of applying machine learning approaches for various tasks. This section will mainly consider two issues

of training data: *feasibility* and *sufficiency*. The *feasibility* is the main concern when adapting the Support Vector Machine to analyze the geometrical property of the configuration space. Support Vector Machine relies on the "binary partition" between the "positive set" and "negative set". When adapting Support Vector Machine to extract the medial axis, specifically medial axis of a robot's configuration space, one needs to define the equivalent sets to the "positive set" and "negative set". In this way, the geometrical objects or boundaries need to be separated into several parts. The separation lies in both the region between disconnected objects and the concave region of a single object. For the disconnected objects, the separation are already given; for the objects with concave parts, **segmentation methods** are required to separate the concave object into convex parts. The medial axis among the decomposed convex parts contains the medial axis of the original concave regions.

The *sufficiency* issue dominates the main concerns for the training data of appearance ans shape synthesis. Nowadays, people have labeled a lot of dataset for some tasks like image classification, image segmentation, object detection etc. However, there are few dataset which labels the physical properties for the objects in the image. Even for some available labeled dataset of physical properties, like the reflectance property, the labels sometime are just the text description based reflectance category information, not the accurate numerical representations. One way which has been shown to be successful is to first synthesize and simulate the training data and later fine-tune the trained model using the few real data. Dosovitskiy et al. [24] rendered 3D chairs models using different views as the training data for training a model which would be able to synthesize an image given certain descriptions (class, view and transformation parameters). Su et al. [25] render images using different camera poses to train a model which is capable to have a favorable estimation of camera poses from a random image. For motion tasks, Mnih et al. [26] generated synthetic data running a lot simulations to learn how the game agent operates in Atari game. Similarly, the training data will be synthesized for appearance synthesis and shape synthesis. Specifically, 3D models, material dataset, illumination dataset etc will be used to render images and transfer the corresponding accurate geometric information, material information and

lighting information onto the images. During this process, there is also a challenging issue that when synthesizing multi-material images, how to generate plausible images with different material in a single images? Naively assigning different materials for each pixel independently would render almost noisy images, which are meaningless and totally different from the real images. Instead, 3D models will first be segmented into meaning parts and different materials are assigned part-wisely for the rendering. For example, given a 3D chair model, it is usually preferred to decompose the chair into the parts like legs, frames, seat and backs, and maybe assign one material to legs and frames and another material to the seat and back. Unfortunately, most of the times, such part information doesn't come with the model itself.

1.5.3 Segmentation Methods for Data Preparation

Those issues discussed above requires us to design good **segmentation approaches** to prepare high-quality training data. To adapt the Support Vector Machine for medial axis, segmentation approaches are needed to partition the geometric elements into the *feasible* equivalent "positive set" and "negative set". To train Deep Neural Network models for appearance synthesis and shape synthesis tasks well, segmentation approaches will be needed to decompose the 3D shapes into parts, which will be used together with per-part material assignment to provide *sufficient* training dataset with enough varieties.

However, designing good segmentation approach is not an easy task, either. The challenges include: the segmentation results usually needs to be both perceptually meaningful and satisfying some bounded geometric properties; there are sometimes geometric noises in the input shapes; many previous segmentation methods rely on many unintuitive user parameters. For example, as mentioned in Section 1.5.1, partition of "positive set" and "negative set" requires the segmentation results to meet with the geometric constraint to reduce the errors introduced by training data; decompositions for material synthesis prefers the segmentation to be perceptually meaningful which make the final rendered images with part-wise material assignment to be more consistent with the real world's multi-material

images.

In this dissertation, I will propose a new 3D shape part-aware feature called Continuous Visibility Feature (CVF) and two nearly convex decomposition methods. it will show that CVF better encodes the surface and part information of mesh than the tradition line-of-sight based visibility. For example, it will be shown that existing segmentation algorithms can generate better segmentation results using CVF and its variants than using other visibility-based shape descriptors, such as shape diameter function [27]. Similar to visibility and other mesh surface features, continuous visibility would have many applications. Traditionally, developing shape segmentation which is both fast and similar to human segmentation requires expensive computation and tedious and unintuitive parameter settings; also many such methods would not be able to satisfy some bounded geometric constraints. I will propose a 2D shape decomposition method which is robust to noises and shape holes and can generate segmentation results which will be matched with the human segmentation results collected from Amazon Mechanical Turk and ensure each part has bounded concavity. Similarly, I will propose another method to segment a 3D model into several parts which are also close to the human segmentation results [28] and with bounded concavity. Both methods will take a single user parameter, concavity tolerance, as input. The number of the user parameters is fewer than most, if not all, other alternatives.

1.6 Overview of Thesis

Figure 1.4 shows an overviews of my dissertation and the relations among different parts of this dissertation. Specifically, taking the 3D and 2D shape data as input, the first part, **Data Preparation** will segment them into *perceptually meaningful* parts with *bounded concavity* with few user input parameters. The *perceptual meaningfulness* is important to simulate the multi-material images which would be closer to real images; while, the *bounded concavity* helps to improve the quality of medial axis. The simulated multi-material images will be used to train the **Appearance Synthesis** model and **Shape Synthesis** model. Those two

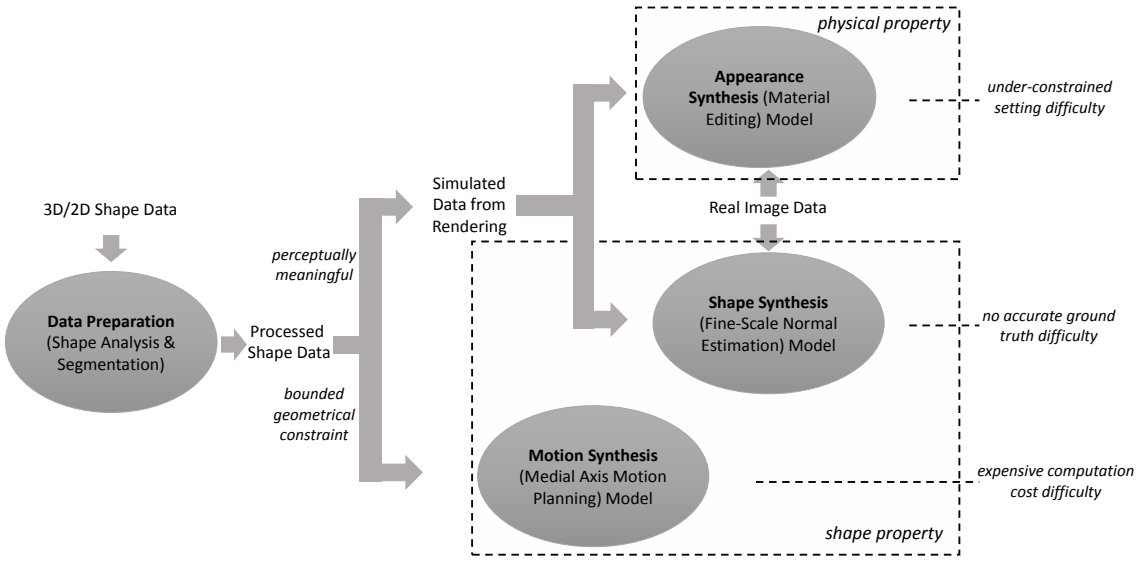


Figure 1.4: Overview of this dissertation.

models will be later fine-tuned in real image data. On the other hand, the segmented results will also be directly used to train the **Motion Synthesis** model. Appearance Synthesis part represents my effort towards extracting *physical property* problem where the problem itself is under-constrained; Shape Synthesis and Motion Synthesis are the representative tasks for *shape property* estimation. Shape Synthesis needs to tackles the difficulty of missing accurate ground truth data issue; Motion Synthesis has to address the difficulty of expensive computation cost.

1.6.1 Contributions

In this dissertation, I will design new approaches to explore one type of physical property and two types of shape properties: material property, fine-scale normal direction and medial axis. I will also show how those estimated properties can be used for image editing and motion planning tasks. The contributions of this work are as follows:

- In data preparation part, I propose a new 3D shape feature which can capture the part information robustly and is useful for various shape analysis tasks.

- In data preparation part, I also propose two nearly convex decomposition methods to segment the concave objects into nearly convex parts. The segmentation process is faster than previous approaches and the segmentation is similar to human segmentation with fewer user parameters as input.

- In appearance synthesis part, I present the first end-to-end network architecture for image-based material editing that encapsulates the forward image formation process in a rendering layer. Compared with previous material editing works, my method achieves more realistic material editing results with less assumptions.

- In appearance synthesis part, I also present the first differentiable rendering layer that supports both diffuse and specular materials and utilizes an environment map to represent a variety of natural illumination conditions. This layer can trivially be integrated into other networks to enable in-network image synthesis and thus boost performance.

- In shape synthesis part, I develop a framework that can generate much more detailed normal predictions than previous works.

- In motion synthesis part, I show a method that can generate the approximate medial axis of robot's high-dimensional configuration space efficiently, and how it can speed up and improve the motion planning solutions.

1.6.2 Outline

The rest of this thesis is organized as follows:

- In Chapter 2 (and paper [1, 2, 4]), I explain how to prepare the data for the training stage for those tasks.

- In Chapter 3 (and paper [5]), I introduce an end-to-end network architecture to estimate the reflectance property from 2D images and perform material editing tasks.
- In Chapter 4, I focus on the fine-scale normal prediction from 2D scenes images.
- In Chapter 5 (and paper [3]), I show how the approximate medial axis can be computed efficiently and used for faster motion planning.
- In Chapter 6, I will make a summary of this dissertation.

Chapter 2: Data Preparation

In this chapter and papers [1, 2, 4], a new shape feature called Continuous Visibility Feature (CVF), a 2D nearly convex decomposition approach DuDEand a 3D nearly convex decomposition approach CoRiSEare presented. CVF is a part-aware feature and can be used to generate semantically meaningful decomposition. DuDEand CoRiSEaim at decomposing shapes into parts with bounded geometric constraints.

These semantically meaning decomposition results and decompositions parts with bounded geometric constraints will be used in the following Chapters to produce training data to train the methods for appearance synthesis task, shape synthesis task and motion synthesis by using the decompositions results directly or further simulating rendering based on them. The motion planning part requires the nearly convex parts to train the machine learning that can approximate the medial axis. The decomposition results will also be directly used simulate multi-material rendering images. To render such multi-material images, different materials will be assigned part-wisely. These parts are obtained using CVF, DuDEand CoRiSe. While these decompositions methods are used to generate the large amount of training data for machine learning models, less human interactions are preferred due to the huge quantity fact. One advantage of using CVF, DuDEand CoRiSEfor decomposition is that less user parameters are needed than previous methods. CVF naturally encodes the part-aware information and two globally fitted parameters are needed for the soft-clustering nearing facets with similar CVF values. DuDEand CoRiSErequire a single parameter, concavity tolerance.

CVF provides stronger visibility measure by considering the continuously visible region for a vertex or facet. CVF will be shown to encode the surface and part information of mesh better than the traditional line-of-sight based visibility. It also be shown that existing segmentation algorithms can generate better segmentation results using CVF and its variants

than using other visibility-based shape descriptors, such as shape diameter function. The comparison are conducted on the public segmentation benchmark [28] and the comparison result would show that segmentation results using CVF is close to human segmentation results, which means CVF can be used to generate closely semantically meaningful parts.

There are also a lot of scenarios where decomposition algorithms are required to ensure that the decomposed parts would satisfy bounded geometric constraints. These bounded geometric constraints usually allows humans to have better quality or error control when the decomposed results are used for other applications. 2D and 3D nearly convex decomposition methods, DuDEand CoRiSEtake a concavity tolerance as a single user parameter and generate segmented parts with concavity same or less than the concavity tolerance. The decomposition algorithms are fast and decomposition results are close to human segmentation as well.

2.1 Related Work

2.1.1 Shape Decomposition

Shape decomposition has been extensively studied theoretically and experimentally in areas including computer graphics, computational geometry [29], computer vision and pattern recognition. In computational geometry, researchers are traditionally interested in creating decompositions subject to some optimization criteria, such as a minimum number of convex components [30, 31, 32, 33, 34], to ease the future shape analysis tasks and applications. In pattern recognition and computer vision, shape decomposition is usually a step toward shape recognition. For instance, Siddiqi and Kimia [35] use curvature and region information to identify *limbs* and *necks* of a polygon and use them to perform decomposition. Intensive approaches have been proposed for 2D decomposition. Liu et al. [36, 37] and Ren et al. [38] have been proposed to improve [39] to create fewer and more natural nearly convex shapes. Both methods [36, 38] use mutex pairs to enforce the concavity constraint. Points p_1 and p_2

form a mutex pair if their straight line connection is not completely inside the given shape. Their focus is on separating all mutex pairs with concavity-based weights larger than a user-specified threshold. Liu et al. [36] used linear programming to compute decomposition with minimum cost. Short cut rule is the only rule considered for evaluating the cost of a cut. Ren et al. [38] applied a dynamic subgradient-based branch-and-bound search strategy to get minimum number of cuts. Other than short cut rule, [38] also took minima rule into consideration which is that a cut resolving at positions with greater negative curvatures is preferred. Similarly, Juengling and Mitchell [40] formulate decomposition of a polygon as an optimization problem and applies dynamic programming to find the optimal subset of cuts from all possible cuts. The objective functions used for optimization favors short cuts that create dihedral angles close to π. Mi and DeCarlo [41] propose to decompose shape into elliptical regions glued by a hyperbolic patches. Many methods have been developed to decompose 3D mesh models as well. Comprehensive surveys can be found in [28, 42, 43].

Recently, more techniques using data-driven approach are proposed. These methods usually provide more consistent segmentations over a set of models [44] and produce segmentations that are more natural via machine learning approaches [45, 46]. However, these methods are computationally intensive (require hours of computation) and are not suitable for interactive use.

While many algorithms focus on decomposing a model into visually meaningful parts, other algorithms focuses on partitioning models into geometric primitives such as ellipsoids [47] and convex objects [39]. In other words, multiple primitives are used to jointly approximate the original shape.

2.1.2 Shape Features and Their Applications for Segmentation

Many of the existing single-shape segmentation methods are based on clustering mesh faces, such as k-means clustering [48], fuzzy clustering [49], mean-shift clustering [50], and spectral clustering [51]. Shape features, such as geodesic distance, local concavity, curvature,

have significant impact on these clustering methods. More advanced features include shape diameter function [27] that is a measure of the diameter of the object's volume in neighborhood of a point, Mumford-Shah model [52] that measures the variation within a segment and continuous visibility feature [2] that uses more restricted visibility to better capture shape than the traditional line of sight visibility. Methods that are not clustering based do exist. For example, the method proposed by Wang et al. [53] uses the training segmentation of 2D projection images. Random cuts method [54] uses other different segmentation algorithms with various parameters to generate a collection of segmentations to find consistent cut positions.

The core of these clustering based approaches lies in extracting good shape features from 3D models. These shape features can be roughly classified into per-vertex/facet features and global signatures. However, It should be pointed out that some global signatures are also generated from per-vertex/facet features via statistical approaches. Osada et al. [55] introduced Shape Distributions to capture the statistical information of vertices' and facets' shape function values. Shape matching is achieved by measuring the similarity between two shapes' distribution. Even though the feature function used in [55] is simple, it paves the way for many research works in shape matching, shape retrieval and other shape analysis tasks.

Features can be further classified into several categories depending on whether they are invariant to translation, scaling, and deformation. Some features are designed to be invariant to rigid transformation, e.g., spin image [56] and shape context [57]. Invariance to deformation has attracted more attention. For example, geodesic distance is used to build deformation invariance features in a multi-dimensional scaling based approach [58] and spectral domain analysis [59]. The idea of heat diffusion process has triggered the emergence of diffusion geometry [60]. These features are usually using Laplaca-Beltrami operator, e.g., heat kernel signature [61], global point signature [62] and multi-solution spectral descriptor [63].

Most of the aforementioned features are designed for shape matching, retrieval and correspondence. In shape segmentation, researchers have developed other types of features. These features directly control the quality of the final segmentation results for the approaches based on k-means clustering [48], fuzzy clustering [49], Gaussian-mixture model followed by graph cut [27]. Some other recent work on segmentation which acquires better result also needs some feature definition [64, 65]. Liu et al. [1] used concavity as features. Kalogerakis et al. [45] and Huang [46] proposed the learning-based segmentation using a rich collection of the features.

Visibility-based Features. Several recent works use visibility to derive part-aware features. The intuition behind most of these visibility-based features is that two points in the a visually or functionally meaningful part (such as the leg of a table) tend to be visible from each other. For example, Shape Diameter Function (SDF) [27] captures the thickness of a shape locally among visible points. SDF is determined by sampling rays inside the cone in the anti-normal direction of a facet. The SDF value is the sum of the projected length of the rays inside the model. However, SDF may not correspond well to a visually meaningful component if the thickness is not evenly distributed. For example, as shown in Figure 2.1(c), the tabletop has quite different feature values at its center from those on the boundary. In addition, the feature values at the ends of the leg are also quite different from those closer to the tabletop.

Another visibility-based feature is Volumetric Shape Image (VSI) [66]. With the motivation of capturing the general volumetric context instead of only getting the local volumetric context of the local cone used for sampling rays, VSI tries to sample rays in more directions. The VSI feature is computed by first finding the proxy center for each vertex and then sampling ray at fixed direction. The field of VSI is achieved by comparing the difference between the sampling of a source vertex and other vertices on the mesh.

Recently, weak convex decomposition proposed by [64] also uses lines-of-sights. It first computes the pairwise visibility between all pairs of the vertices/facets on the mesh. Then

the similarity matrix is constructed with each entry's value to be 0 or 1 indicating the pairwise invisibility or visibility, followed by spectral clustering. Even though the mathematical background seems to be quite straight-forward by referring to the similar issues in machine learning problem, the decomposition sometimes is not satisfying especially when a mesh model has a large area of bended parts. van Kaick et al. [65] used the technique in [64] as pre-processing step to over-segment the mesh, but it also needs a lot of post-merging.

In all of these features [27, 64, 65, 66], traditional line-of-sight visibility is used. However, from the Figure 2.2, it could be seen that the continuous visibility 2.2 makes more sense than the general visibility in terms of charactering a potential component.

2.1.3 Concavity and Nearly Convex Decomposition

Most of the times, creating a minimum number of convex components of a concave objects is NP-hard [32, 67, 68]. More recently, several methods have been proposed to partition at salient features of a polygon. Tănase and Veltkamp [69] decompose a polygon based on the events that occur during the construction of a straight-line skeleton. Dey et al. [70] partition a polygon into collections of Delaunay triangles of points sampled from the polygon boundary. Lien and Amato [39] partition a polygon into *approximately convex* components. Their method reveals significant shape structures by recursively resolving the most concave features until the concavity of every component is below some user specified threshold. Wan [71] extends [39] to incorporate both concavity and curvatures and prevent over segmentation by avoid cuts inside pockets. Liu et al. [72] proposed the idea of α-decomposition that use persistence analysis of features obtained from the continuous convolution [73] between the polygon and a disc.

Concavity and its counterpart, convexity, have shown to be valuable for various decomposition methods. Intuitively, concavity of a polyhedron P measures how different P is from a convex object, which is typically the smallest convex object enclosing P, i.e. the convex hull $H(P)$ of P. Concavity can be measured globally from the difference between

the volumes of P and $H(P)$ or the difference of their projections [74]. Concavity can also be measured locally for every point on the surface ∂P of P. Local concavity measures how far away a point on the surface of P is from the surface of $H(P)$, thus provides richer information to guide the decomposition process.

There have been several definitions of local concavity, such as curvature [75]. A more general definition of the concavity of a point p is the length of the shortest distances from $p \in \partial P$ to $H(P)$ without intersecting P. Due to the high computational complexity of 3D shortest-path problem, approximation is required. For example, Zimimer et al. [76] proposed to compute the shortest path by tessellating the space between ∂P and $\partial H(P)$ using constrained Delaunay triangulation (CDT). However, this solution required ∂P to be closed. Many other approaches resort to the Euclidean distance between the vertex and the convex hull in the outward normal direction of the vertex. It is clear that the association between ∂P and $\partial H(P)$ is usually complex and cannot be captured by the surface normals of P. This can result in inaccurate measurement of the concavity and lead to incorrect decomposition. Lien and Amato [39] proposed to determine the association by projecting the edges and then facets of $\partial H(P)$ to ∂P. Unfortunately, such an association between ∂P and $\partial H(P)$ is not always well defined as it usually depends on how P and $\partial H(P)$ are tessellated.

Mamou and Ghorbel [77] proposed a method called Hierarchical Approximate Convex Decomposition (HACD) for 3D meshes. HACD iteratively clusters mesh facets by successively applying half-edge collapse decimation operations (see more detailed discussion in Section 5.5). Attene et al. [78] proposed to convert a model into tetrahedral mesh and then merge tetrahedra to form near convex components. However, this method requires human interaction, thus not suitable for segmenting a large number of models. The method proposed in [79] is based on a region growing approach controlled by convexity, compactness and part cost. Top-down approaches are rare. For example, Ghosh et al. [80] proposed a notion named *relative concavity* to model the concavity measure before and after every mesh cut. Nearly convex components are obtained by finding the cuts that have largest relative

concavities via dynamic programming. In many of these methods, several parameters are needed to be set to balance various features.

It should be noted that there are also methods that use convexity and concavity but do not produce segmentation with bounded convexity or concavity. For example, Au et al. [75] used concavity-aware field to form potential cuts and the final cuts are achieved by maximizing the cut score. Even though concavity is used, it doesn't have direct control of concavity for final components and several parameters and thresholds need to be set. Asafi et al. [64] defined the convexity using the line-of-sight and applied spectral clustering on the visibility matrix to achieve segmentation. van Kaick et al [65] applied merging on the initial result of [64] based on the Shape Diameter Function. However, since the pairwise visibility needs to be computed, the computation is time-consuming. Moreover, even though these two methods use convexity as clue for segmentation, they didn't directly control the convex/concave geometric property for the final components.

An important requirement in shape decomposition is its robustness to boundary noise. Several of these methods require pre-processing (e.g., model simplification [40, 69]) or post-processing (e.g., merging over-partitioned components [41, 70]) due to boundary noise. Other methods [36, 38, 39] are designed to tolerate these noise. However, few existing approaches focused on handling topological noise that appear quite commonly in polygons generated from images with significant noise and overlapping objects. On the other hand, as what will be discussed in this chapter, Continuous Visibility Feature and DuDEare robust to noises to some extent.

2.2 Continuous Visibility Feature for Shape Analysis

2.2.1 Introduction

Feature extraction often serves as a fundamental engineering task for higher level applications. For example, almost all of the recognition tasks start with defining or learning

features. This is particularly true for two-dimensional image that has a simple and uniform grid structure and pixels can be indexed by a 2D vector in a continuous coordinate system. Each pixel has fixed number (usually 4 or 8) of neighbors. Thus most of the image features are defined by the convolution operations of the local areas. Consequently, there are some default features like SIFT [81] and HOG [82] used widely in computer vision research.

Unlike images, 3D models are usually modeled in the continuous domain which makes defining 3D features challenging. Consequently, many of the features defined for 3D models usually are designed for specific types of shapes.

For example, visibility among points inside a given shape has been used directly or indirectly as shape features, in particular for the task of semantic shape segmentation. The intuition behind most of these visibility-based features is from the observation that two points sampled from the same semantic part tend to be visible from each other. For example, shape diameter function (SDF) [27] is a descriptor to describe the thickness of a mesh defined via local visibility. The idea is initially proposed for shape segmentation based on the assumption that a visually meaningful component should have similar thickness everywhere. However, this assumption does not hold in many cases. For example, for a long component, such as a leg, the thickness at one end of the component may have different thickness from the other end. Another example is a flat component or a component whose size grows gradually in a certain direction.

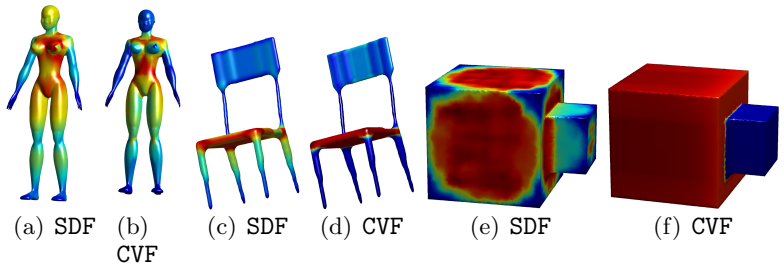


Figure 2.1: Examples of SDF and CVF on the same models. Red and blue indicate the largest and smallest SDF and CVF values, respectively. From these examples, It can be seen that CVF provides a better metric to distinguish visually different parts of a model. More extensive comparison can be found in Section 2.2.4.

I believe that the traditional line-of-sight visibility used in all existing visibility-based shape features is insufficient. In this section, I will describe a new feature named *continuous visibility feature*. The feature provides stronger visibility measure by considering the continuously visible region for a vertex or facet. Thus this feature is defined in a per-vertex manner. A point q is continuously visible by a point p if there exists a geodesic path connecting p and q that is entirely visible by p. CVF of p is defined as the area of a set of continuously visible points by p. A more precise definition of CVF can be found in Sections 2.2.2. CVF better encodes the surface and part information of mesh than SDF does. Figure 2.1(a) illustrates some example models color mapped by CVF and SDF values. It will also be shown that existing segmentation algorithms can generate better segmentation results using CVF than using other shape descriptors, such as shape diameter function [27]. Segmentation results using the Princeton Segmentation Benchmark will be demonstrated and applications of CVF and its variants (namely CVF_{avg}, strong CVF and weak CVF) beyond segmentation will also be discussed in Section 2.2.4.

A major technical challenge of computing CVF is the computational efficiency because it is known the determining the visibility of two points is expensive and determining their continuous visibility will require many more pairwise visibility checks. In Section 2.2.3, two ways to compute CVF efficiently will be presented. The first approach will use the continuous property of CVF to construct the continuously visible region in a Breadth-First-Search manner. The second approach further improves the efficiency by first collecting the potential continuous visible region and then using visibility test to search the true boundaries of the continuous visible region. Comparisons show that the second approach provides significant speedup over the first approach.

2.2.2 Definitions and Properties of CVF

Visibility among the points inside a given shape has been used directly or indirectly as shape features in the past. The intuition behind most of these visibility-based features is from the observation that two points sampled from the same (visual or functional) part tend to be

visible from each other. It usually remains true if the property is described the other way around: two points that are visible from each other are likely to be from the same part. However, there are many exceptions. For example, in a human model in a standing pose, a point from the head may see a point in the heel and in most cases head and heel will be distinguished as different parts of the model.

To overcome the drawbacks of the line-of-sight based visibility approaches and to better capture the component information of a mesh, a new feature named **continuous visibility feature** (CVF) is defined. A point q on the mesh is continuously visible by p if there exists a geodesic path connecting p and q that is entirely visible by p. Note that, this path may not be the shortest geodesic path between p and q. More specifically, the function $\text{CV}(p,q)$ is defined to return **TRUE** if and only if

$$\exists \pi \text{ such that } \forall r \in \pi, \overline{pr} \cap M = \emptyset \ ,$$

where π is a geodesic path connecting p to q on mesh M. An illustration of the continuous visibility is shown in Figure 2.2 (a).

Intuitively, by applying this stronger version of visibility measure, two points from the same part not only should see each other but the entire path connecting them should be visible. Under these definitions, a continuously visible region (CVR) of a point p is a collection of all points that are continuously visible by p. That is, CVR of p can be expressed as the set $\{q \in M \mid \text{CV}(p,q) = \textbf{TRUE}\}$. CVF of a given point p can be represented either in vector or in scalar form. In this section, scalar representation will be used, in which CVF is the area of p's CVR. If the input model is convex, then all vertices in this convex model have the same CVF, which is the surface area of the model.

Unlike visibility, continuous visibility is not commutative. That is, if $\text{CV}(p,q)$ is true, $\text{CV}(q,p)$ may not be true. A 2D example is illustrated in Figure 2.2 (b), in which the point v cannot continuously see the point w because the visibility continuity is interrupted by edges e_1 and e_2. Therefore, when both $\text{CV}(p,q)$ and $\text{CV}(q,p)$, p and q have *strong continuous*

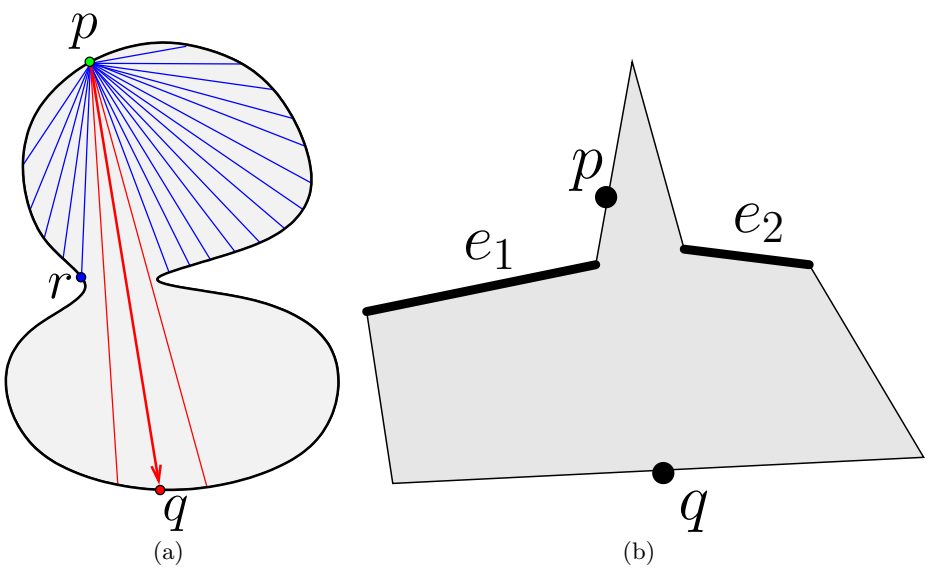


Figure 2.2: (a) Example of 2D continuous visibility. The vertex r is continuously visible from the vertex p but the vertex q is not. (b) While p is continuously visible from q, q is not continuously visible from p.

visibility.

2.2.3 Computing CVF

In this section (2.2.3), I will discuss how continuous visibility can be determined for all vertices of a given mesh. Without loss of generality, I will focus the computation of the continuous visibility of a given vertex v. In Section 2.2.3.1, a flooding approach using Breadth-First Search (BFS) rooted at v will first be introduced. However, BFS-based approach requires many visibility check between v and all vertices visited by the BFS tree. The second approach will be proposed to address this efficiency issue in Section 2.2.3.2.

2.2.3.1 BFS-based Approach

This first approach uses the property that the continuous visible region of the vertex v must be continuous because, by definition, every vertex in the continuous visible region is

connected to v via a path that is entirely visible by v. In this Breadth First Search (BFS) based method, the iterative search is expanded only at the vertices that are visible to v and stop at the vertices that are invisible to v. The correctness of this approach can be proved by saying that, it is impossible to find a vertex u that is continuously visible by v but is not discovered by the BFS approach.

Although the BFS-based method is simple to implement and provably correct, it may require many visibility checks. In the next section, an alternative approach will be proposed to speed up the computation.

2.2.3.2 Filtering

I further propose a more efficient way to compute the feature values. The main idea is to filter out the faces that guaranteed to be outside the continuously visible region of a given vertex v. If a triangle f and v form a tetrahedron of positive volume, then, then f is a positive triangle. Then a *continuously positive region* of v is a set T_v of positive triangles continuously connected to v. It is clear that v's continuously visible region must be a subset of T_v, and T_v can be constructed using the same BFS-based approach discussed in the previous section, except that it starts from the incident facets of v.

The next step involves finding the boundaries between the continuously positive region CVR_v and the continuously visible region T_v. This is done iteratively by connecting v to a vertex u that is adjacent to T_v but is not in T_v. This guarantees that $\textsc{cv}(v, u)$ must be **FALSE**. A geodesic path π connecting v and u now must cross at least one boundary between CVR_v and T_v. Let w be the vertex closest v on π that is invisible to v and w' be the last visible vertex on π before arriving w. By definition, w' is continuously visible and edge $e = \{w', w\}$ connecting w' and w must be a boundary edge. To determine the entire boundary including e, the algorithm first identifies the visibility of the third vertex of the triangle t incident to e and then move on to the edge $e' \neq e$ of t whose end points also form a pair of visible and invisible vertices. This process repeats until a closed loop is discovered. Using this loop removes vertices that not continuously visible by v. The algorithm sketching

this idea can be found in the Algorithm 1.

Algorithm 1 Filter-based CVR computation

Input: a vertex v and its positive continuous region T_v
Output: the continuous visible region of v

while $\exists f \in \partial T_v$ whose visibility to v is undetermined **do**
  Let $u \notin T_v$ be a vertex incident to f
  Find a path π connecting v and u
  Starting from v, let $e = \{w', w\}$ be the first edge such that w' is invisible and w is visible
  Let $L = \{e\}$
  while L is not a closed loop **do**
    Let f be a face incident to e that is not visited
    Let w'' be the vertex of f that is not in e
    if w'' is invisible from v **then** $e = \{w', w''\}$
    else $e = \{w'', w'\}$
    end if
    $L = L \cup e$
  end while
  Mark all vertices connected to u without crossing L be continuously invisible from v
end while

2.2.3.3 Running Time

Experiments are done by using the models from Princeton Segmentation Benchmark to obtain the running time reposted below. Some of the benchmark models are shown in Figure 2.4. The results are obtained from the C++ implementation on a machine with Intel Xeon 2.30GHz CPU and 32 GB memory, with 24 models running parallely. Using the BFS-based approach, for models with about $1.5K$ vertices, the number of intersection test is around 0.9 million and the running time is 10 seconds on average. A model with about $5K$ vertices and $10K$ facets, BFS-based approach requires about 8 million intersection tests and the total running time is around 100 seconds. For a model with $10K$ vertices, the total intersection number is around 70 million and the running time is about 900 seconds.

On the other hand, if the filter-based approach is used on the same set of models, the intersection tests for models with less $5K$ vertices reduce by 72% and the running time

reduces by 30%. For models with $5K \sim 10K$ vertices, the intersection tests and running time reduce by 82% and 37% respectively. For models with more than $10K$ vertices, the intersection tests and running time reduce by 89% and 49%.

2.2.4 Comparison and Applications

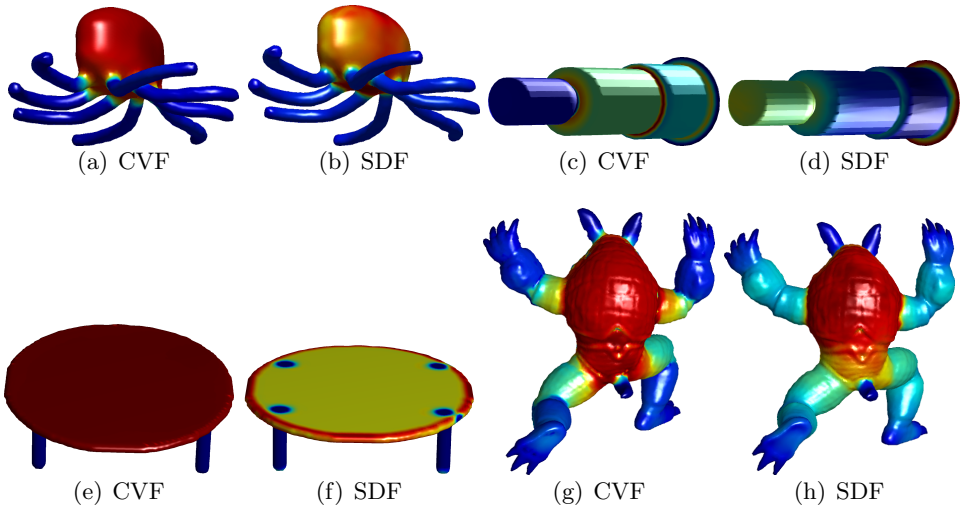
In this section, comparisons between continuous visibility feature (CVF) and shape diameter function (SDF) will be shown. SDF has been widely used in part-based shape analysis. For example, MeshLab has adopted it as a filter and CGAL has re-implemented SDF for extracting features from shape and then used it for shape segmentation.

2.2.4.1 Visual comparisons

Figure 2.3 shows a visual comparison between CVF and SDF. From the pictures, it can be seen that semantic parts in these examples have more constant CVF in each part and higher variance when SDF is used. For example, the head of the octopus model has more consistent CVF values than SDF. For the tool model, the differences between parts are more distinct than those in SDF. Furthermore, SDF color-map shows four spots on the tabletop because the rays sent from these places would reach the leg of the table. Finally, in the legs, hands and torso of the Armadillo model, CVF provides better distinction between the neighboring parts and, again, more consistency within the parts.

2.2.4.2 Shape Segmentation

Segmenting a mesh into meaningful parts is a standard step toward part-based shape analysis. In order to have a fair comparison with SDF, CGAL's SDF-based segmentation is adopted and SDF feature is replaced with CVF. The shape segmentation implemented in CGAL first infers a Gaussian Mixture Model (GMM) on the distribution of the feature values (SDF or CVF), and then each facet is assigned a soft label based on the inferred



(a) CVF (b) SDF (c) CVF (d) SDF

(e) CVF (f) SDF (g) CVF (h) SDF

Figure 2.3: Side-by-side comparison between CVF and SDF values. In the ideal situation, each semantic part should have a constant feature value. In these pictures, the CVF values have lower variance in each semantic part than SDF values do.

Gaussian Mixture Model. The final segmentation is achieved by applying the k-way graph cut. The data term of the graph cut is encoded by the soft labels, and the smoothness term is encoded by dihedral angle and edge length. Figure 2.4 shows segmentations using CVF with CGAL implementation. Figure 2.5 shows the segmentation evaluation results with CVF, SDF and other segmentation algorithms from the Princeton Segmentation Benchmark [28]. The evaluation is measured by Rand Index (RI) scores. Lower RI scores indicates higher similarity to human generated segmentations.

Originally, RI evaluation of two segmentations (of the same polygon) is measured by the likelihood that each pair of cells is in the same component of two segmentations [83] and, thus, higher score means higher similarity with two identical segmentations having score to be 1. However, in the Princeton Segmentation Benchmark[28], the RI score is defined as $1 - original\_RI$, which measures the discrepancy between two segmentations. More specifically, let S_1 and S_2 be two segmentations, and s_i^1 and s_i^2 be the indices of cell

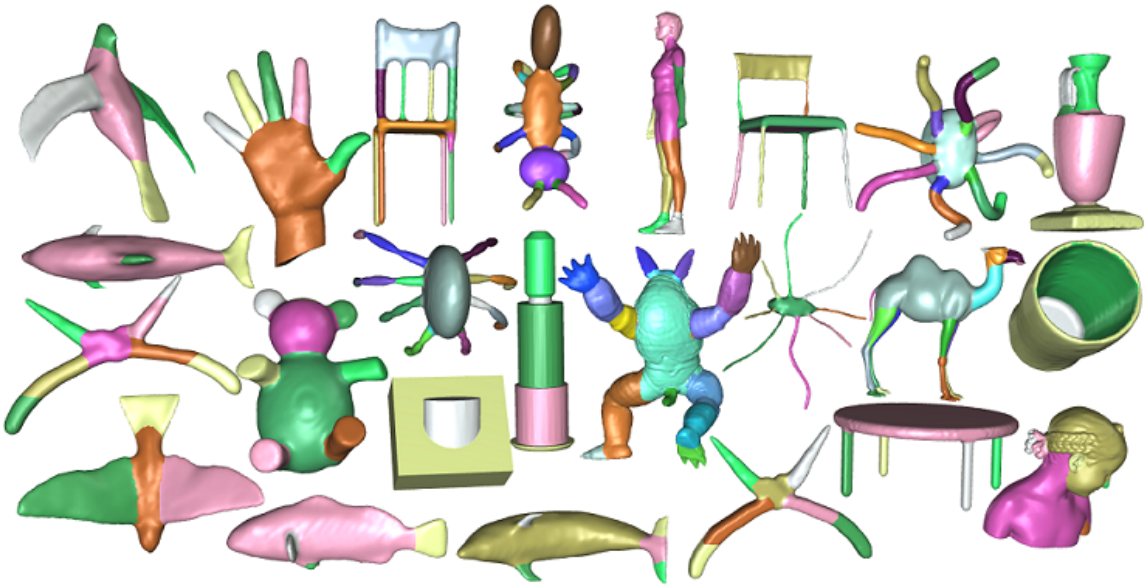


Figure 2.4: Segmentation created using CVF.

i in S_1 and S_2, respectively. Then their RI is defined as:

$$\mathrm{RI}(S_1, S_2) = 1 - \binom{2}{N}^{-1} \sum_{i,j,i<j} \left[C_{ij}P_{ij} + (1 - C_{ij})(1 - P_{ij}) \right] ,$$

where N is the number of cells inside the original polygon, $C_{ij} = 1$ if $s_i^1 = s_j^1$, $P_{ij} = 1$ if $s_i^2 = s_j^2$, $C_{ij}P_{ij} = 1$ if cell i and cell j have the same index in both S_1 and S_2, and, finally, $(1 - C_{ij})(1 - P_{ij}) = 1$ indicates the cell i and j have different indices.

Additional comparisons evaluated using other metrics such as consistency error, cut discrepancy, Hamming distance all show similar trend as RI.

The bar labeled as "CVF" shows the evaluation results using the original CVF (as oppose to CVF_{avg} that will be discussed later). It can be seen that the RI score of CVF (0.17) is slightly better than the RI score of SDF (0.18). That fact that CVF does not provide more significant difference leads us to investigate deeper into the segmentation results. Table 2.1

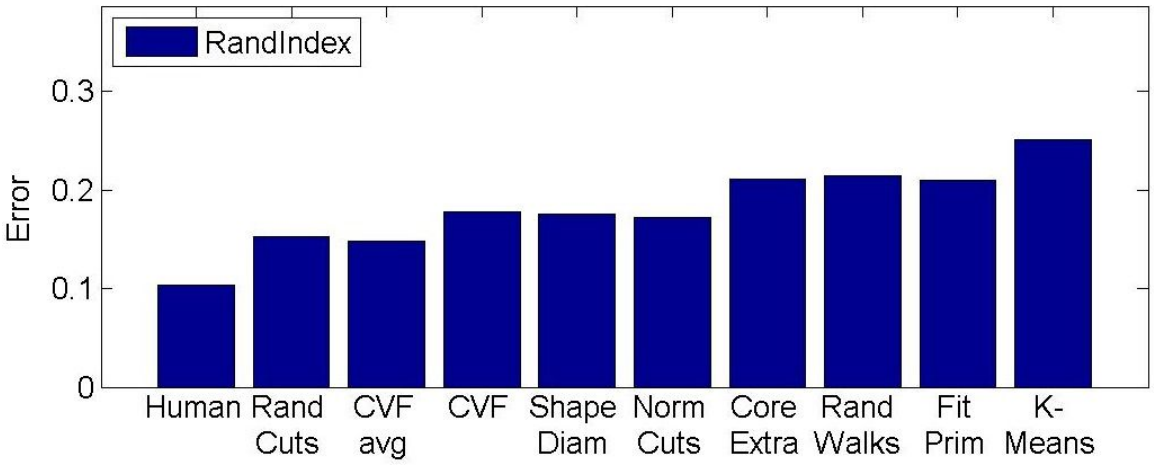
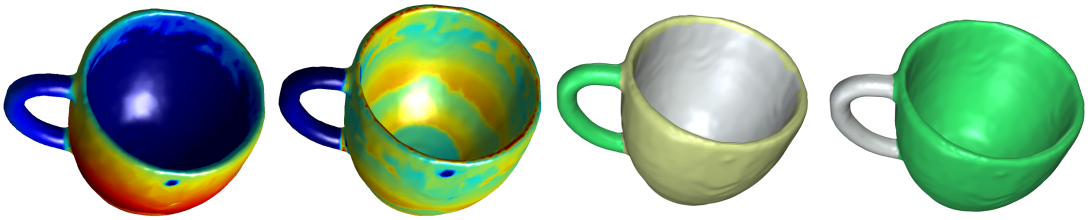


Figure 2.5: The comparison of segmentations using Princeton Segmentation Benchmark. The y-axis indicates errors measured in Rand Index values.

shows the RI scores for all categories in the benchmark.

Note that the methods compared in the Benchmark used the features such as geodesic distance, cut perimeter, component area, etc. Some recent approaches do not introduce new features, but instead combing existing features via heuristics or learning techniques. For example, Kalogerakis et al. [45] use a set of descriptors to learn the segmentation, including curvature, PCA, shape contexts, geodesic distance, spin image etc. van Kaick et al. [65] employ a multi-step process: 1. Generate over-segments; 2. Merge segments with SDF dissimilarity lower than a threshold; 3. Cut boundary refinement. Thus it is also worthwhile to try to incorporate CVF with the feature-rich approaches to understand how CVF can be combined with other features and how it performs in the combination.

From Table 2.1, segmentation using CVF in the cup, airplane, and bird categories performed poorly with respect to the human data collected in the benchmark. Figure 2.6 shows the distinct CVF values from the inside (concave part) and the outside (convex part) of the cup. The reason that CVF would be different for inside and outside parts is that the inside part is concave everywhere. The vertices in the inside parts can hardly continuously see any facets except the ones adjacent to it, while the outside part is convex, the vertices



(a) CVF (b) CVF_{avg} (c) CVF (d) CVF_{avg}

Figure 2.6: Segmentations created using CVF and CVF_{avg}.

there can continuously see a lot of facets. The top left picture in Figure 2.6 shows the visualization of the cup. The bottom left shows the segmentation result using the CVF. The GMM-based shape segmentation separates the inside from the outside, which in some sense is a reasonable segmentation but makes the evaluation score rather low because most results from the benchmark segment the cup handle from the cup body. In the next section, I will discuss three variants of CVF that will address these issues.

2.2.4.3 CVF variants

CVF is versatile and can be defined in various ways to address several issues identified in plain CVF. In this section, I will describe three variants of CVF that provide better results than CVF. Due to space limitation, detailed comparisons between these variants can be found in the supplementary materials.

In the first CVF variant, I propose to average the CVF values of a facet (or vertex) p. That is, a ray will be sent in its counter-normal direction and identifies the nearest facet q intersected by the ray. Then the CVF_{avg} of p is the average between original CVF_p and CVF_q. The upper right picture in Figure 2.6 shows the visualization of the CVF_{avg}, and the lower right picture in Figure 2.6 shows the segmentation using CVF_{avg}. It can be seen that the results are more consistent to human segmentation.

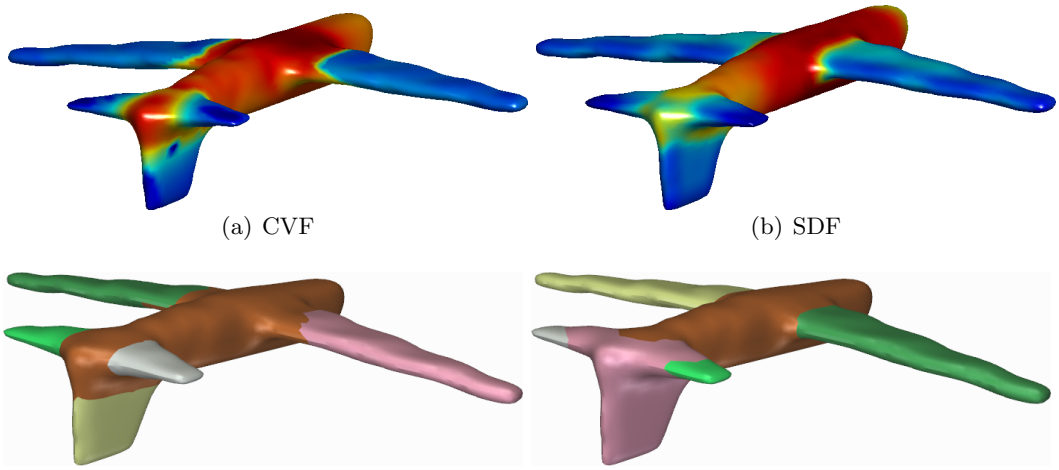
In Table 2.1, the RI scores of CVF_{avg} are shown. Note that CVF_{avg} improves CVF significantly in the cup category (from 0.45 to 0.23). Moreover, surprisingly, CVF_{avg} also

improved most of the RI scores from other categories (with the exceptions in airplane, plier, and fourleg). Consequently, it can be seen that CVF_{avg} outperforms SDF in 11 categories (out of 19). Even when SDF has lower RI scores, the differences between CVF_{avg} and SDF are usually small, except in the Airplane category. Figure 2.7 shows an example in this category.

Table 2.1: Compare CVF and CVF_{avg} with SDF . The CVF_{avg} means the the feature values for a vertex are are achieved by averaging the CVF values between original CVF values and the the facet that is hit by the ray sent out from that vertex in its anti-normal direction (counter-normal direction.)

	RI			
	SDF	CVF	CVF_{avg}	Diff
Human	0.18	0.16	**0.14**	0.02
Cup	0.36	0.45	**0.23**	0.22
Glasses	0.2	0.21	**0.19**	0.02
Airplane	**0.09**	0.17	0.2	-0.03
Ant	**0.02**	0.04	0.04	0
Chair	0.11	0.1	**0.07**	0.03
Octopus	0.05	0.04	**0.03**	0.01
Table	0.18	0.12	**0.09**	0.03
Teddy	**0.06**	0.08	0.07	0.01
Hand	0.2	0.17	**0.13**	0.04
Plier	0.38	**0.21**	0.22	-0.01
Fish	0.25	**0.18**	**0.18**	0
Bird	**0.12**	0.25	0.18	0.07
Armadillo	**0.09**	0.11	0.1	0.01
Bust	**0.30**	0.31	**0.30**	0.01
Mech	0.24	0.21	**0.14**	0.07
Bearing	**0.12**	0.14	0.13	0.01
Vase	0.24	**0.18**	**0.18**	0
Fourleg	**0.16**	**0.16**	0.17	-0.01
Average	0.18	0.17	0.15	0.03

The second CVF variant is called *strong CVF* that requires mutual continuous visibility. two points p and q have *strong continuous visibility* if both $\text{CV}(p,q)$ and $\text{CV}(q,p)$. The first two columns in Fig. 2.9 show the color-map of CVF and strong CVF. In most examples, CVF and



(a) CVF (b) SDF

(c) CVF (d) SDF

Figure 2.7: Limitation of CVF. In the left figures, the vertices (colored in yellow) between the airplane body and wings can continuously see both the wing and body. This results in large CVF values can causes the segmentation cuts to be at the middle of the wings.

strong CVF are almost identical, except in the last example (bottom of Fig. 2.9). Notice the checker board-like pattern in CVF. The strong CVF provides much more consistent feature values than CVF in this example.

The third CVF variant is designed to address the issues found in the airplane or bird-like models, in which the fuzzy regions between the wings and the body cause poor segmentation as shown in Fig. 2.7. Now, let us go back to the definition of continuous visibility. Even if both points p and q are continuously visible to each other, the geodesic paths that prove their mutual continuous visibility may be different. A such example illustrated in Fig. 2.8 shows that two different geodesic paths are taken by points p and q. One can see that path π_p that witnesses $\text{CV}(p,q)$ is much longer than the path π_q that witnesses $\text{CV}(q,p)$. In fact, it is possible to make the path π_v arbitrary long by moving the right most (shaded) face in Fig. 2.8 to its right. Intuitively, there is a negative correlation between the likelihood of v and w belong to the same part and the ratio between the continuous visibility path length and the length of the shortest geodesic length. Similarly, the edge length of e in Fig. 2.8 also has negative effect on the relationship between CVF and the likelihood of v and w

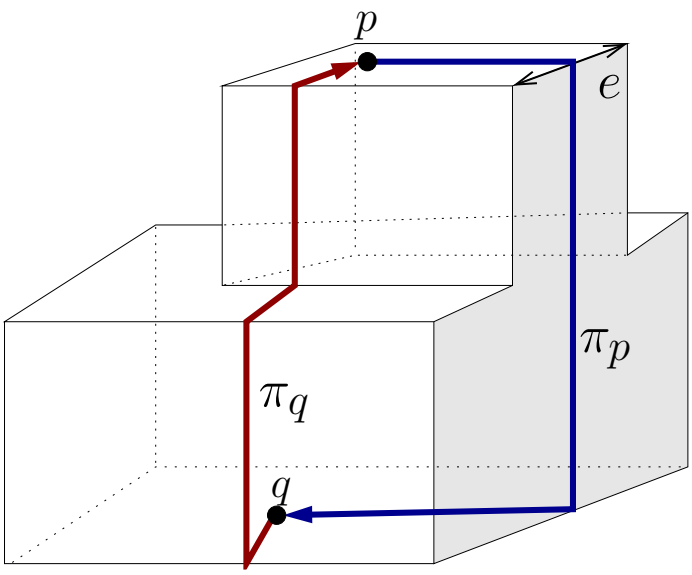


Figure 2.8: An example of the continuous visibility between two points v and w through different paths. The point v is continuously visible by the point w via path π_w and w is continuously visible by v via a much longer path π_v.

belonging to the same part. In all cases, it will be desirable to take the visibility along the shortest geodesic path into consideration. Consequently, I propose that a measure of *weak continuous visibility* WCV. Given two points p and q, $\{\text{WCV}(p,q) = \textbf{TRUE}\}$ if p and q are visible to each other but the length of the invisible part of the shortest geodesic path connecting p and q is smaller than a user defined value τ. Let π be the the shortest geodesic path connecting p and q and let $\bar{\pi}_w$ and $\bar{\pi}_v$ be subset of π invisible to v and w, respectively. The function $\text{WCV}(p,q)$ returns **TRUE** if $\max(||\bar{\pi}_w||, ||\bar{\pi}_v||) \leq \tau$.

The last column in Fig. 2.9 shows the results of weak CVF. It is clear that weak CVF provides significantly sharper boundaries that can lead to better segmentation. See segmentation results using weak CVF with varying values of *tau* in the supplementary materials.

Since the CVF and its variants are based on visibility checking, one question may be raised by people is whether CVF and its variants are robust to noises. CVF is insensitive to bumps and noise. With the original or strong CVF in 3D, it is unlikely that noise or random bumps block the visibility of all paths between two points on the surface. In addition, the weak CVF can tolerate a certain amount of invisibility since it allows part of the shortest

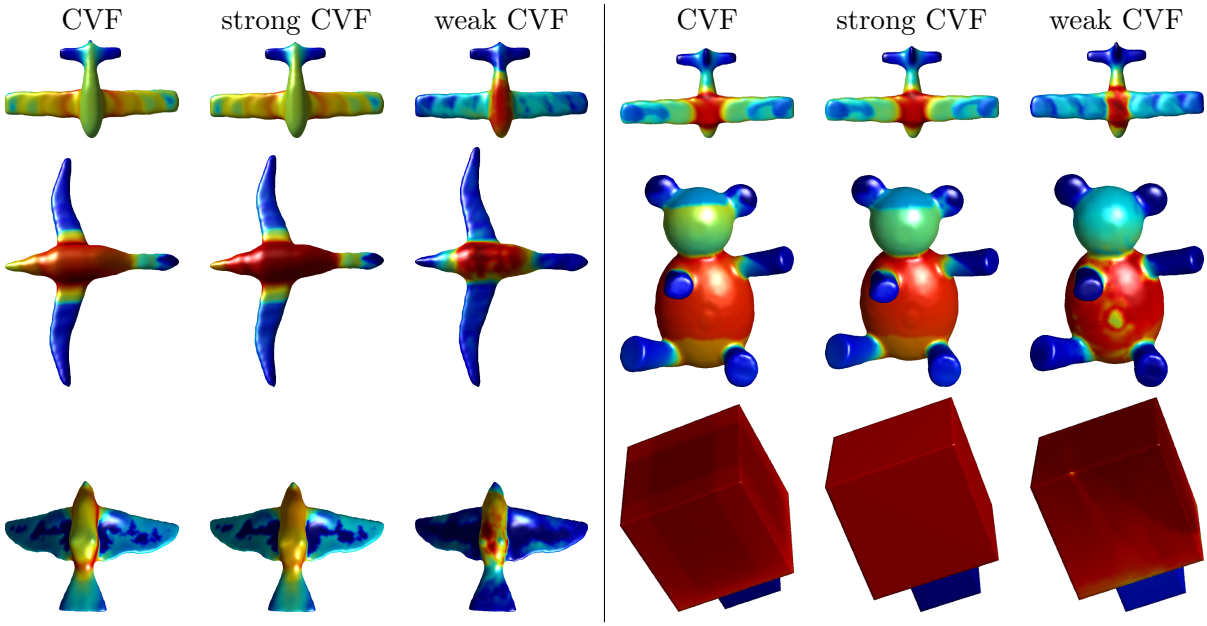


Figure 2.9: Results obtained from CVF (left) vs. strong CVF (middle) vs. weak CVF (right).

geodesic path to be invisible.

2.2.4.4 Application: Part-based Retrieval

In most shape retrieval problems, the objective is to retrieve a shape that is globally similar to a query shape. Since CVF can be used to generate segmentation and encode shape signature of each component, CVF can be used for context-aware part retrieval following the strategy described in [27]. Specifically, the segmentation using CVF will first be generated as described in the previous sections and then will be used construct a hierarchical part structure of the given shape from the segmentation. In this hierarchical part structure, the whole model is the root of the tree and the leaves are the components in the segmentation. Then each node in the tree is encoded with the CVF histogram as its signature. During the retrieval stage, the retrieval is reduced as a bipartite matching problem. Given a query component, it is compared with all the components in the dataset. To match two nodes

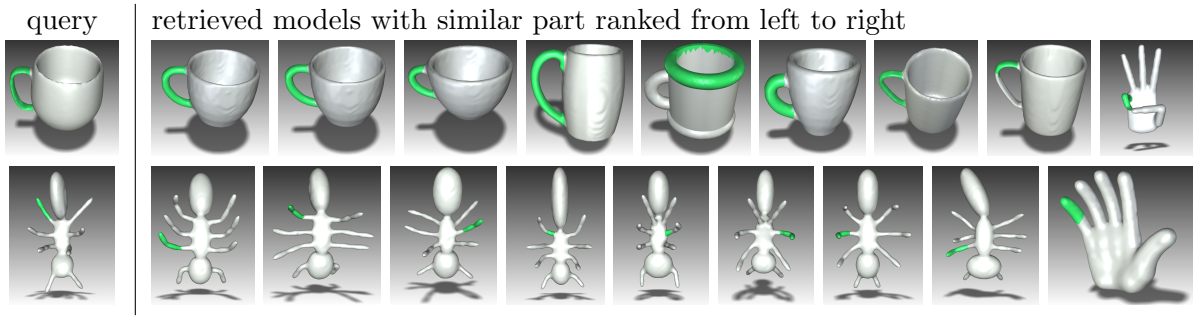


Figure 2.10: Part-based retrieval using CVF.

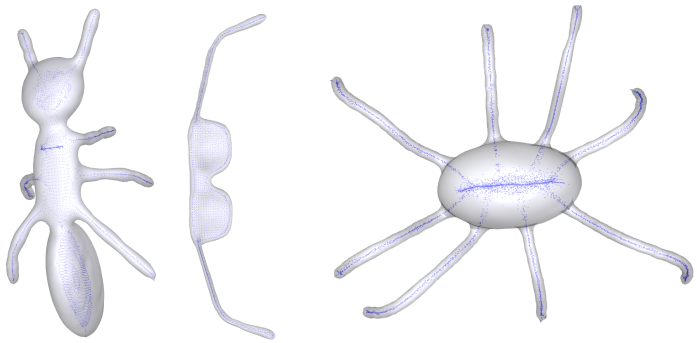


Figure 2.11: Medial-axis samples created using CVF.

(components) p and q, a bipartite graph will be built between nodes of the paths leading the root to p and q in the hierarchy.

The total matching score is achieved by solving this maximum weight bipartite matching problem. Fig. 2.10 shows two examples and top 9 matching parts in the database.

2.2.4.5 Application: Medial Axis Extraction

Figure 2.11 shows the medial axis sample points generated by CVF. Skeleton points are constructed by using the idea of reference points in [66]. Basically, for a vertex p, a ray will be sent to the continuously visible region and each ray gives a reference point. It can be seen that the skeleton points provide a good approximation of the medial axis.

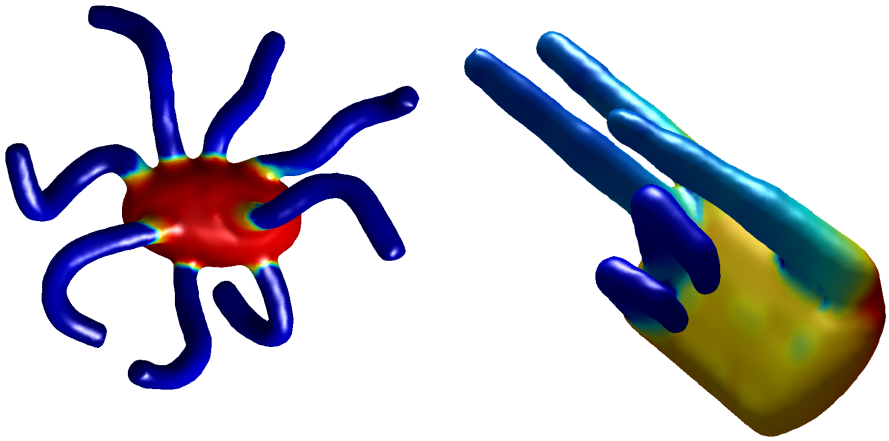


Figure 2.12: Invariance to pose change and deformation.

2.2.5 Discussion & Future Work

In this section, I showed a new shape feature called continuous visibility feature (CVF). I proposed two algorithms to compute CVF for each vertex of a given mesh: one is based on BFS and the other one is based on identifying potential continuous visibility regions. I showed that the second approach is significantly faster than the BFS approach. In the experimental results, I show that CVF provides good results comparing to the methods in Princeton segmentation benchmark. If the CVF values are averaged between vertices on the opposite sides of their anti-normal directions, called CVF_{avg} then I see significant improvement over CVF and SDF. I demonstrated a couple of applications beyond shape segmentation, like shape retrieval and medial axis sampling.

My experiments show that CVF is insensitive to pose change and deformation. It is true that the CVF value is not isometry invariant; however, it is the distribution of CVF that matters. Pose change or deformation does not change the CVF of one vertex only. Rather, many vertices' CVF change. If I consider the overall distribution of all vertices' CVF, for most poses, the overall distribution still preserves part-awareness. For example, in Fig 2.12, even though the legs of octopus (fingers of hand) have different poses or deformation, the distribution of CVF can still capture the part information.

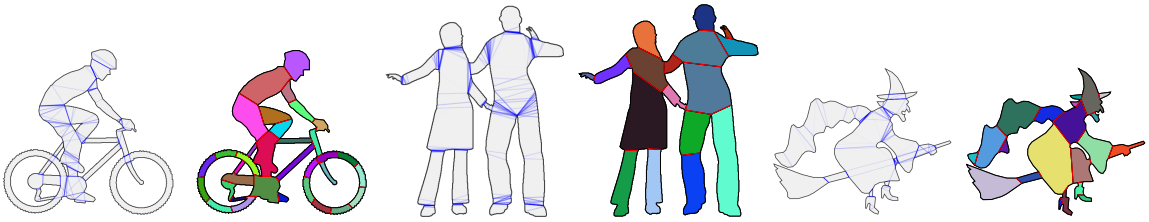
I believe the continuous visibility feature would have more applications. Currently I only focus on the scalar representation, more applications of CVF can be explored with vector representation, a list of Booleans representing the continuous visibility of all vertex pairs. For example, modifying the existing features with the constraints of CVF's vector representation. This work was also published in the paper [2].

2.3 Dual-Space Nearly Convex Decomposition for 2D Shapes

2.3.1 Introduction

Decomposing two-dimensional (2D) shapes into functional and visually meaningful parts is a fundamental process in human vision [35, 84, 85, 86, 87, 88]. Many segmentation and decomposition techniques have been proposed to mimic this process computationally [89, 90, 91, 92]. While these techniques (usually known as *part segmentation*) generate impressive results, they also have only focused on relatively simple shapes, such as those composed of a single object either without holes [35] or with few simple holes [36, 38, 39]. Even when holes are explicitly considered, existing techniques usually decompose the shape so that no holes are left [36, 38, 39]. In many situations, shapes are created from images (e.g., silhouettes) of overlapping objects and can also contain many undesired holes due to sensor noise. In other cases, these holes can be important part of the shape and arbitrary complex. These complex shapes, such as those shown in Figures 2.13, 2.18, and 2.19 pose grand challenges to the existing part segmentation methods.

Another major limitation in the existing part segmentation methods is the lack of well-defined criteria and benchmark for quality comparisons. Unlike image segmentation, comparison between part segmentation methods is usually done visually and evaluated based on the size of the final decompositions, i.e. smaller decomposition is better. The most widely used MPEG 7 image set [93] consists of only shapes with single boundary. This limitation is further hindered by the lack of public domain implementations of many existing methods.



(a) Human (b) DuDe (c) Human (d) DuDe (e) Human (f) DuDe

Figure 2.13: Examples of decompositions from human and dual-space decomposition (DuDe).

To address the aforementioned issues, in this section, a new decomposition method, called *Dual-space Decomposition* (or simply DuDe), is proposed. DuDe is designed to handle complex 2D shapes that may be composed of overlapping objects with significant number of holes. DuDe accomplishes this by recognizing the importance of these holes by classifying holes as either topological noise and structurally important features. DuDe is developed based on the idea of nearly convex parts. Strategies that decompose shapes into nearly convex components have been proposed recently in the communities such as computer vision, pattern recognition and graphics [36, 38, 39, 71]. In order to recognize the importance features from the external and hole boundaries, DuDe creates the nearly convex decomposition of the shapes by segmenting both positive and negative regions of the shape (the polygon itself and its complementary).

In this section, an efficient dual-space method is proposed to decompose a non-convex polygon into several approximate convex parts, similar to human decomposition results which will be explained in following sections. This method is designed to handle shapes with complex holes. To quantitatively evaluate DuDe, I take an initial step to collected more than 3800 segmentation from 142 non-expert using 72 polygons (shown in Figure 2.18). Using two statistics-based evaluation methods described in Section 2.3.4, it will be shown that DuDe generates segmentation closer to man-made segmentation than existing methods.

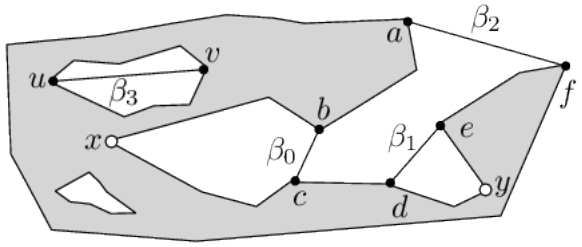


Figure 2.14: Bridges β_0, β_1, β_2, and β_3 and a concave feature x. The end points of β_3, u and v, are the antipodal vertices of the hole. Bridges β_0 and β_1 are the children of β_2.

2.3.2 Concavity

Concavity and the process of measuring it play the key role in the method (DuDe). To ease the concavity measuring process, 2D shapes are represented using polygons. Let us first define some notations used throughout the section. A polygon P is represented by a set of n boundaries $\{P_0, P_1, \ldots, P_{n-1}\}$, where P_0 is the external boundary and $P_{k>0}$ are boundaries of *holes*. Each boundary consists of an ordered set of vertices $\{p_i\}$ which defines a set of edges. Each edge starts at vertex p_i as $e_i = \overline{p_i p_{i+1}}$ and has two associated vectors: the vector $\overrightarrow{v_i} = \overrightarrow{p_i\, p_{i+1}}$ and the outward normal $\overrightarrow{n_i}$. Because DuDe performs decomposition in both areas enclosed by P and, $\overline{P}$, the complement of P, it is worth noting that $\overline{P}$ can be obtained by reversing the ordering of $\{p_i\}$. In many cases, polygons are usually extracted from images by detecting the silhouettes, thus usually containing many geometric and topological noises.

Because DuDe relies heavily on the measure of concavity, let us now define the concept of bridge and pocket.

Definition 1. A **bridge** β of a given polygon P is a segment $\beta = \overline{vu}$ connecting two points v and u on the boundary ∂P of P from the space exterior to P. More specifically, a segment $\overline{vu}$ is a bridge of P if and only if $v, u \in \partial P$ and the open set of $\overline{vu}$ is in the complement $\overline{P}$ of P, i.e. $\overline{vu}^\circ \subset \overline{P}$.

Therefore, a bridge cannot enter P or intersect the boundary of P except at its end

points. Examples of bridge are shown in Fig. 2.14. Note that, unlike bridges defined [39], this definition of bridge can be inside the convex hull of P.

Definition 2. A **pocket** ρ of a bridge $\beta = \overline{vu}$ is a subset of the boundary ∂P connecting v and u so that the region enclosed by β and ρ is completely in $\overline{P}$.

Intuitively, when traversing the boundary of P, a bridge can be viewed as a short cut over its pocket. For example, the pocket of the bridge β_0 in Fig. 2.14 is a polyline between vertices b and c via x.

The relationship between the bridge and pocket gives us an intuitive way to define concavity. For example, in the simplest case, the concavity of vertices in the pocket is simply the straight line distances to the bridge. A complete definition of concavity will be provided in Section 2.3.3 for the cases that the pocket is embedded in other pockets.

2.3.3 Dual Space Decomposition of Polygons

In dual-space decomposition (DuDe), the input polygon P is decomposed based on the decomposition of the complement $\overline{P}$ of P (*dual-space*). Algorithm 2 outlines the idea. The algorithm starts by determining the bridges and pockets from the convex hull $CH(P)$ of P. Algorithm 2 then proceeds by decomposing the regions $\overline{P_i}$ enclosed between the bridge β_i and pocket ρ_i pair. The cuts generated by these recursively calls Dual-Decomp($\overline{P_i}$, τ) are, by definition, bridges of P. Concavity is then measured, important concave features are identified using the bridges from $CH(P)$ and the cuts for $\overline{P_i}$. Finally, P is decomposed using these concave features. In the rest of this section, each of these steps will be discussed in detail. Figure 2.15 illustrates $\overline{P}$ is decomposed to find concave features of P.

2.3.3.1 Bridge Hierarchy

The bridges β are obtained from two sources: (1) the convex hull boundary (line 3 in Algorithm 2) and (2) the decomposition of the pockets (line 5 in Algorithm 2). First, note

Algorithm 2 Dual-Space Decomposition

procedure DUAL-DECOMP(P, τ)
    Compute convex hull $CH(P)$ of P
    Determine bridges $\beta = \partial CH(P) \setminus \partial P$
    Determine pockets ρ from β
    for each $\overline{P_i}$ enclosed by $\rho_i \in \rho$ and $\beta_i \in \beta$ **do**
        $\kappa = \kappa \cup$DUAL-DECOMP($\overline{P_i}$, τ)
    end for
    Measure concavity using $\beta \cup \kappa$ ▷ see Section 2.3.3.1
    $c =$ FEATURES($\beta \cup \kappa$, τ) ▷ see Section 2.3.3.2
    return CUT(P, c) ▷ see Section 2.3.3.3
end procedure

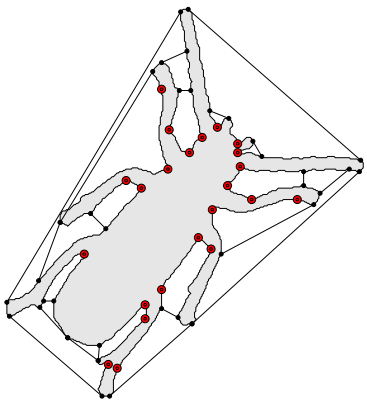


Figure 2.15: Decomposed $\overline{P}$.

that these bridges form a hierarchy. More specifically, a bridge β is the parent of β' if the pocket ρ of β is contained in the pocket ρ' of β'. For example, in Fig. 2.14, bridges β_0 and β_1 are both kids of β_2. It is not difficult to show that this relationship of all bridges can be uniquely determined. That is, when two pockets overlap, one must enclose the other one. As a consequence, the following theorem stands.

Theorem 3. The hierarchical relationship of bridges determined in Algorithm 2 must form a tree structure.

As a result, each vertex of P has a unique path to the convex hull boundary of P. The length of this path determines the concavity of the vertex.

Note that a hole boundary is a pocket associated with only bridges from the decomposition. When a hole is nearly convex, there is no bridge associated with the hole. Therefore, a pseudo bridge is created by connecting the *antipodal pair*, i.e., two farthest apart vertices, of the hole (see β_3 in Figure 2.14). When a hole is decomposed by DuDe, bridges are formed from the cuts. Given k bridges, k bridge hierarchies can be formed by using each cut as the root. The hierarchy with lowest depth is chosen to measure the concavity of vertices in the hole.

The objective in measuring the concavity is to identify pockets that have large intolerable concavity. Intuitively, the concavity of a pocket is the largest concavity of its vertices.

Definition 4. For a pocket ρ without children, the definition is:

$$\text{concavity}(\rho) = \max_{v' \in \rho} \big(\text{dist}(v', \beta)\big) \ . \tag{2.1}$$

Otherwise, concavity(ρ) is defined as:

$$\max_{\rho' \in C(\rho)} \big(\text{concavity}(\rho') + \text{dist}(\beta', \beta)\big) \ , \tag{2.2}$$

where $C(\rho)$ is the children of ρ, β' is the bridge of ρ', and dist(x, y) is the shortest Euclidean distance between objects x' and y.

ρ is an *intolerable pocket* if concavity(ρ) is greater than τ. Finally, it should be pointed out that this hierarchical definition of concavity provides better accuracy and efficiency over the traditional concavity measurement computed directly between vertices and the bridge using either the (fast but inaccurate) straight-line distance or the (accurate but slow) shortest-path distance [39].

2.3.3.2 Detect Intolerable Concave Features

Given an intolerable pocket ρ either from the external boundary or a hole boundary of P and the complement polygon $\overline{P}$ enclosed by ρ and its associated bridge, The intolerable concave features of ρ are determined by finding the smallest CH' approximation of the convex hull $CH(\overline{P})$ of $\overline{P}$ such that the distance from CH' to $CH(\overline{P})$ is smaller than τ. Using the idea similar to the variational shape approximation [94], an optimal approximation CH'_i of $CH(\overline{P})$ with i vertices in iteration i is obtained. The approximation process starts from two vertices, i.e. $i = 2$, in the approximation and iteratively add a vertex to the approximation to obtain a better approximation until the two-way Hausdorff distance between CH'_i of $CH(\overline{P})$ is less than τ. The vertices in CH'_i are the intolerable concave features. An example of the intolerable concave features is shown in Figure 2.16(a).

Because the complement polygon $\overline{P}$ of ρ must be nearly convex (otherwise $\overline{P}$ is decomposed), it is provable that all the intolerable concave vertices of ρ must be on the convex hull $CH(\overline{P})$ of $\overline{P}$ [95]. In addition, by determining the smallest approximation of $CH(\overline{P})$ with bounded Hausdorff distance, the method above provides the minimum number of intolerable concave features .

2.3.3.3 Decompose at Intolerable Concave Features

This section (2.3.3.3) describes how the intolerable concave features of a given polygon P are connected into *cuts* that segment P into nearly convex components. My method starts off by determining the *resolvers* of each concave feature. A resolver of a concave feature v is a set of diagonals of P that can locally reduce the concavity of v to τ or less. Once all resolvers from P are identified and evaluated, the method then proceeds to determine a set of resolvers that maximizes the total scores subject to the constraints that no conflicting resolvers are selected.

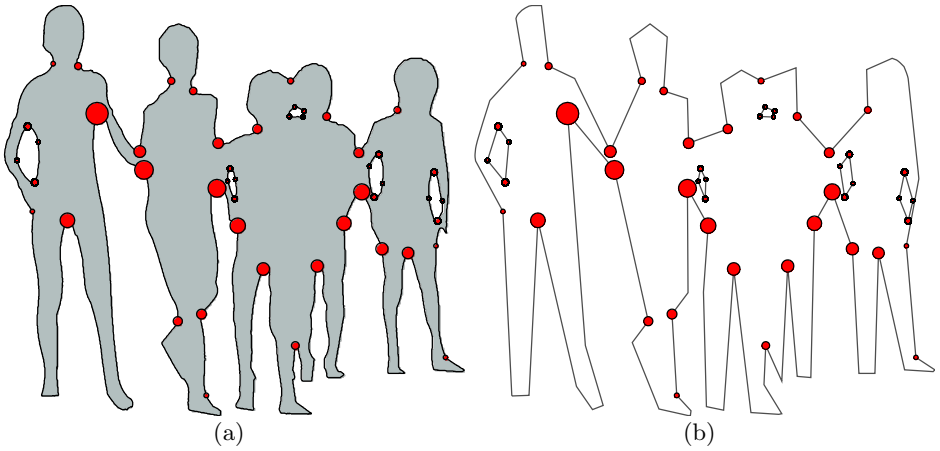


Figure 2.16: (a) Polygon with identified concave features. Size of the circle indicates the significance of the feature. (b) Simplified polygon using the concave features.

The rest of this section will discuss how a resolver is defined and identified in Section 2.3.3.3.1. The optimization problem is addressed by solving *0-1 integer linear programming* in Section 2.3.3.3.2.

2.3.3.3.1 Resolvers of Intolerable Concave Features

A resolver of an intolerable concave feature v consists of a set of diagonals incident to v. A diagonal d is valid for a given concave feature v if (1) d is in the interior of P and (2) the end points of d belong to different pockets.

Given a polygon P, valid diagonals are determined using the Constrained Delaunay Triangulation (CDT) of both P and the simplified polygon $\tilde{P}$. The simplified polygon $\tilde{P}$ is composed of intolerable concave features and the vertices between two consecutive bridges. Essentially, $\tilde{P}$ is P with all pocket vertices replaced by the intolerable concave features. An example of $\tilde{P}$ is shown in Figure 2.16(b). To find all valid diagonals, the $\mathrm{CDT}_{\tilde{P}}$ of $\tilde{P}$ is computed. Let $\mathrm{D}_{\tilde{P}} \subset \mathrm{CDT}_{\tilde{P}}$ be a set of valid diagonals incident to concave feature of P. The CDT_P is computed using the edges of P and the diagonals in $\mathrm{D}_{\tilde{P}}$ as constraints. Let $\mathrm{D}_P \subset \mathrm{CDT}_P$ be a set of valid diagonals incident to concave feature of P. The valid

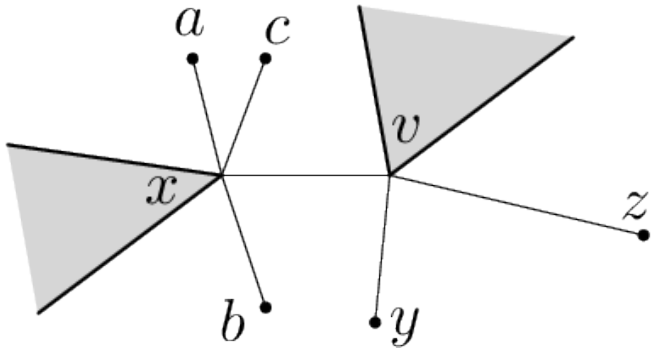


Figure 2.17: Resolvers of vertices x and v. The resolvers of vertex v include $\{\overline{vx}, \overline{vz}\}$ and $\{\overline{vy}\}$. The resolver $\{\overline{vx}, \overline{vz}\}$ conflicts with the resolver $\{\overline{vy}\}$ and all resolves of x.

diagonals $\mathrm{D}_{\tilde{P}} \cup \mathrm{D}_P$ are candidates for resolvers.

A set of valid diagonals D can locally resolve a concave feature v if the *local residual concavities* at v are all less than τ. These local residual concavities at v due to D can be computed using only D and the edges of $\tilde{P}$ incident to v. Then, a resolver $\mathcal{R}$ of v is simply a *minimum set of valid diagonals* that can resolve v. An example of resolvers for vertices v and x is shown in Fig. 2.17.

The resolvers from a vertex v are mutually exclusive because a single resolver, by definition, can resolve v. As a consequence, no two resolvers of v should be selected in the final decomposition. To ease the discussion, the complement $\overline{\mathcal{R}_i}$ of a resolver $\mathcal{R}_i$ is defined to be a set of mutually exclusive (conflicting) resolvers. For example, in Fig. 2.17, the complement of resolver $\{\overline{vy}\}$ (for vertex v) is the resolver $\{\overline{vx}, \overline{vz}\}$. See the caption of Fig. 2.17.

2.3.3.3.2 Final Cut Selection

Cut selection in dual decomposition is an optimization problem that maximizes the total score of the selected cuts subject to the constraints that no conflicting resolvers are selected and each concave feature requires a resolver. I will now show that this optimization problem can be solved using 0-1 integer linear programming.

Let $\mathcal{R}$ be a resolver and $\overline{\mathcal{R}}$ be the conflicting resolvers of $\mathcal{R}$. The score $S(\mathcal{R})$ of a resolver

$\mathcal{R}$ is a function of (1) the number of features resolved by $\mathcal{R}$, (2) the concavity of resolved features and (3) the total length of $\mathcal{R}$. My goal is to select a subset $\mathcal{K}$ of all resolvers $\mathbf{R}$ so that $\sum_{\mathcal{R}\in\mathcal{K}} S(\mathcal{R})$ is maximized subject to the following constraints (1) $||\mathcal{K}|| \leq n$ and (2) $\mathcal{K} \cap \overline{\mathcal{R}_i}$ is empty for all $\mathcal{R}_i \in \mathcal{K}$. To formulate this problem as a 0-1 integer linear programming, let $\mathbf{R}$ be a vector of all resolvers and $\mathbf{x}$ be a 0-1 vector of size $||\mathbf{R}||$. Then the final cuts are obtained by solving the following linear programming problem:

$$\text{maximize} \qquad S(\mathbf{R}^{\mathrm{T}}\mathbf{x}) \tag{2.3}$$

$$\text{subject to} \qquad \sum_{x_i\in\mathbf{x}} x_i \leq n \ , \tag{2.4}$$

$$\text{and} \quad x_i + x_j \in \{0,1\},\ \forall(\mathcal{R}_j \in \overline{\mathcal{R}_i}) \ , \tag{2.5}$$

where n is the number of intolerable concave features.

2.3.4 Experimental Results and Evaluation

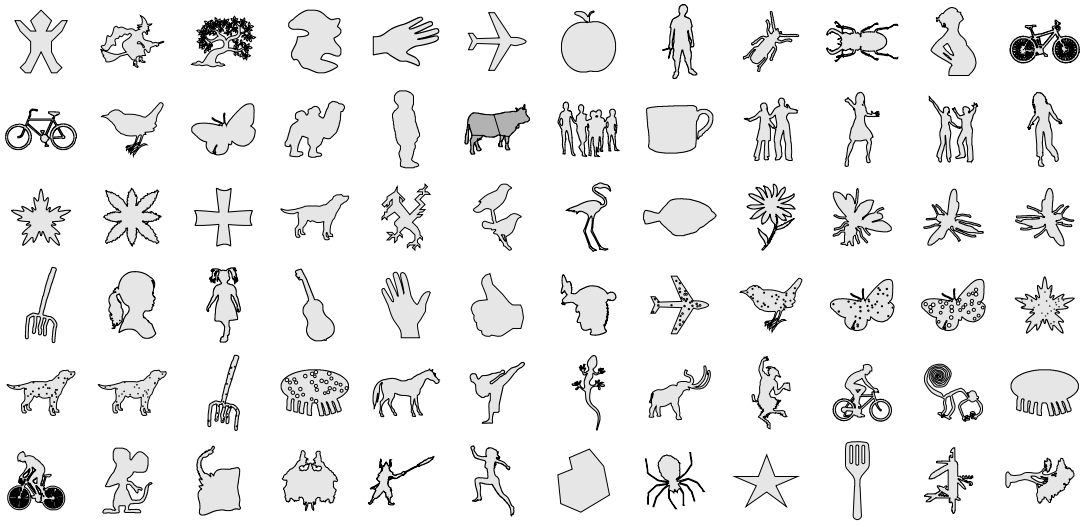
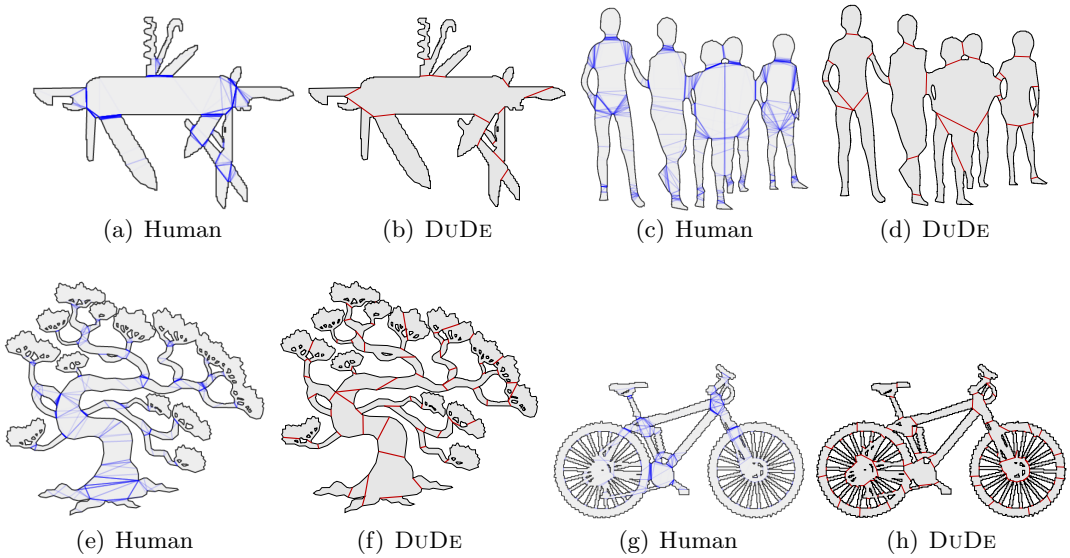


Figure 2.18: Shapes used in the experiments

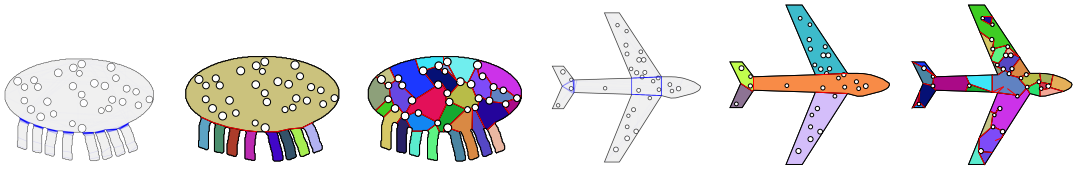


(a) Human (b) DuDe (c) Human (d) DuDe

(e) Human (f) DuDe (g) Human (h) DuDe

Figure 2.19: Differences between human segmentation and DuDe.

I have implemented DuDe in C++. To evaluate DuDe, 72 shapes are used as shown in Fig. 2.18. To provide a baseline for comparison, 3818 segmentation from 142 human subjects were collected. Many shapes are from the widely used MPEG7 database. All the images from MPEG7 contain only a single boundary. To test the polygons with many holes, synthesized noise are added to MPEG7 images and silhouette images, such as trees, human crowd, and bikes are used. In all experiments, the value of concavity tolerance τ is fixed for both DuDe , MNCD [38] and ACD [39]. The segmentation collected from human, DuDe, MNCD and ACD are then compared using two statistics-based evaluation methods (detailed in Section 2.3.4.2). The average running time for DuDe is less than 2 seconds on a regular laptop. Example output can be found in Figures 2.19 and 2.20. Complete segmentation results, the benchmark and the software tools are available to the public at *http://masc.cs.gmu.edu*.



(a) Human (b) DuDe (c) ACD (d) Human (e) DuDe (f) ACD

Figure 2.20: Decomposition results of the aggregated human cuts, DuDe and ACD.

2.3.4.1 Collect Human Segmentation

I use Amazon Mechanical Turk to collect human segmentation. Non-expert users are asked to draw lines to segment the displayed polygon into *meaningful* parts via a web-based interface. Several good and bad examples were shown before the user starts. Constraints were added to prevent user making segments outside or intersecting with the polygon or other existing cuts. There were in total 142 people participated, and 3818 valid segmentation were obtained. On average, each user spent around 90 seconds to segment a polygon, and made about 8.12 cuts for a polygon. To visualize the segmentation, I overlay user created segments, each of which is drawn translucent therefore the most frequently picked segments are darker. Examples of human segmentation can be found in Figures 2.19 and 2.20.

2.3.4.2 Evaluate Part Segmentation

Although few works focusing on evaluating part segmentation, evaluating image segmentation has been widely investigated (see a survey in [28]). In this section, two statistical methods are used for the evaluation. One is based on *Rand Index* (RI) [83] and the other one is called *Cluster Coverage* (CC).

To perform RI evaluation, the representative human segments are obtained via k-means clustering for each polygon. Here k is determined by the mean number of cuts created by human for a given polygon.

Next, I overlay the decomposition with a 200×200 regular grid tightly bounding the

polygon P. A cell is considered as valid if it has more than half of its area inside P. For a given segmentation, each component in the segmentation is assigned a unique index and the cells enclosed by this component are assigned the same index number. Different from Section 2.2, the original definition of RI is used here. Namely, RI evaluation of two segmentations (of the same polygon) is measured by the likelihood that each pair of cells is in the same component of two segmentations [83]. More specifically, let S_1 and S_2 be two segmentations, and s_i^1 and s_i^2 be the indices of cell i in S_1 and S_2, respectively. Then their RI is defined as:

$$\mathrm{RI}(S_1, S_2) = \binom{2}{N}^{-1} \sum_{i,j,i<j} \left[C_{ij} P_{ij} + (1 - C_{ij})(1 - P_{ij}) \right] ,$$

where N is the number of cells inside the original polygon, $C_{ij} = 1$ if $s_i^1 = s_j^1$, $P_{ij} = 1$ if $s_i^2 = s_j^2$, $C_{ij}P_{ij} = 1$ if cell i and cell j have the same index in both S_1 and S_2, and, finally, $(1 - C_{ij})(1 - P_{ij}) = 1$ indicates the cell i and j have different indices.

A major drawback of the RI-based evaluation is that the size of the representative cuts can affect the RI value dramatically. Therefore, another evaluation method called *Cluster Coverage* is proposed. A Cluster Coverage (CC) value between a segmentation S and clusters C of segmentation created by the human subjects is determined in the following way:

$$\mathrm{CC}(S, C) = \frac{\sum_i^k W_i f_i}{\sum_i^k W_i} ,$$

where W_i is the number of elements in C_i and f_i is an indicator function that returns one if $\exists S_j \in C_i$.

The RI and CC values in Table 2.2 are obtained from the evaluations between segmentations created by a given method and the representative human segmentations over all the polygons shown in Fig. 2.18. Therefore, for both RI and CC evaluations, higher score

indicates higher similarity to the representative human segmentations. Human vs. Human comparison is used as the baseline for the comparison. From Table 2.2, it can be seen that the average RI value of DuDe is only slightly below that of Human, while higher than those from MNCD and ACD. This means DuDe provides segmentations closer to the aggregated human segmentations. The average CC value of Human is only slightly below that of DuDe while the standard deviation of DuDe is slightly higher. The CC evaluation again confirms that DuDe provides segmentation closer to the aggregated human segmentation than MNCD and ACD.

Table 2.2: Mean values of Rand Index (RI) and Cluster Coverage (CC) and their standard deviations SDV_{RI} and SDV_{CC}.

Method	Human	DuDe	MNCD	ACD
RI	0.73	0.68	0.46	0.39
SDV_{RI}	0.12	0.21	0.17	0.36
CC	0.67	0.77	0.73	0.40
SDV_{CC}	0.094	0.15	0.26	0.38

2.3.4.3 Compare with CSD and MNCD

Both CSD [36] and MNCD [38] generate candidate cuts and mutex pairs using Reeb graphs from multiple rotations. The major difference between DuDe and these methods is that DuDe can handle holes (that appear frequently in shapes of many overlapping objects) much more naturally. Figure 2.21 shows the difference between the decompositions of MNCD and DuDe. Consequently, as shown in Table 2.2, DuDe can generate more similar results than MNCD compared to human segmentations. The reasons of lower RI and CC scores may also be coming from the fact that there are insufficient optimal candidate cuts in their methods that are generated from intersection of scanning lines and polygon boundary. Moreover, for polygon with complex holes, MNCD and CSD require parameters to balance

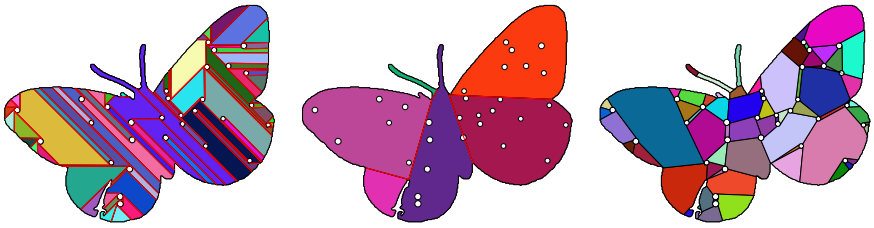


Figure 2.21: Decomposition result of MNCD and DuDe. From left to right: MNCD ($\tau = 0.1$), DuDe ($\tau = 0.1$), DuDe ($\tau = 0.01$).

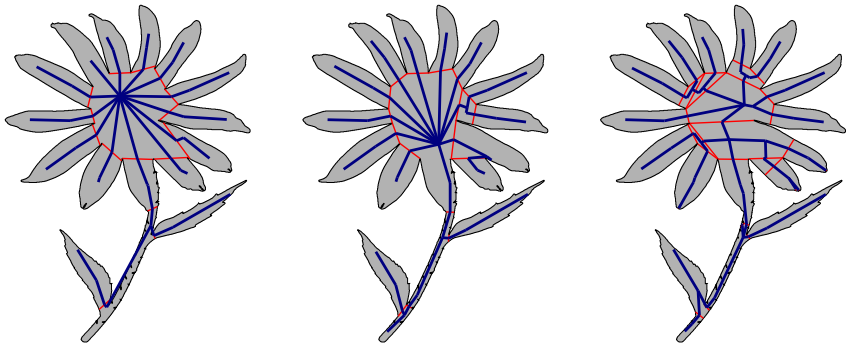
between the visual naturalness and the number of cuts for different shapes (while DuDe requires only the value τ). Find good values for these parameters is usually not easy. In terms of computation efficiency, for polygons with over several thousand vertices, based on my implementation, CSD and MNCD usually take several minutes to solve the optimization problem.

2.3.5 Discussion & Future Work

In this section and also paper [1]., a new method: *Dual-space Decomposition* (DuDe) to do 2D shape decomposition is proposed, which has applications in region/part-based recognition, skeleton-extraction (Fig. 2.22), etc. DuDe is designed to segment shape composed overlapping objects with a significant number of holes. Using two evaluation methods: Rand Index and Cluster Coverage, it has been shown that segmentation created by DuDe is statistically more similar to the man-made segmentation than those created by other methods.

Since DuDe is developed purely based on the geometric properties, its ability to segment meaningful part is limited. For example, collected data shows that people are good at segmenting shapes composed of humans and animals. When a figure contains human shape and other non-human objects, people prefer to separate human from the models.

Although there has been many methods recently proposed in segmenting 3D shapes, 2D part segmentation has its own difficulty because 2D shapes tend to overlap thus create



(a) Human (b) DuDe (c) ACD

Figure 2.22: Skeleton extracted from different methods

ambiguity. When (either human or non-human) shapes overlap, people also tend to segment each individual object or human from the group, even when the cut is long (e.g., the crowd shape in Figure 2.19). Moreover, different people also assign the overlapping area to different parts. More people tend to assign the overlapping area to an object if it has larger ratio of non-overlapping parts in the whole shape rather than other objects that share this overlapping area. For example, in the bike polygon in Figure 2.19, when the frame has overlapping area with wheel, more people tend to assign the area to wheel instead of frame.

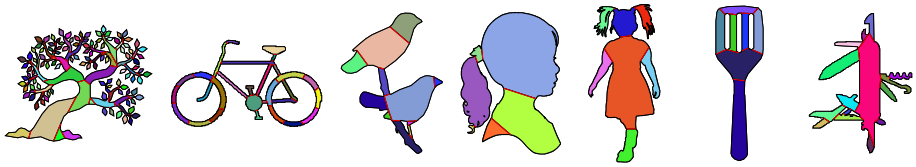


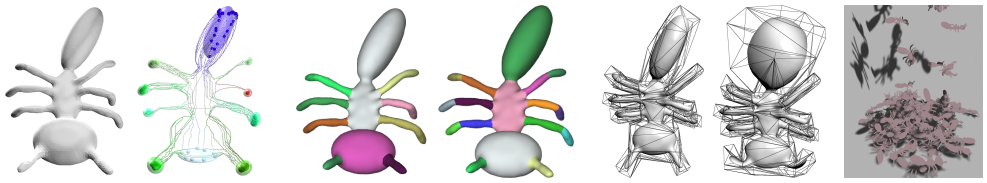
Figure 2.23: Decomposition results of DuDe for polygons with many holes.

2.4 Nearly Convex Decomposition of 3D Shapes through Convex Ridge Separation

2.4.1 Introduction

In interactive applications, it is necessary to create simplified representations of a mesh to support various computationally intensive operations. In this work, one goal is to obtain such simplified representations via decomposition. Taking deformation as an example, a mesh that has been decomposed into visually meaningful parts (e.g. head, torso, limbs) eases the process of creating deformation at semantic level. On the other hand, if a mesh is caged and partitioned by a set of convex shells, artists can use these shells to perform physically-based deformation efficiently. In many situations, both of these higher-level (semantic) and lower-level (physical or geometric) deformations are required. While it is ideal to keep both representations, it is also desirable to have an unified representation. An unified representation not only reduces space requirement but also allows deformations created at various semantic levels to be applied to the original mesh through a single approximation. Therefore, there is a demand to have decomposition methods, such as the one proposed in this paper, that provide users both visual saliency and bounded geometric properties.

The first type of decomposition above is called *shape decomposition*. In the past decade, significant progress has been made in producing high quality results [27, 45, 46, 52, 54, 65, 75]. Shape decomposition provides implicit shape approximation and is useful for shape analysis and recognition and semantic level shape editing and deformation. The second type of decomposition can be accomplished through the Approximate Convex Decomposition (ACD) [39]. ACD decomposes a 3D mesh into nearly convex parts. Unlike shape decomposition, ACD provides explicit approximation with bounded approximation errors and is therefore suitable for lower level processing. For example, Bullet physics library uses hierarchical ACD (HACD) [77] to speed up the collision detection and response computation



(a) Input (b) Convex ridges (c) Iterative decomposition (d) Cage-based deformation (e) Physically-based simulation

Figure 2.24: An example of CoRiSe with concavity tolerance $\tau = 0.05$. (a) Input mesh. (b) Ten convex ridges (shown as translucent ellipsoids) are connected by *deep valleys* (shown as the geodesic paths). Formal definition of convex ridge and valley can be found in Section 2.4.3. (c) Left is the decomposition result of the first iteration; right is final decomposition (the second iteration). Note that colors are just used to distinguish different parts. (d) CoRiSe used for automatic cage generation. (e) The convex hulls of CoRiSe components can be used to speedup the computation in physically-based simulation.

of the non-convex shapes, and Muller et al. [96] proposed an interactive tool to generate approximate convex parts for speeding up dynamic fracture simulation.

The most challenging task in developing such a decomposition method stems from the fact that current convex decomposition and shape segmentation methods are computationally expensive and require different parameter settings for different shapes which are usually unintuitive and make processing a massive number of meshes in applications difficult. Recently, learning based segmentation methods [45, 46] are proposed to learn common parameters in an unsupervised or supervised manner, but the computation time of these methods is often intolerably high (minutes to hours), in particularly, for interactive applications.

Contribution In this section, an efficient decomposition method, called CoRiSe, is proposed. The only user input parameter is a concavity tolerance which directly controls the approximation error. The novelty of CoRiSe comes from the idea of *convex ridge* which can be efficiently and robustly determined. Essentially, CoRiSe extracts nearly convex components of a 3D mesh P by separating each convex ridge from all the other convex ridges using graph cut. An example of convex ridge and CoRiSe decomposition is shown in Fig. 2.24. Formal definition of convex ridge can be found in Section 2.4.3. In addition,

through the new concept of *residual concavity*, CoRiSe is insensitive to the parameter in graph cut and requires only a single user parameter: concavity tolerance. Section 2.4.4 will discuss residual concavity and graph cut.

Using the Princeton Segmentation Benchmark, it will be shown that CoRiSe generates noticeably better segmentation in significant shorter time than the existing methods [65, 77].Finally, applications of CoRiSewill be demonstrated, including physically-based simulation, cage generation, model repair and mesh unfolding in Section 5.5. Fig. 2.24 shows two of these applications.

2.4.2 Overview

CoRiSe takes a single 3D mesh and a concavity tolerance parameter τ as input and decomposes this mesh into nearly convex components in a top-down fashion that includes three main steps:

Step 1. Build bridges and compute concavities Bridges are the convex hull edges, and the shortest geodesic path on the mesh connecting the end vertices of a bridge is called *valley*. Then, the concavity of a point in the valley is measured as the shortest-path distance to the bridge. Exact shortest geodesic path and shortest-path distance are computationally expensive thus an efficient way to approximate them will be described. A valley is *deep* if the maximum concavity in the valley is greater than τ, otherwise the valley is *shallow*. It is important to remember that, throughout the entire segmentation process, concavity is only defined for points in the valleys and undefined elsewhere.

Step 2. Identify convex ridges In CoRiSe, a convex ridge consists of a group of convex-hull vertices connected *only* by shallow valleys. Intuitively, a convex ridge can be viewed as a *protruding* part of the mesh that is guaranteed to be convex enough. This implies that a continuous subset of the mesh containing two or more convex ridges must be too concave. Fig. 2.24(b) shows an example of 10 convex ridges.

Step 3. Partition using convex ridges CoRiSe segments the mesh by ensuring

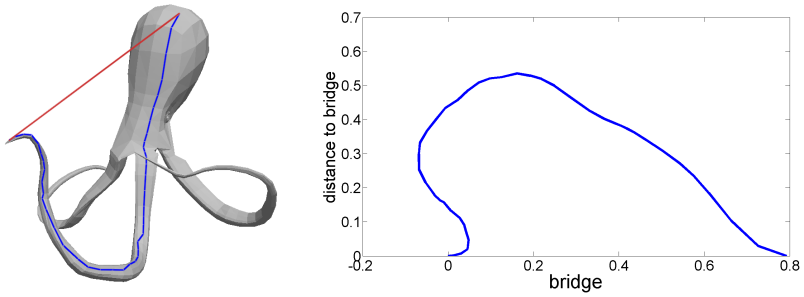


Figure 2.25: The left figure shows one bridge and its valley in 3D. The right part shows its projected concavity polygon in 2D. Concavities are measured in this 2D polygon using the shortest-path distance.

that each component in the segmentation contains only a single convex ridge. The segmentation process is bootstrapped by splitting each deep valley into shallow ones at the *valley separators*. Then CoRiSe applies graph cut using the region defined by valley separators as data term and edge angle and length as smoothness term to find optimal cut loops near these valley separators.

The above three steps are repetitively applied to the segmented part whose concavity is greater than τ.

2.4.3 Bridge, Valley and Convex Ridge

To approximate a mesh using convex shapes, the relationships between the mesh and its convex hull should be established. Let the input be a polyhedron $P = \{V_P, F_P, E_P\}$ composed of a list of vertices V_P and faces F_P connected by edges E_P. It is assumed that P consists of a single connected component. The convex hull of P is denoted as $H(P) = \{V_H, F_H, E_H\}$, where $V_H \subset V_P$. Let us consider a convex hull edge $e = \{a, b\} \in E_H$. Since e is the shortest Euclidean path connecting a and b, let us call e a *bridge*. Because a and b must also be the vertices of P, a path connecting a and b on ∂P can always be found. The shortest geodesic path connecting a and b on ∂P is called a *valley* underneath the bridge e_β.

A bridge e_β and its associated valley $e_\vee$ form a 3D polygon. This polygon is called the *concavity polygon*, which plays an important role in many stages of the algorithm. For

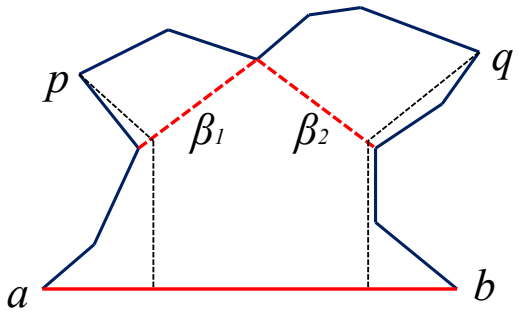


Figure 2.26: The concavity of the projected polygon with highly concave regions or self-intersections.

example, the concavity polygon is projected into a 2D space and only the concavity in the projected 2D space is measured. More specifically, each point $p \in e_\vee = (a, b)$ is projected to

$$p' = (\overrightarrow{ap} \cdot \overrightarrow{ab}, \mathbf{d}(p, e_\beta)) \ , \tag{2.6}$$

where $\overrightarrow{ap} \cdot \overrightarrow{ab}$ is the inner product of $\overrightarrow{ap}$ and $\overrightarrow{ab}$, and the function $\mathbf{d}$ defines the Euclidean distance between a point and a line segment. Fig. 2.25 shows an example of bridge, its valley and the projected 2D concavity polygon. Then, the concavity $C(p)$ of the point p is determined as the *shortest-path distance* between p' and the projected e_β. It is known that the shortest-path distance can be measured in $O(n)$ time for a polygon of size n.

It is possible that the projected polygon can self-intersect. However, self-intersection seldom happens; all projected polygons of models studied in this work are free of self intersection. However, when the projected polygon is self-intersecting or has large concave regions, a more sophisticated concavity measure proposed in [1] will be used, which decomposes the polygon into several near convex parts and determines a hierarchy of bridges. The concavity will be measured by the accumulated distance of the point-to-adjacent bridge distance and bridge-to-bridge distances. For example, in Figure 2.26, the concavity of p will be the distance sum of the distance between p and β_1 and the distance between β_1 and ab.

The concavity of a valley $e_\vee$ (and its associated bridge e_β) is defined as the maximum

concavity of its vertices, i.e., $C(e_\vee) = C(e_\beta) = \max_{p \in e_\vee}(C(p))$. Let us call $e_\vee$ *deep valley* if $C(e_\vee)$ is larger than the tolerance, otherwise $e_\vee$ is called a *shallow valley*. For the convenience of future discussions, bridges are classified into *high* and *low* bridges (as measured by the *deck height* of bridge) if their corresponding valleys are deep and shallow, respectively.

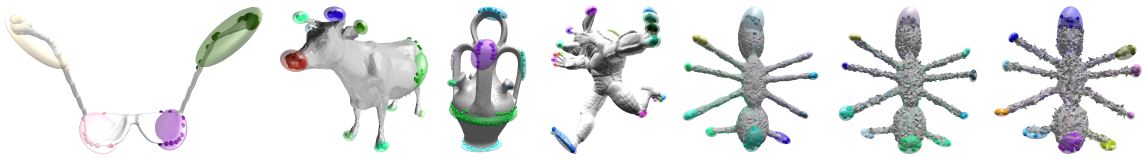
2.4.3.1 Convex Ridge

Now let us formally define the convex ridge. A convex ridge R of P is a graph representing a subset of convex hull $H(P)$. A subset of $R \subset H(P)$ is a convex ridge if all bridges of R are shallow and there are no high bridges connecting two vertices in R. More specifically, a convex ridge R must satisfy the following two conditions:

$$\begin{aligned} \forall e \in E_R, \quad & C(e) < \tau \text{ , and} \\ \nexists e = (a \in V_R, b \in V_R), \quad & C(e) > \tau \text{ ,} \end{aligned} \tag{2.7}$$

where V_R and E_R are the vertices and edges of R, respectively. In short, a convex ridge can only have vertices connected by bridges whose valleys are shallow.

The convex ridges of a given mesh is constructed by iteratively greedy clustering the convex hull vertices while ensuring that the properties in Eq. 2.7 is maintained for each cluster. The clustering proceeds by collapsing low bridges sorted in an ascending order based on the length of the associated valleys. Each final cluster would become a convex ridge. The detailed procedure can be found in Algorithm 3. Figure 2.27 shows examples of convex ridges. It can be seen that convex ridges are quite insensitive to surface noise in Figure 2.27. In fact, as long as shallow valleys remain shallow, all convex ridges will be unaffected.



(a) (b) (c) (d) (e) (f) (g)

Figure 2.27: Convex ridges found in these models. Each ellipsoid represents a convex ridge. Even though several convex ridges in these examples concentrate around small compact regions, the convex ridges in (c) manifest various ridge-like shapes (regions that are locally with higher "elevation"). Figures 2.27(e), 2.27(f) and 2.27(g) show the convex ridges on the ant model with random noise added. The convex ridges of figures 2.27(e), 2.27(f) and 2.27(g) are all generated using τ=0.08.

Algorithm 3 Build Convex Ridges

Sort bridges B in an ascending order by the valleys length
$V \leftarrow$ the end vertices of all bridges B
Create $||V||$ clusters each of which contains a vertex in V
for each shallow bridge $b_i = (a_1, a_2)$ in B **do**
    $g_1 \leftarrow$ the cluster contains a_1
    $g_2 \leftarrow$ the cluster contains a_2
    if $\nexists e = (b_1 \in g_1, b_2 \in g_2)$ such that $C(e) \geq \tau$ **then**
        merge g_1 and g_2
    end if
end for
Report each cluster as a convex ridge

2.4.3.2 Residual concavity

Residual concavity measures the concavity of a valley $e_\vee$ after $e_\vee$ is split into two subpaths at a vertex of $e_\vee$. Let $e_\vee = \{v_0, v_1, \cdots, v_{n-1}\}$ be a deep valley, and let $e_\vee^k = \{v_0, v_1, \cdots, v_{k-1}\}$ be a prefix subset of $e_\vee$, where $k \leq n$. Moreover, let $\hat{e}_\vee = \{v_{n-1}, v_{n-2}, \cdots, v_0\}$ be the reverse of $e_\vee$. The residual concavity of $e_\vee$ at the k-th vertex is then defined as:

$$RC(e_\vee, k) = \max\left(C(e_\vee^k), C(\hat{e}_\vee^{(n-k+1)})\right) \ . \tag{2.8}$$

Recall that the concavity is always measured in the 2D concavity polygon projected using Eqn.2.6. It is important to note that once the valley $e_\vee$ is split, two or more new bridges must be formed to determine the concavity of $e_\vee$'s sub-valleys. Therefore, at the first glance, computing RC seems to be time consuming, but it has been shown that RC can be computed in linear time.

Lemma 1. The computation of residual concavity $RC(e_\vee, k)$ takes time linear to the size of $e_\vee$, i.e. $O(n)$ for $e_\vee$ with n edges.

Proof. Similar to the previous definition, the concavity of a prefix $e_\vee^k$ is defined as the shortest-path distance between $e_\vee^k$ and edges of $H(e_\vee^k)$ inside the 2D concavity polygon (made of edges of e_β and $e_\vee$). Due to the special property of the concavity polygon $P_\vee$, whether a convex hull edge $e = (v_i, v_j)$ of $H(e_\vee^k)$ is inside $P_\vee$ can be determined by the order of v_i and v_j. That is, if $i < j$, then $e = (v_i, v_j) \subset P_\vee$. Since $e_\vee^k$ is a polyline, $H(e_\vee^k)$ can also be constructed in linear time $O(k)$, which makes the computation of residual concavity $RC(e_\vee, k)$ linear to the size of $e_\vee$, i.e. $O(n)$. □

2.4.3.3 Shape of a valley

The residual concavity gives us a way to estimate the shape of a valley. A valley $e_\vee$ is *V-shaped* if there exists a vertex v_k such that the residual concavity of $e_\vee$ is less than the tolerance τ. Otherwise $e_\vee$ is *U-shaped*. An example of a U-shaped valley can be found in Fig. 2.28. If $e_\vee$ is U-shaped, it is always true that the bottom of $e_\vee$ as the sub-polygon bounded by the prefix and suffix of $e_\vee$ that have residual concavities less than τ. In Fig. 2.28, the bottom of the U-shaped valley is defined as (v_i, v_j), where i and j are the maximum indices such that $C(e_\vee^i) < \tau$ and $C(\hat{e}_\vee^{(n-j)}) < \tau$, respectively; recall that $\hat{e}_\vee$ is the reverse of $e_\vee$. As what will be discussed in Section 2.4.4, the shape of valley can be used to determine

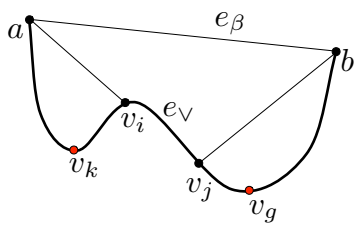


Figure 2.28: A deep U-shaped valley $e_\vee$ and its valley separators v_k and v_g. The sub-valleys $e^i_\vee$ and $\hat{e}^j_\vee$ are subsets of $e_\vee$ whose residual concavity is less than the concavity tolerance τ. The valley separators $v_k \in e^i_\vee$ and $v_g \in \hat{e}^j_\vee$ minimize the residual concavities $RC(e^i_\vee, k)$ and $RC(\hat{e}^j_\vee, g)$.

the number of vertices needed to separate each convex ridge in the wired representation to ensure that its concavity is less than τ.

2.4.4 Convex Ridge Separation

CoRiSe segments a mesh by ensuring that each component in the segmentation contains only a single convex ridge. Essentially, CoRiSe finds such segmentation in two steps: first, CoRiSe only focuses on the wireframe representation of the mesh that consists of vertices and edges of all convex ridges and deep valleys. An example of such a representation can be found in Fig. 2.24(b). For each deep valley, CoRiSe determines one or two key vertices (depending on whether it is a V- or U-shaped valley) that can separate each deep valley into at least two shallow valleys. Section 2.4.4.1 will discuss how CoRiSe decomposes this wireframe representation. Then, once the wireframe is decomposed, CoRiSe switches back to the original model and applies a two-way graph cut to find optimal cut loops near the valley separators. This step will be discussed in Section 2.4.4.2.

2.4.4.1 Valley separators

To separate the convex ridges in the wireframe presentation, CoRiSe finds *valley separators* for each deep valley and ensures that each convex ridge only connects the subset of the valley

that has concavity less than the tolerance. For a V-shaped valley $e_\vee$, the valley separator is a single vertex that has the largest concavity in $e_\vee$. By splitting at the vertex with the largest concavity, it guarantees that the two new sub-valleys of $e_\vee$ are shallow.

For a U-shaped valley $e_\vee$, things are a bit more complex. Basically, CoRiSe splits $e_\vee$ into three sub-valleys and guarantees that two sub-valleys incident to two convex ridges are shallow even though the bottom of $e_\vee$ may not be. Let $e_\beta = (a, b)$ be a bridge whose valley $e_\vee$ is U-shaped. Using the residual concavity defined in Eqn. 2.8 in Section 2.4.3.2, it can be found that two vertices v_i and v_j such that $e^i_\vee$, a sub-valley between a and v_i, and $\hat{e}^j_\vee$, a sub-valley between v_j and b, are shallow. See Fig. 2.28 for the illustration of $e^i_\vee$ and $\hat{e}^j_\vee$. The goal is to define two valley separators in $e^i_\vee$ and $\hat{e}^j_\vee$, respectively. It is true that it is still guaranteed that $e^i_\vee$ and $\hat{e}^j_\vee$ are shallow if using v_i and v_j as the valley separators. However, the locations of v_i and v_j are quite arbitrary in most cases. Therefore, the valley separators for a U-shaped valley $e_\vee$ are defined as a couple of vertices (v_k, v_g):

$$(v_k, v_g), \text{where } k = \arg\min_k RC(e^i_\vee, k), g = \arg\min_g RC(\hat{e}^j_\vee, g)) \ .$$

From Fig. 2.28, it can be seen that the valley separators v_k and v_g of $e_\vee$ are identified at the locations of large concavity if the valley is split at v_i and v_j. Once the valley separators are identified for all deep valleys, all convex ridges are separated and only attached to shallow valleys. Next, CoRiSe switches back to the original representation and segments the mesh using graph cut.

2.4.4.2 Convex ridge separator

Graph cut [97] is a powerful optimization tool that has been proven to be successful in segmentation. The previous section has discussed how to find separators for deep valleys. And the valley separators have roughly defined where the cuts should be. However, since the valley separators are merely feature vertices capturing concavity constraints, separating

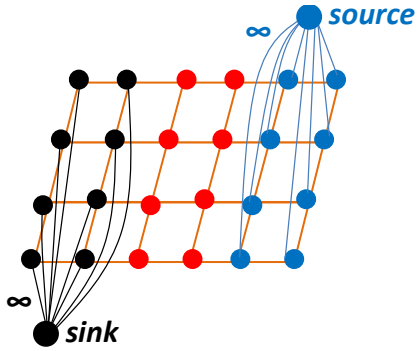


Figure 2.29: Simplified graph cut: black nodes and blue nodes represent the must-link regions. They have infinite data term associated with source or sink. Red nodes represent the fuzzy region.

convex ridges needs close cut loops. Finding the cut loops is formulated as a simplified graph cut problem. It is called simplified graph cut because the data term value is either infinity or zero, which corresponds must-link region and fuzzy region. The smoothness term is defined using dihedral angle and edge length. Fig. 2.29 shows the simplified graph cut formulation. In the rest of this section, the data term and smoothness term will be discussed in detail.

Data Term To formulate the graph cut problem, two terms: *must-link region* and *fuzzy region* are introduced. *Must-link region* would have infinite data term values associated with the *source* or *sink*. *Fuzzy region* would have zero data term values. For each iteration of graph cut, a facet belongs to either *must-link region* or *fuzzy region*. In other words, *must-link region* plus *fuzzy region* will cover the whole model. Intuitively, each convex ridge has a must-link region which is a set of faces that must be segmented with the convex ridge. To get the *must-link* region and *fuzzy region*, another term named *potential region* will be introduced. It is called *potential region* because a facet belonging to *potential region* has the chance to become either part of *must-link region* or part of *fuzzy region*. The ways to determine these regions are described as following:

Potential Region. A *potential region* will be collected for each convex ridge R_k. This potential region is the union of the facets collected by R_k's all the sub-valleys using geodesic

distance. The way of how each sub-valley collects its facets and doing the union are described as following:

1. Collecting using Geodesic Distance: for each sub-valley $e^i_\vee$, ($e^i_\vee$ is created from valley separator as described in Section 2.4.4.1) incident to R_k, let T_i be a set of facets whose geodesic distance to the end point of $e^i_\vee$ in R_k is less than the path length of $e^i_\vee$.

2. Union: from T_i, a set U_{R_k} of facets that forms a superset containing R_k can be determined. The set U_{R_k} is the *potential region* of convex ridge R_k. More specially,

$$U_{R_k} = \bigcup_{1 \le i \le n} T_i \ , \tag{2.9}$$

where n is the number of sub-valleys incident to R_k.

Potential Region's Overlapping Issue. Two neighboring convex ridges R_k and R_j are likely to compete on certain facets. In this case, U_{R_k} and U_{R_j} overlap. The overlapping region of I_{R_k} with other convex ridges is then defined as:

$$I_{R_k} = \bigcup_{\forall j \neq k} \left(U_{R_k} \bigcap U_{R_j} \right) \ . \tag{2.10}$$

Must-link and Fuzzy Regions. With U_{R_k} and I_{R_k}, it is ready to define the must-link and fuzzy regions. The must-link region C_{R_k} of R_k is simply U_{R_k} with facets in I_{R_k} removed,

$$C_{R_k} = U_{R_k} \setminus I_{R_k}. \tag{2.11}$$

Then the fuzzy region is the union of the facets that do not belong to any convex ridge's *must-link region*.

$$F_R = F \setminus \bigcup C_{R_k}, \tag{2.12}$$

where F is the facets of the mesh.

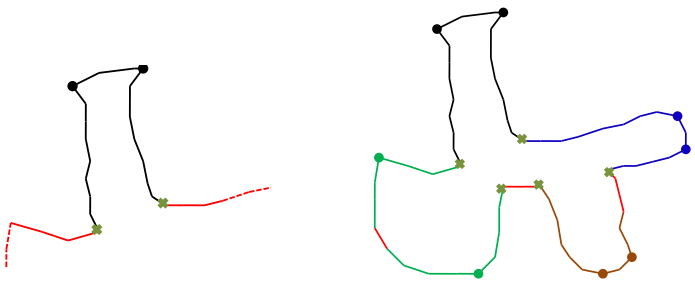


Figure 2.30: Left: the potential region (including all the black and red regions). Right: must-link region and fuzzy region. In the right part, the convex ridges are differentiated by colors (4 in total). The colored circles (dots) are the end vertices of the deep valleys. The crossings are the detected valley separators. The red lines (region) are the fuzzy region. The colored lines attached to circles with same color are must-link region attached to corresponding convex ridge.

An example is shown in Fig 2.30. In the left part of Fig 2.30, the black segments show the potential region of a convex ridge. The black vertices are the vertices of a convex ridge. The crossing vertices are the valley separators. The right part of Fig 2.30 shows the must-link region and fuzzy region. The red segments represent the fuzzy region. Except the red, each of other colors stands for the must-link region for each convex ridge. There are three parts of fuzzy region (red regions). The left part corresponds to the area that is not covered by deep valley's potential region. The middle part is part of fuzzy region because it lies in the middle part of a U-shape valley and it is not within the potential region of either of end vertices from two convex ridges. The right part becomes fuzzy region due to the fact that it has been covered by two convex ridges.

Fig 2.31 shows the must-link regions on the 3D models. The filtered out regions are the fuzzy regions.

Smoothness Term The smoothness term evaluates the similarity/compatibility between two adjacent triangle facets. CoRiSeuses the same definition of smoothness as that in [49].

A parameter α is introduced to balance the importance of dihedral angel and edge length. The definition is as below: Let v_i and v_j be two adjacent vertices, namely are the

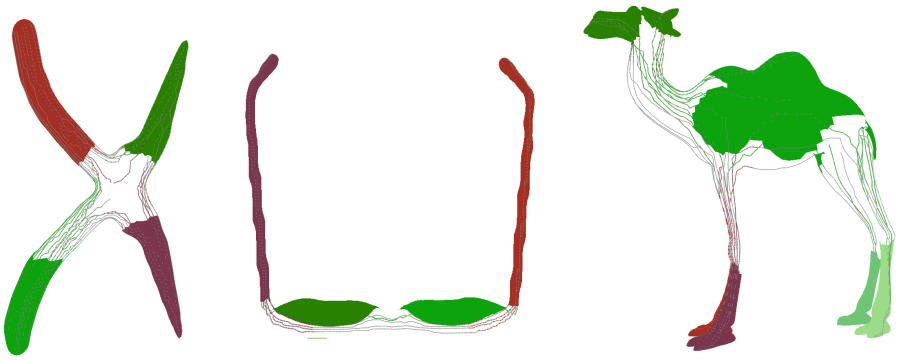


Figure 2.31: The must-link regions connected by the deep valleys.

two end vertices of an edge (denoted as e_{ij}) on the mesh. And let f_i and f_j to be the two triangle facets sharing this edge. Let θ_{ij} be the dihedral angle between f_i and f_j, $|e_{ij}|$ be the length of e_{ij}. Let Θ_{ij} be a function of θ_{ij} such that Θ_{ij} is a positive small number (the second parameter, fixed to be 0.3) for concave edges and 1 for convex edges. Then the smoothness term defined in [49] is:

$$w_{ij} = \alpha(\frac{\Theta_{ij}}{\mu_\Theta}) + (1-\alpha)\frac{|e_{ij}|}{\mu_e} \ , \tag{2.13}$$

where μ_Θ and μ_e are the average values of Θ and edge length, respectively, of the entire mesh. α is set to be 0.8 (the third parameter, fixed) in all experiments.

Segmentation using Graph Cut. Binary graph cut optimization is solved for $n-1$ times (n is the number of convex ridges) using the max-flow algorithm. The order of $n-1$ graph cut is determined by the bounding box's diameter of convex ridges' vertices. One convex ridge is separated at one time using the graph cut. At each graph cut iteration i, CoRiSEwill assign infinite data term values associated with *source* for the facets in its *must-link region* C_{R_i} and assign infinite data term values associated with *sink* for the facets in other convex ridges' *must-link region* $\bigcup_{\forall j \neq i} C_{R_j}$. Facets in *fuzzy region* don't have data term with neither *source* nor *sink*. The smoothness term will be defined for any pair of

neighboring facets using the Equation 2.13. This convex ridge will be separated from the model and the rest of model will be used for applying region for next binary graph cut. Segmentation is performed iteratively until all components have smaller concavity than τ. Fig 2.24 (c) shows the first iteration decomposition result on the left and second (final) iteration decomposition result on the right.

Discussion One may argue that there are three issues in the above formulation of graph cut problem. 1. Why use must-link region? The must-link region is defined using valley separators, which roughly defines where the cut should be and satisfied the concavity constraints. The must-link region can fully utilize the rough cut positions proposed by valley separators. 2. Why use $n-1$ binary graph-cut? Another option is to k-way graph cut. However, choosing a right k is important for k-way graph cut. A proper k would be the number of final components. However, the problem is the number of final components is unknown and the number of convex ridges is not the real number corresponding to the number components. Some components may haven't been detected by the current convex hull. The real number should be $\geq n$. CoRiSEapplies $n-1$ times graph cut and postpone handling of the issue of undetected components in the further iteration of decomposition, because these components may be detected after CoRiSEhas already cut some components out of the model in previous iteration. 3. How to decide the order of cutting which convex ridge and how to deal with overlapping labeling issues? The order of applying graph cut is based on the heuristic that CoRiSEhandles the small components first (in terms of the bounding box's diameter of convex ridges' vertices). In for each time of graph cut, if it is applied on the same region, there may be one issue that a facet has been labeled as "source" for several times. In this paper, after applying one time of graph cut, CoRiSEseparate one convex ridges from the model and the rest of the model would be used as the applying region for further graph cut. In this way, the overlapping labeling issue would be avoided.

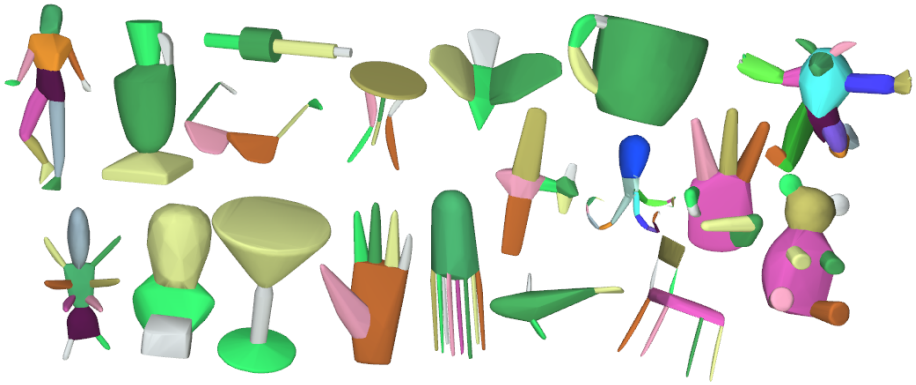
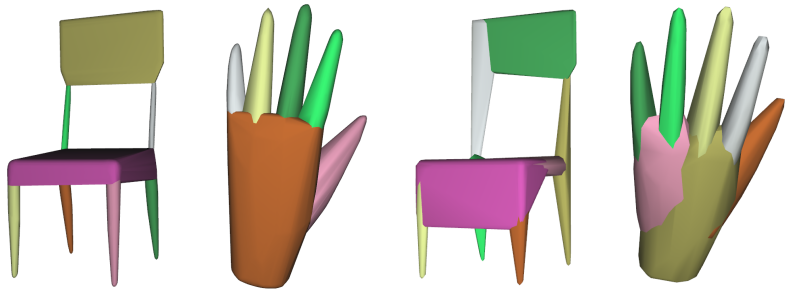


Figure 2.32: Approximate shapes using the convex hulls of decomposed parts. The results in this figure are generated by CoRiSe using models in the Princeton Segmentation Benchmark.

2.4.5 Experimental Results

CoRiSeis implemented in C++. In order to evaluate CoRiSe, CoRiSe is compared extensively to existing approximate convex decomposition methods HACD [77] that is available in the Bullet Physics Library and WCSeg [65]. CoRiSe is also compared with other shape decomposition methods using the Princeton 3D Mesh Segmentation Benchmark [28], which includes 380 surface meshes of 19 different object categories. All the models are scaled to have a 1.0 radius bounding sphere. The quality of the decomposition is measured via the Rand Index that estimates the similarity by measuring facet pairwise label difference between the segmentations generated by CoRiSe and those generated by humans and the state-of-art methods [27, 52, 54, 65]. All experimental results are collected on a workstation with two Intel Xeon 2.30GHz CPUs and 32 GB memory. HACD and CoRise are implemented using C++ and WCSeg uses MATLAB.



(a) CoRiSe (b) HACD

Figure 2.33: Shape approximation using the convex hulls of all components in the decompositions from CoRiSe and HACD.

2.4.5.1 Comparison of Approximation

Given a concavity tolerance, CoRiSe with HACD [77] and WCSeg [65] are compared based on (1) the number of decomposition components, (2) decomposition time, and (3) approximation accuracy. HACD first simplifies the model using Quadric Error Metrics (QEM) and then hierarchically merges facets and ensures that the concavities of all clusters are lower than the tolerance concavity. WCSeg creates initial segmentation using line-of-sight concavity/convexity measurement and then uses Shape Diameter Function [27] to merge the initial segmentations. Table 2.3 shows the decomposition size, computation time, and the ratio between the volume of the convex hull approximation and the volume of the original model.

From Table 2.3, it can be seen that, on average, CoRiSe generates fewer components than HACD and WCSeg. In addition, CoRiSe is significantly faster than HACD and WCSeg. This is because HACD requires many concavity evaluations in the bottom-up approach clustering process. Note that even though it is not totally fair to directly compare the computation time of CoRiSe and WCSeg shown in Table 2.3 due to WCSeg's matlab implementation, doing spectral clustering that involves solving the eigen-decomposition of a large pairwise matrix (whose size is square to the number of facets) would still be a bottleneck using the faster C++ implementation. Without simplifying models, HACD can

even take more than several hours to decompose a model with around 10,000 vertices.

Table 2.3: Statistical results of the number of final components, computation time and the volume ratio which is approximations' volume compared to that of the original model on the Princeton 3D Mesh Segmentation Benchmark. All results are obtained using concavity tolerance = 0.1.

Category	**# of components**			**Time (sec)**			**Volume Ratio**		
	HACD	WCSeg	CoRiSe	HACD	WCSeg	CoRiSe	HACD	WCSeg	CoRiSe
human	12.10	18.85	9.85	84.80	677.92	2.33	1.23	1.24	1.31
cup	18.90	5.50	4.55	372.09	1,362.30	1.25	2.66	3.02	3.15
glass	6.40	8.90	5.53	433.84	416.09	0.63	1.69	1.90	1.75
airplane	5.20	7.70	7.55	1,103.85	431.91	1.07	1.30	1.25	1.37
ant	13.45	11.25	14.50	223.99	413.63	1.76	1.11	1.17	1.14
chair	9.50	16.35	10.64	297.72	1,021.50	1.86	1.67	1.56	1.45
octpus	14.40	9.10	14.45	903.44	599.90	1.98	1.23	1.83	1.27
table	6.00	5.55	6.05	3,385.97	2,762.60	1.25	1.33	1.68	1.42
teddy	6.80	7.45	7.05	1,548.05	1,026.30	2.16	1.05	1.05	1.07
hand	9.10	9.20	9.00	205.92	670.00	1.67	1.14	1.16	1.27
plier	5.20	6.50	5.89	393.18	368.90	0.88	1.30	1.37	1.33
fish	5.05	6.25	5.20	2,969.80	546.04	0.84	1.12	1.16	1.18
bird	5.55	11.10	6.39	547.89	515.03	0.73	1.36	1.26	1.35
armadillo	15.30	20.10	14.90	56.89	1,914.80	5.46	1.12	1.12	1.19
bust	9.30	8.70	7.80	118.18	2,285.80	4.50	1.05	1.08	1.11
mech	4.45	3.70	3.65	1,286.43	2,587.30	1.04	0.87	1.02	1.02
bearing	5.47	3.00	1.58	1,095.92	1,188.90	0.25	0.95	1.08	1.17
vase	9.80	5.65	5.15	1,059.17	1,247.60	1.63	1.06	1.10	1.12
four-leg	12.95	14.05	12.25	61.85	614.97	1.86	1.20	1.23	1.25
Average	9.21	9.42	8.00	849.95	1,086.90	1.75	1.29	1.38	1.36

Table 2.3 also shows the comparison of volume ratios. The volume ratio is defined as $\text{vol}(\cup_i(CH_i))/\text{vol}(P)$, where $\text{vol}(\cup_i(CH_i))$ is the volume of the union of all convex hulls and $\text{vol}(P)$ is the volume of the input mesh. From Table 2.3, HACD has smallest average volume ratios. However, the result is biased because HACD simplifies the input model before decomposition, thus these convex hulls created by HACD do not guarantee to enclose the original model. This may also produce noticeable penetration if these convex hulls are used in collision detection. It also shows that the approximation volume of CoRiSe is slightly smaller than WCSeg. However, because WCSeg produces more segments, it can be said

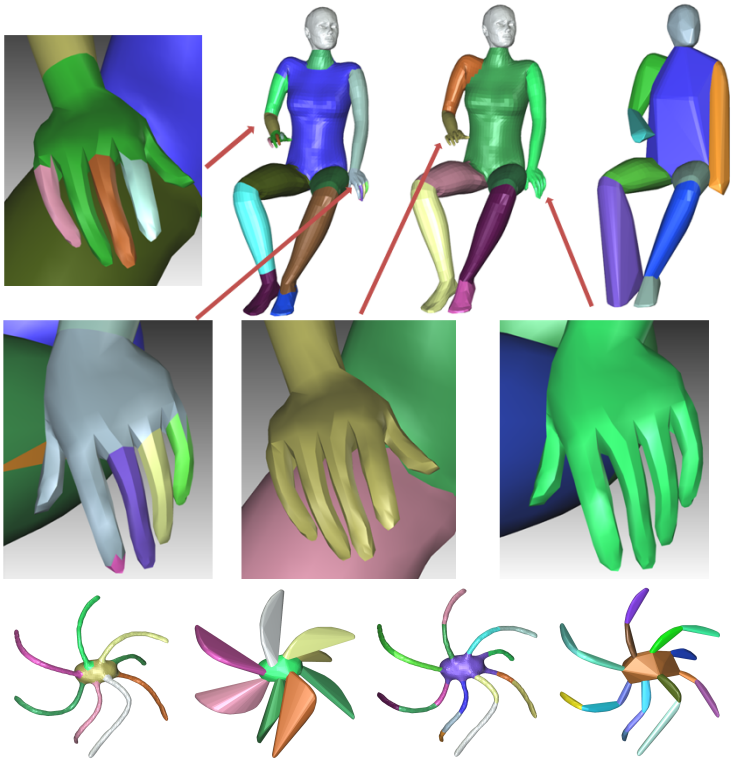


Figure 2.34: WCSeg vs. CoRiSe. Top row from left to right: WCSeg decomposition, CoRiSe decomposition and convex hulls of CoRiSe components. Bottom row from left to right: WCSeg decomposition, convex hulls of WCSeg components, CoRiSe decomposition, and convex hulls of CoRiSe components.

that CoRiSe is more effective (use less components to provide tighter approximation). This is especially true in the human and octopus category. Fig. 2.34 shows an example of the segmentations of a human model and an octopus model from both CoRiSe and WCSeg. The main differences between CoRiSe and WCSeg stem from how concavity is used for segmentation. CoRiSe cares about the approximation error, thus ignores the fingers of the human model because the concavity between two fingers is smaller than the concavity tolerance. On the other hand, the legs of the octopus model are bended and CoRiSe cuts the legs into several components to satisfy the concavity tolerance while WCSeg merges them together.

2.4.5.2 Comparison using benchmark dataset

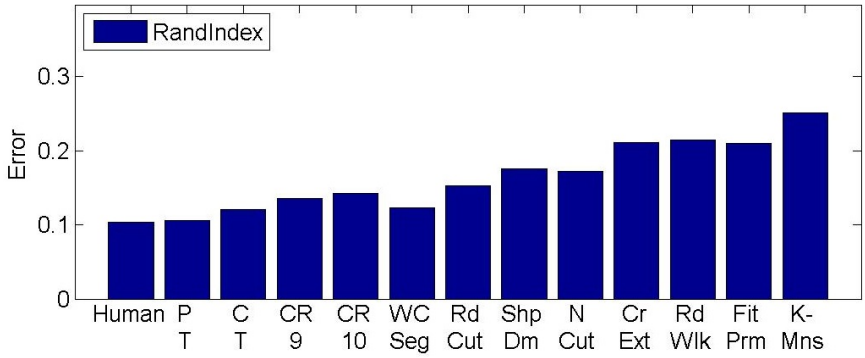


Figure 2.35: Rand Index (RI) comparison on the Princeton Benchmark data. **human**: human cut; **PT**: CoRiSe with a τ for each model; **CT**: CoRiSe with a τ for each category; **CR9**: CoRiSe with $\tau = 0.09$; **CR10**: CoRiSe with $\tau = 0.1$; Additional comparisons evaluated using other metrics such as consistency error, cut discrepancy, Hamming distance all show similar trend as RI and can be found in the supplementary materials.

Firstly, note that CoRiSe aims at resolving the concavity of a model and is not designed for semantic segmentation. For example, in Fig. 2.39, the bird is decomposed into over 30 parts which is much more than that of other shape segmentation. This will certainly influence the evaluation if CoRiSe is compared to segmentations created by human. However, I feel it is important to compare CoRiSe to shape segmentation methods since, after all, convexity is one of the important properties used in shape segmentation, and I also believe that a visually meaningful decomposition can provide visually convincing simulation.

Fig. 2.35 shows the Rand Index (RI) scores obtained from seven methods on the same benchmark. Although CoRiSe performs worse than the learning-based approach, CoRiSe outperforms some single-model based methods [27, 54] that require more user parameters, such as number of clusters. If CoRiSe can choose a concavity tolerance for each category, its performance is comparable to [65] which also has several parameters and thresholds. Moreover, if CoRiSe chooses concavity tolerance for each model individually, just as other

methods selecting component size for each model, its RI score is lower than all single-model methods in Fig. 2.35.

Compared with other shape-segmentation methods, CoRiSe has three major advantages: First, CoRiSe provides the bounded convexity that can be used by broader applications. Second, CoRiSe has less number of parameters, while others require several user parameters for each model to specify component number, soft-clustering number, or smooth factors. Table 2.4 summarizes the number of parameters of each method. Finally, CoRiSe is efficient without simplification as pre-processing.

Although it is true that only a small subset of concavity tolerances produce semantic decomposition of a given shape, those that do create semantic decompositions reflect the geometric property of the shape.

Table 2.4: Number of parameters of segmentation methods (including fixed parameters). **WCSeg**: the parameters will be used for encoding pairwise matrix, computing Shape Diameter Function and post-merging; **RC**: rand cut uses the statistics of other approaches' decomposition results. The parameter number will be more the sum of those approaches; **SDF**: Shape Diameter Function method ≥ 5 parameters (number of rays per facet; cone angle; component number of GMM; two fixed parameters in encoding smoothness term same as CoRiSe ; other parameters); **HACD**: many parameters in mesh simplification and thresholds for angles and edges during decomposition; **CORISE**: 3 parameters (the first is the concavity tolerance; the other two are the fixed parameters in encoding smoothness term).

WCSeg	RC	SDF	HACD	CoRiSe
> 5	> 5	≥ 5	7	3

2.4.6 Applications

CoRiSe is designed to approximate a 3D shape with a set of convex objects. In this section, several applications are demonstrated using this type of approximation.

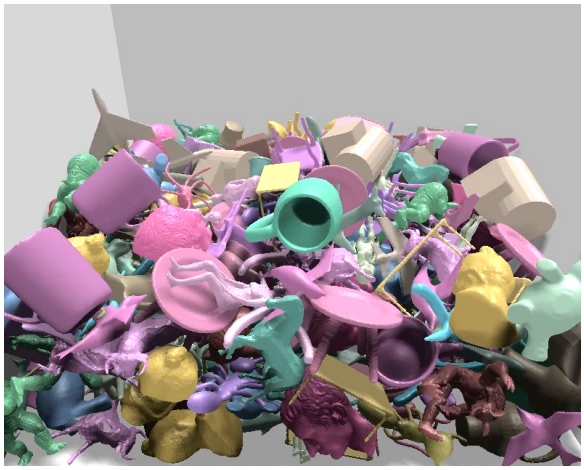


Figure 2.36: Simulation created using the convex hulls of CoRiSe.

2.4.7 Physically-based Simulation

Physics simulation libraries, such as Bullet and Box2D, use multiple convex shapes to approximate the original non-convex shape. Approximating a shape with bounded concavity error allows computations, such as collision response and penetration depth, become much more efficient. In these applications, CoRiSecares about the number of approximation components and the approximation error. Exact convex decomposition has no approximation error, however, the number of components is usually prohibitively huge, thus the computation, e.g., penetration depth, is inherently expensive. It is therefore desirable to have smaller number of components while the approximation error is bounded. From the comparisons between HACD and CoRiSe shown in Table 2.3, CoRiSe provides following advantages: 1). CoRiSe guarantees to enclose the original model while HACD does not; 2). CoRiSe is much faster than HACD; 3). CoRiSe produces smaller segmentation given the same concavity tolerance.

CoRiSehas been successfully integrated with the Bullet library. This allows us to generate convincing physically-based simulations using only the convex hulls of CoRiSe decompositions. I encourage the readers to the website masc.cs.gmu.edu to view the supplementary video.

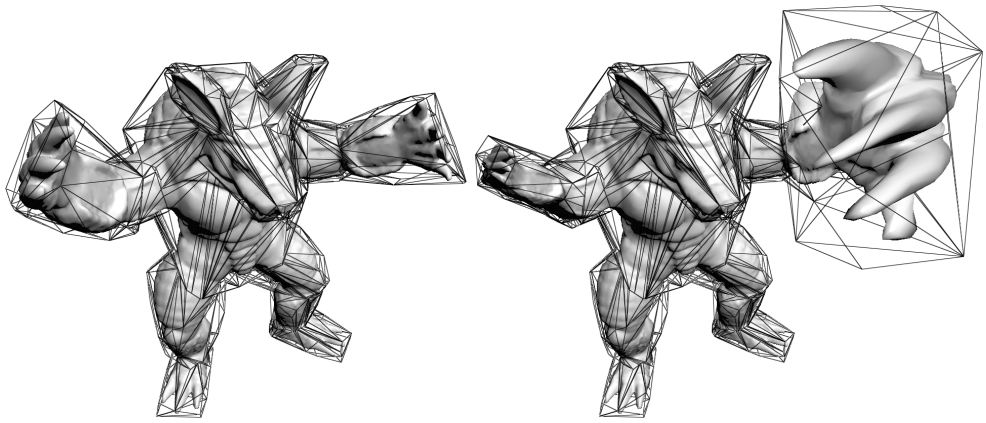
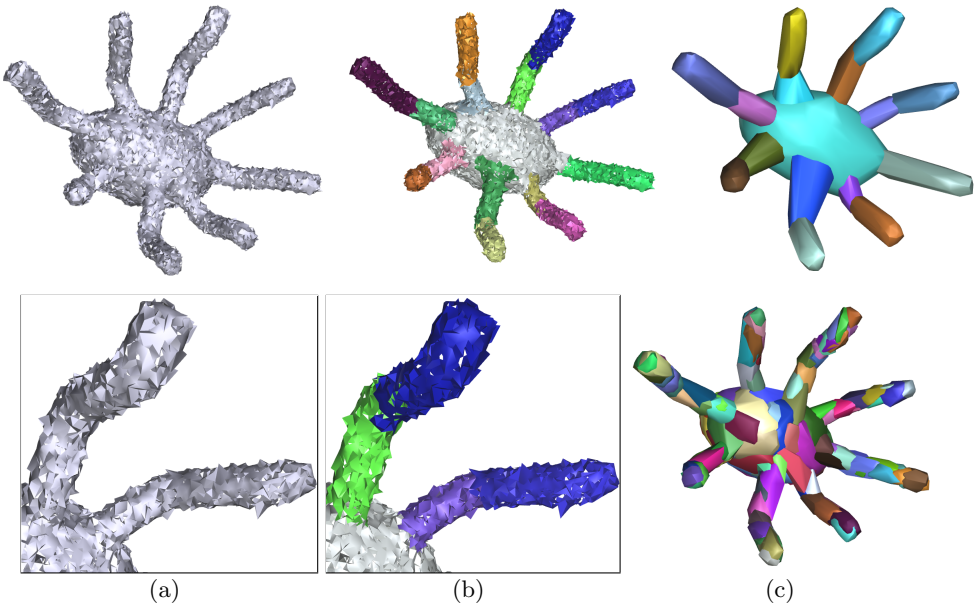


Figure 2.37: A cage created from CoRiSe decomposition.

2.4.7.1 Cage-Based Deformation

Cage based deformation technique aims to be an easy-to-use tool for graphics modelling, texturing and animation. The advantage of cage-based space deformation is its simplicity, flexibility and efficiency. Most cages in cage-based deformation are created manually, and the process can be tedious. Xian et al. [98] proposed an automatic cage generation technique based on iteratively splitting the bounding boxes. However, their method requires mesh voxelization that is both time consuming and, more importantly, ignores important structural features of the input shape. For example, their cage usually misses vertices near the concave regions of the input mesh. Moreover, their method is not suitable for interactive applications as for a shape with only 10k vertices, their method can take several minutes. CoRiSe generates a cage in almost real-time. CoRiSe first generates a nearly convex decomposition of the shape. Then convex hulls are computed for each part. These convex hulls are slightly expanded and simplified. The final cage is the union of the convex hulls. Fig. 2.37 shows a cage created for the Armadillo model and its deformation using Green coordinates. More examples can be found in the supplementary materials.



(a) (b) (c)

Figure 2.38: (a) Noisy mesh with missing facets. The bottom figure provides a close-up view. (b) CoRiSe results. (c) Top: Convex hulls of 16 segmented parts in (b). Bottom: Convex hulls of HACD segmentation (138 components). All results are obtained using concavity tolerance 0.1.

2.4.7.2 Model Repair

All the models used in Table 2.3 are watertight. However, many digitized models have degenerated features and may require mesh repair. The convex hulls of CoRiSe decomposition provide a convenient way to generate a watertight representation. Fig. 2.38 shows a mesh that has 45% of the facets removed, and CoRiSe successfully decomposes the mesh and produces better approximation than HACD does. In fact, CoRiSe will be able to produce similar results as long as the mesh edge connectivity has not been significantly changed. For models with many holes, CoRiSe can get help from methods such as [99] that can be used to ensure that the projected geodesic path of the bridges is still well defined.

2.4.8 Discussion & Future Work

In summary, in this Chapter and also paper [4], a 3D decomposition method that produces components with bounded concavity is proposed. The method called CoRiSe meets the needs of real-time simulations and computer games better than the existing methods. CoRiSealso shows that the decomposition results can produce semantically meaningful decomposition with low computation cost when proper concavity tolerances are given. The decomposition is achieved by identifying *convex ridges* and then solving a graph cut optimization problem formulated with *valley separators* and *must-link regions*. Comparing CoRiSe with other methods on the public benchmark dataset, CoRiSe can generate results close to manual decomposition. Comparing CoRiSe with other approximate convex decomposition methods, CoRiSe is significantly more efficient.

Known Limitations. Each of the major step of CoRiSe is quite robust. For example, the convex ridges are always identified at expected locations for all the tested models. A main limitation of CoRiSe is that its semantic meaning of CoRiSe components is significantly influenced by the concavity tolerance τ. When τ is too small, small components are produced and the semantic meaning of these components diminishes. For example in Fig. 2.39(c) and (d), even though CoRiSe still provides a tight approximation of the initial model using a relatively small concavity tolerance (0.05), many small components do not have significant meanings.

Another limitation of CoRiSe is that CoRiSe may not decompose a cup-like shape if none of the valleys reach the bottom of the cup's concavity (see Fig. 2.39(b)). This is caused by the fact that a valley is the (approximated) shortest geodesic path. One way to address this issue is to ensure that the vertex p with largest concavity of a valley has the locally maximum concavity. If this is not the case, then CoRiSeiteratively descends p until a point p' with locally maximum concavity is reached. A new valley is then computed by enforcing it to pass through p'.

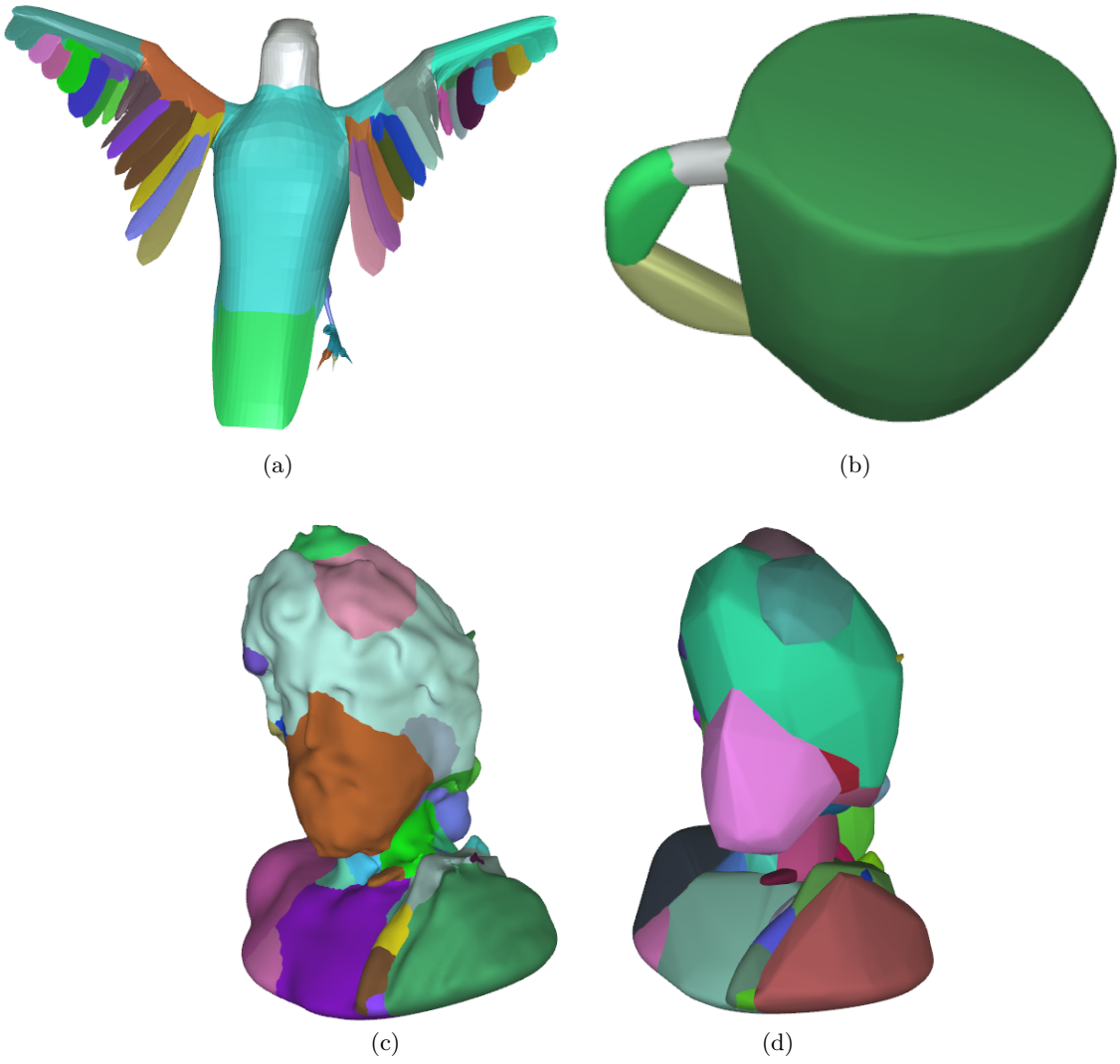


Figure 2.39: (a) CoRiSe result of a bird. (b) CoRiSe result of a cup. (c) CoRiSe result using $\tau = 0.05$. (d) The corresponding nearly convex approximation of (c).

Chapter 3: Appearance Synthesis - Material Editing

In this chapter, I focus on the appearance synthesis. Specifically, I will develop methods to editing the materials on 2D images. The ability to edit materials of objects in images is desirable by many content creators. However, this is an extremely challenging task as it requires to disentangle intrinsic physical properties of an image. An end-to-end network architecture that replicates the forward image formation process will be introduced to accomplish this task. Specifically, given a single image, the network first predicts intrinsic properties, i.e. shape, illumination, and material, which are then provided to a rendering layer. This layer performs in-network image synthesis, thereby enabling the network to understand the physics behind the image formation process. Due to the lack of corresponding training data, synthetic dataset will mainly be used to train the network. For multi-material cases, multi-material images will also be synthesized. To render such multi-material images, instead of naively assigning different materials for each pixel independently, I will assign different material part-wisely. The parts are obtained through segmenting the 3D models into meaning parts using the segmentation approaches described in Chapter 2 (If necessary, I will manually merge some of the parts as the post-processing). Both the results on synthetic images and real images will be demonstrated.

3.1 Introduction

One of the main properties of an object that contributes to its appearance, is material. Hence, many designers desire to effortlessly alter materials of objects. In case of 2D design, however, often the designer only has access to a single image of the object. Given one such image, e.g. an image of a fabric sofa, how would you synthesize a new image that depicts the same sofa with a more specular material?

A typical approach to addressing this image-based material editing problem is to first infer the intrinsic properties (e.g. shape, material, illumination) from the image which then enables access to edit them. However, this problem is highly ambiguous since multiple combinations of intrinsic properties may result in the same image. Many prior work has tackled this ambiguity either by assuming at least one of the intrinsic properties is known [6] or devising priors about each of these properties [7]. The recent success of deep learning based methods, on the other hand, has stimulated research to learn such priors directly from the data. Inferring intrinsic scene properties such as shape, material, and illumination from a single image has been a long standing problem in computer vision. This problem is highly ambiguous and ill-posed since multiple combinations of intrinsic properties may result in the same image. Many prior work has tackled this ambiguity either by assuming at least one of the intrinsic properties is known [6] or devising priors about each of these properties [7]. The recent success of deep learning based methods, on the other hand, has stimulated research to learn such priors directly from the data itself.

A feasible approach is to predict each of the intrinsic properties independently from an input image, i.e. using individual neural networks to predict normals, material, and lighting. However, the interaction of these properties during the image formation process inspires us to device a joint prediction framework. An alternative common practice of deep learning methods is to represent the intrinsic properties in a latent feature space [8] where an image can be *decoded* from implicit feature representations. Directly editing such feature representations, however, is not trivial since they do not correspond to any true physical property. On the contrary, the physics of image formation process is well understood and motivates us to replace this *black-box decoder* with a *physical decoder* and thus enables the network to learn the physics of image formation.

An end-to-end network architecture that replicates the image formation process in a physically based rendering layer is presented. This network is composed of prediction modules that infer each of the intrinsic properties from a single image of an object. These predictions are then provided to a rendering layer which re-synthesizes the input image.

Figure 3.1: The materials of the objects in the input images are replaced with the material properties of the objects in the middle column, resulting edited images are shown in the right column.

The loss function is defined as a weighted sum of the error over the individual predictions and perceptual error over the synthesized images. Comparisons with and without incorporating the perceptual error are provided and show that the combined loss provides significant improvements (see Figure 3.5.Due to the lack of a large scale dataset of real images with ground truth normal, material, and lighting annotations, the network is trained on a rendering based synthetic dataset. Nevertheless, this network performs reasonable predictions for real images where the space of materials and lighting is much more diverse. The results on real images are further refined with a post-optimization process and refined results show that the network predictions provide a good initialization to this highly non-linear optimization. This paves the road to plausible editing results (see Figure 3.1).

3.2 Related Work

Intrinsic Image Decomposition. A closely related problem to image-based material editing is *intrinsic image decomposition* which aims to infer intrinsic properties, e.g. shape, illumination, and material, from a single image and thus enables access to edit them. The work of Barrow et al. [100] is one of the earliest to formulate this problem and since then several variants have been introduced.

Intrinsic image decomposition is an ill-posed problem since different combinations of shape, illumination, and material may result in the same image. Therefore, an important line of work assumes at least one these unknowns to be given, such as geometry. Patow et al. [101] provides a survey of earlier methods proposed to infer illumination, material, or combination of both for scenes with known geometry. More recent work from Lombardi et al. [6, 102] introduces a Bayesian formulation to infer illumination and material properties from a single image captured under natural illumination. They utilize a material representation based on Bidirectional Reflectance Distribution Functions (BRDFs) and thus can handle a wide range of diffuse and specular materials. Approaches that aim to infer all three intrinsic properties [7], on the other hand, often make simplified prior assumptions such as diffuse materials and low-frequency lighting to reduce the complexity of the problem.

An important sub-class of intrinsic image decomposition is *shape from shading (ShS)* where the goal is to reconstruct accurate geometry from shading cues [103]. Many ShS approaches, however, assume prior knowledge about the material properties [104] or coarse geometry [105] to be given.

As opposed to these approaches, my method assumes no prior knowledge about any of the intrinsic properties to be given and handle both diffuse and specular materials under natural illumination. Instead of making assumptions, my method aims to infer priors directly from data via a learning based approach.

Material Editing. Some previous methods treat material editing as an image filtering problem without performing explicit intrinsic image decomposition. Khan et al. [106] utilize

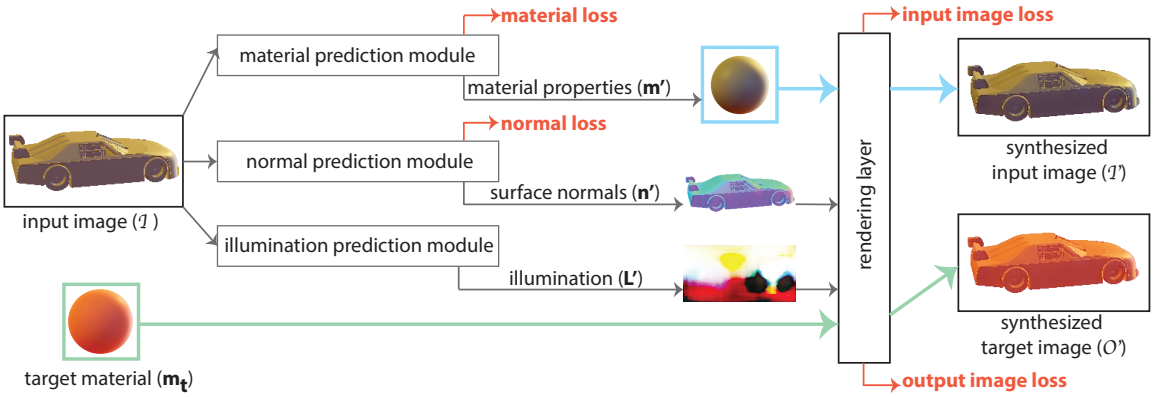


Figure 3.2: Given a single image of an object, I, a network architecture which first predicts material ($\mathbf{m}'$), surface normals ($\mathbf{n}'$), and illumination ($\mathbf{L}'$) is proposed. These predictions are provided to a rendering layer which re-synthesizes the input image ($\mathbf{I}'$). In addition, a desired target material, $\mathbf{m_t}$, is passed to the rendering layer along with $\mathbf{n}'$ and $\mathbf{L}'$ to synthesize a target image $\mathcal{O}'$ which depicts the object with the target material from the same viewpoint and under the same illumination. By defining a joint loss that evaluates both the synthesized images and the individual predictions, this framework perform robust image decomposition and thus enable material editing applications.

simple heuristics to infer approximate shape and illumination from an image and utilize this knowledge to perform material editing. Boyadzhiev et al. [107] introduce several image filters to change properties such as shininess and glossiness. While these approaches achieve photo-realistic results they can provide limited editing scenarios without explicit knowledge of the intrinsic properties.

Material Modeling via Learning. With the recent success of learning based methods, specifically deep learning, several data-driven solutions have been proposed to infer intrinsic properties from an image. Tang et al. [108] introduce *deep lambertian networks* to infer diffuse material properties, a single point light direction, and an orientation map from a single image. They utilize Gaussian Restricted Boltzmann Machines to model surface albedo and orientation. Richter et al. [109] use random forests to extract surface patches from a database to infer the shape of an object with diffuse and uniform albedo. Narihira et al. [110] predict relative lightness of two image patches by training a classifier on features extracted by deep networks. Similarly, Zhou et al. [111] use a convolutional neural

network (CNN) to predict relative material properties of two pixels in an image and then perform a constrained optimization to solve for the albedo of the entire image. Narihira et al. [112] propose a CNN architecture to directly predict albedo and shading from an image. Kulkarni et al. [8] use variational auto-encoders to disentangle viewpoint, illumination, and other intrinsic components (e.g. shape, texture) in a single image. One of the limitation of these methods is the ability to handle diffuse materials only. The recent work of Rematas et al. [113] predict the combination of material and illumination from a single image handling both diffuse and specular materials. In a follow-up work [114], they propose two independent network architectures to further disentangle material and illumination from such a combined representation. However, any error that occurs when inferring the combined representation is automatically propagated to the second step. In contrast, in this chapter, an end-to-end network architecture is proposed to perform this disentangling in one pass. Last but not least, several recent work [115, 116] uses neural networks to estimate per-pixel intrinsic properties from a given image. However, these estimations are aligned with the input image, thus, it is not easy transfer these properties across images of different objects as my method does.

3.3 Approach

Overview. An end-to-end network architecture for image-based material editing is proposed. The input is $\mathcal{I}$, a single image of an object s with material $\mathbf{m}$ captured under illumination $\mathbf{L}$. The object is assumed to be masked out in the image. Given a desired target material definition $\mathbf{m_t}$, the goal is to synthesize an output image $\mathcal{O}$ that depicts s from the same viewpoint with material $\mathbf{m_t}$ and illuminated by $\mathbf{L}$.

While inferring illumination, material, and shape from a single image is an ill-posed problem, once these parameters are known, the forward process of image synthesis, i.e. *rendering* is well defined. I propose a network architecture that encapsulates both this inverse decomposition and the forward rendering processes as shown in Figure 5.1. The

first part of the network aims to infer illumination ($\mathbf{L}'$), material ($\mathbf{m}'$), and 3D shape (represented as surface normals, $\mathbf{n}'$) from $\mathcal{I}$ via three different prediction modules. The output of each of these modules is then provided to a *rendering layer* to synthesize $\mathcal{I}'$. In addition, the target material $\mathbf{m_t}$ is passed to the rendering layer along with predicted $\mathbf{L}'$ and $\mathbf{n}'$ to synthesize an output image O'. A joint loss function is defined to evaluate both the outputs of the rendering layer and the predictions of two of the individual modules. It will show that this joint loss significantly improves the accuracy of the inverse image decomposition process and achieves compelling material editing results.

Sec. 3.3.1 introduces the render layer; Sec. 3.3.2 discusses prediction modules providing data to it; and the training procedure is described in Sec. 3.3.3. Then, Sec. 3.3.4 shows how objects composed of multiple materials are handled.

3.3.1 Rendering Layer

The color of each pixel in an image depends on how the corresponding object surface point reflects and emits incoming light along the viewing direction. The rendering layer aims to replicate this process as accurately as possible. Compared to previously proposed differentiable renderers [117], the main advantage of the rendering layer is to model surface material properties based on BRDFs to handle both diffuse and specular materials. Environment maps is also utilized, which provide great flexibility in representing a wide range of illumination conditions. Note that the rendering layer also makes several moderate assumptions. It assumes the image is formed under translation-invariant natural illumination (i.e. incoming light depends only on the direction). It also assumes there is no emission and omit complex light interactions like inter-reflections and subsurface scattering.

Under these conditions, the image formation process is modeled mathematically similar to Lombardi et al [6]. Given per-pixel surface normals $\mathbf{n}$ (in camera coordinates), material properties $\mathbf{m}$ and illumination $\mathbf{L}$, the outgoing light intensity for each pixel p in image $\mathcal{I}$

can be written as an integration over all incoming light directions ω_i:

$$\mathcal{I}_p(\mathbf{n_p}, \mathbf{m}, \mathbf{L}) = \int f(\omega_i, \omega_o, \mathbf{m})\mathbf{L}(\omega_i) \max(0, \mathbf{n_p} \cdot \omega_i) d\omega_i, \tag{3.1}$$

where $\mathbf{L}(\omega_i)$ defines the intensity of the incoming light and $f(\omega_i, \omega_o, \mathbf{m})$ defines how this light is reflected along the outgoing light direction ω_o based on the material properties $\mathbf{m}$. In order to make this formulation differentiable, the integral is substituted with a sum over a discrete set of incoming light directions defined by the illumination $\mathbf{L}$:

$$\mathcal{I}_p(\mathbf{n_p}, \mathbf{m}, \mathbf{L}) = \sum_{\mathbf{L}} f(\omega_i, \omega_o, \mathbf{m})\mathbf{L}(\omega_i) \max(0, \mathbf{n_p} \cdot \omega_i) d\omega_i. \tag{3.2}$$

The rest of this section discusses how each property is represented.

Surface normals (n). $\mathbf{n}$ is represented by a 3-channel image, same size as the input image, where each pixel p encodes the corresponding per-pixel normal $\mathbf{n_p}$.

Illumination (L). Illumination is represented with an HDR environment map of dimension 64×128. Each pixel coordinate in this image can be mapped to spherical coordinates and thus corresponds to an incoming light direction ω_i in Equation 3.2. The pixel value stores the intensity of the light coming from this direction.

Material (m). $f(\omega_i, \omega_o, \mathbf{m})$ is defined based on BRDFs [118] which provide a physically correct description of pointwise light reflection both for diffuse and specular surfaces. Non-parametric models [119] aim to capture the full spectrum of BRDFs via lookup tables that encode the ratio of the reflected radiance to the incident radiance given incoming and outgoing light directions (ω_i, ω_o). Although such lookup tables achieve highly realistic results, they are computationally expensive to store and not differentiable. Among the various parametric representations, Directional Statistics BRDF (DSBRDF) model [120] is adopted, which is shown to accurately model a wide variety of measured BRDFs. This

model represents each BRDF as a combination of hemispherical exponential power distributions and the number of parameters depends on the number of distributions utilized. The experiments show that utilizing 3 distributions provides accurate approximations resulting in 108 parameters per material definition in total. The original work [120] provides more details.

3.3.2 Prediction Modules

Three prediction modules are utilized to infer surface normals, material properties, and illumination. The input to each module is the 256×256 input image $\mathcal{I}$.

Normal prediction. The normal prediction module follows the same spirit as the recent work of Eigen et al. [9]. The main difference is that the normal map is predicted to be equal in size to the input image by utilizing a 4-scale-submodule network as opposed to the originally proposed 3-scale-submodule network. The fourth submodule consists of 2 convolutional layers where both input and output size is equal to the size of the input image. A normalization layer is utilized to predict surface normals with unit length.

Illumination prediction. Illumination prediction module is composed of 7 convolutional layers where the output of each such layer is half the size of its input. The convolutional layers are followed by 2 fully connected layers and a sequence of deconvolutional layers to generate an environment map of size 64×128. Each convolutional and fully connected layer is accompanied by a rectifier linear unit. Fully connected layers are also followed by dropout layers.

Material prediction. Material prediction module is composed of 7 convolutional layers where the output of each such layer is half the size of its input. The convolutional layers are followed by 3 fully connected layers and a tanh layer. Each convolutional and fully connected layer is accompanied by a rectifier linear unit. Fully connected layers are also followed by dropout layers. Note that since each of the 108 material parameters are defined at different scales, each is normalized to the range $[-0.95, 0.95]$ and remap to their original scales after prediction.

3.3.3 Training.

The proposed network architecture is trained by defining a joint loss function that evaluates both the predictions of the individual modules and the images synthesized by the rendering layer. Specifically, a L2 normal loss is defined as:

$$l_{normal} = \sum_{p} (\mathbf{n_p} - \mathbf{n'_p})^2, \tag{3.3}$$

where $\mathbf{n_p}$ and $\mathbf{n'_p}$ denote the ground-truth and predicted surface normals for each pixel p respectively. Since both ground-truth and predicted normals are unit length, this loss is equivalent to cosine similarity used by Eigen et al. [9].

The material loss is defined as the L2 norm between the ground truth ($\mathbf{m}$) and predicted material parameters ($\mathbf{m'}$):

$$l_{material} = (\mathbf{m} - \mathbf{m'})^2 \tag{3.4}$$

A perceptual loss, $l_{perceptual}$, is also utilized to evaluate the difference between the synthesized $\{\mathcal{I}', \mathcal{O}'\}$ and ground truth $\{\mathcal{I}, \mathcal{O}\}$ images. The pre-trained *vgg16* network is used to measure $l_{perceptual}$ as proposed by Johnson et al. [121].

The final loss, l, is defined as a weighted combination of these individual loss functions:

$$l = w_n l_{normal} + w_m l_{material} + w_p l_{perceptual}, \tag{3.5}$$

where the weights are empirically set to be w_n=1 $\times$ 10^4, w_m=1 $\times$ 10^3, w_p=1.

3.3.4 Extension to Multi-Material Case

A typical use case scenario for image-based material editing is where a user denotes the region of interest of an object and a desired target material that should be applied to this region. My approach trivially supports such use cases. Given a segmentation mask

Figure 3.3: For each input image, an output image is synthesized with a given target material definition. The ground truth (GT) target images are provided for reference. The left (middle) column shows cases where the target material is more (less) specular than the input material. Examples where both the input and the target material are specular are also provided in the right column.

denoting regions of uniform material, material prediction is performed for each region while predicting a single normal and illumination map for the entire image. All these predictions are then provided to the rendering layer. Target materials are defined for each region of interest separately. The rest of the training and testing process remains unchanged.

Such multi-material dataset will also be synthesized for training as well. Specifically, the 3D segmentation methods described in Chapter 2 will be used to decompose the 3D models into several parts. Then, if necessary, some of the parts will be manually merged to create more semantic segmented parts. During the training, a material will be randomly assigned to each part independently and then use a single environment map image to render an image. the normal and lighting will be estimated from the whole image; but estimate the material property from each part individually.

3.3.5 Refinement with Post-optimization

Due to the lack of a large scale dataset of real images with ground truth normal, material, and illumination annotations, the network is trained with synthetic renderings. Although large 3D shape repositories [122] provide sufficient shape variation, the network does not see a large spectrum of material and illumination properties captured in real images. While increasing the variation in the synthetic data could reduce the discrepancy between the

training and test sets, the gap is almost always there theoretically. Thus, the network predictions on real images are refined with a post-optimization. Specifically, using n^o, m^o, L^o, the network predictions of surface normal, material, and illumination for an input image I as initialization, n^*, m^*, L^* are optimized to minimize the following energy function:

$$\underset{\mathbf{n^*},\mathbf{m^*},\mathbf{L^*}}{\operatorname{argmin}} \parallel \mathbf{I^*} - \mathbf{I} \parallel^2 + a \parallel \mathbf{n^*} - \mathbf{n^o} \parallel^2 + b \parallel \mathbf{L^*} - \mathbf{L^o} \parallel^2 . \tag{3.6}$$

I^* is the image formed by n^*, m^*, L^* and the first term penalizes the image reconstruction loss. The remaining regularization terms penalize the difference between the network predictions of normal and illumination and their optimized values. The relative weights are set to be $a = 1$, $b = 10$ for those regularization terms. L-BFGS is used to solve Equation 3.6 in an alternative scheme where only one intrinsic property is updated at each iteration.

3.4 Evaluation

This section provides quantitative and qualitative evaluations of my method as well as comparisons to previous work.

3.4.1 Datasets and Training

The framework is trained with a large amount of synthetic data generated for *car, chair, sofa*, and *monitor* categories. They include a total of 280 3D models (130 cars, 50 chairs, 50 sofas and 50 monitors) obtained from ShapeNet [122]. For materials, BRDF measurements corresponding to 80 different materials provided in the MERL database [119] are used. For illumination, 10 free HDR environment maps were downloaded from the Internet and random rotation is used to augment these environment maps.

Data generation. For each 3D model, material, and illumination combination 5 random views are sampled from a fixed set of pre-sampled 24 viewpoints around the object with

fixed elevation. The data is split such that no shape and material is shared between training and test sets. Specifically, 80 cars are used for pretraining, and 40 shapes per each category for joint category finetuning, which leaves 10 shapes per category for testing. Out of the 80 materials, 20 are used for testing and the rest are used in pre-training and training. This split allows us to generate $240K$ pre-training instances, and $480K$ multi-category finetuning instances. In total, the network is trained with over $720K$ unique material-shape-light configurations.

Training procedure. The weights are initialized from a uniform distribution: $[-0.08, 0.08]$. Normal and material modules are pre-trained using L2 loss for a few iterations and then trained jointly with illumination network. Adam [123] optimizer is utilized for stochastic gradient descent. Similarly, momentum parameters are first set to be $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and a learning rate to be 0.0001. Later the learning rate is reduced to be 1×10^{-6} and 1×10^{-8}. β_1 is also reduced to 0.5 for a more stable training.

3.4.2 Evaluation on Synthetic Data

To test my network, an image corresponding to shape s, environment map $\mathbf{L}$, and material $\mathbf{m}$ from the test set is randomly selected as input. Given a target material $\mathbf{m_t}$, an output image depicting s with material $\mathbf{m_t}$ and illuminated by $\mathbf{L}$ is synthesized. This synthesized output is compared to ground truth using two metrics. While the L2 metric measures the average pixel-wise error, SSIM metric[124] measures the structural difference. The L2 error is computed on tone mapped images where each color channel is in the range $[0, 255]$ and define per-pixel error as the average of the squared difference of three color channels. Note that lower (higher) numbers are better for the L2 (SSIM) metric. Quantitative results are provided in Table 3.2, last two columns. Note that the network is trained for all categories jointly. Figure 3.3 provides visual results demonstrating that my approach can successfully synthesize specularities when replacing the input material with a more specular target material. Similarly, specularities in the input image are successfully removed when

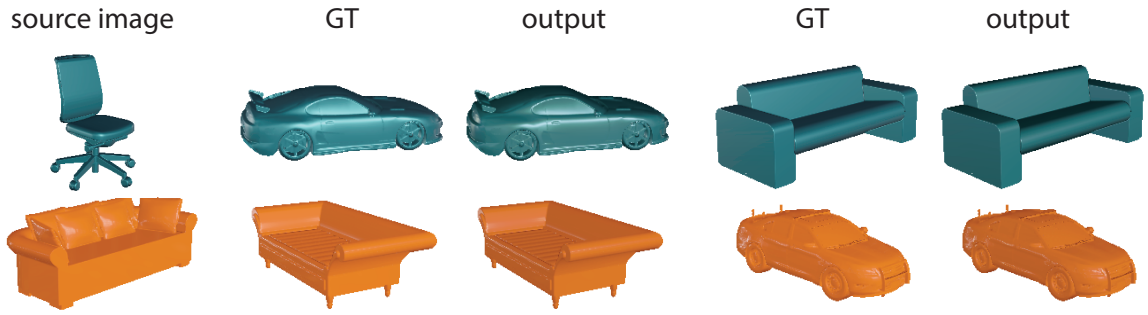


Figure 3.4: An image of an object is synthesized under some desired illumination with material properties predicted from a source image. The ground truth (GT) is provided for reference.

replacing the input material with a more diffuse one.

Figure 3.4 shows examples where material properties predicted from a source image are transferred to a different object under some desired illumination. Such material transfers alleviates the need to define target materials explicitly, instead each image becomes an exemplar.

3.4.3 Evaluation of the Rendering Layer

The effectiveness of utilizing the perceptual loss on the images synthesized by the rendering layer is evaluated as opposed to independent predictions.

Accuracy of Material Coefficients. For this evaluation, given an input image $\mathcal{I}$, ground truth normals ($\mathbf{n}$) and illumination ($\mathbf{L}$) are provided and the network only predicts material. The material prediction module is also trained as a standalone network. While both the standalone module and my network use the L2 loss on material coefficients in spirit similar to the recent work of Georgoulis et al. [114], the latter combines this with the perceptual loss on $\mathcal{I}'$. Qualitative and quantitative results are provided in Figure 3.5 and Table 3.1 respectively. The results demonstrate that the L2 loss is not sufficient in capturing the physical material properties, a small L2 error may result in big visual discrepancies in the rendering output. Incorporation of the rendering layer resolves this problem by treating

Figure 3.5: Each example shows the ground truth image as well as the renderings obtained by utilizing the material coefficients predicted by the standalone material prediction module and the combined approach which also uses the rendering layer.

the material coefficients in accordance with their true physical meaning.

	material	**rendering**	
	L2	L2	SSIM
standalone	4.1038	544.1	0.9297
combined	4.8144	355.5	0.9517

Table 3.1: The accuracy of the predicted material coefficients and the images rendered using these together with ground truth normals and illumination are shown. Results of the material prediction module trained standalone vs with the rendering layer are provided.

3.4.4 Multi-Material Examples

Examples for objects composed of multiple materials are also provided in this section. For these examples, the 3D segmentation method described in Chapter 2 are used to decompose the 3D models into several parts; then some of the parts are manually merged to become more semantic regions where each region gets its own material assignment. Renderings from 5 different viewpoints are generated with random illumination and materials assignments. The model trained on single-material examples is fine-tuned with these additional renderings. 80 chairs are used for fine-tuning and 20 for testing. Figure 3.7 shows examples of

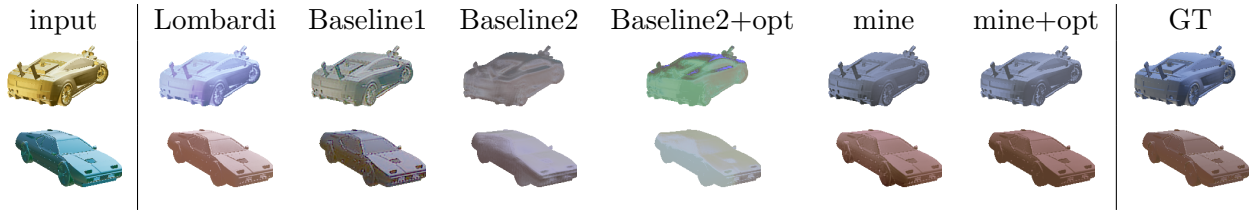


Figure 3.6: For each example, the input and the ground truth target image (GT) are shown. The results obtained by the method of Lombardi et al. [6], two baseline methods, and my approach are provided.

editing each region with a target material. These results are also quantitatively evaluated with respect to ground truth over 4000 examples. It achieves an L2 error of 1272.1 and SSIM index of 0.9362.

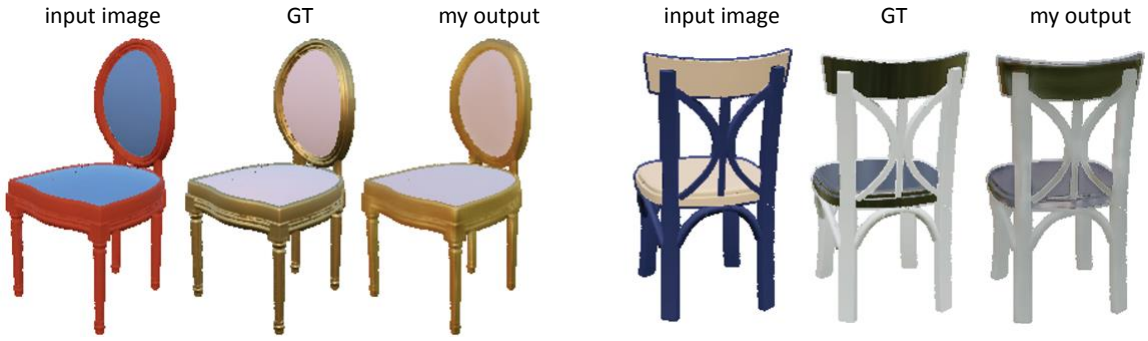


Figure 3.7: Given an image of a multi-material object, this figure shows how different target material assignments are realized for different parts. Ground target (GT) images are provided for reference.

3.4.5 Comparisons

My method is compared with several baseline methods and previous approaches which are briefly discussed below.

Figure 3.8: The target materials are transferred to the input image using the method of Lombardi et al. [6], baseline 2, baseline 2 with post-optimization, my network, and my network with post-optimization.

Lombardi et al. [6] My method is compared to the approach of Lombardi et al. [6] which uses certain priors to optimize for illumination and material properties in a single image of an object with known geometry. The ground truth surface normals are provided to this method and the material and illumination coefficients are from predictions. The target image is rendered given the target material, ground truth normals, and the predicted illumination.

Baseline 1. The first baseline is inspired by the work of Kulkarni et al. [8] and is based on an encoder-decoder network. The encoder maps the input image to a feature representation, $f = (f_m, f_o)$, with one part corresponding to the material properties (f_m) and the remaining part corresponding to other factors (f_o) (i.e. normals and illumination). This disentangling is ensured by passing f_m as input to a regression network which is trained to predict the input material coefficients. f_m is then replaced by the feature representation of the target material. The decoder utilizes the updated feature representation to synthesize the target image. Skip connections are also provided from the input image to the decoder to avoid blurry output. Similar to my approach, *baseline 1* is trained with the perceptual loss between the synthesized and ground truth target image and the L2 loss on the estimated input material coefficients.

Baseline 2. For the second baseline, given the input image,three individual networks

	Lombardi	Baseline1	Baseline2		mine	
			no opt	opt	no opt	opt
L2	802.7	1076.7	886.9	1273.7	805.2	530.9
SSIM	0.9416	0.9173	0.9256	0.9159	0.9408	0.9557

Table 3.2: The accuracy of material transfer results are evaluated on synthetic data for Lombardi et al. [6], baseline 1, baseline 2 with and without post-optimization, and my method with and without post-optimization.

are trained to predict the surface normals, material, and illumination separately. All networks are trained with the L2 loss. The target images are then rendered using the predicted surface normals, illumination, and the target material.

For this comparison, my method and both of the baselines are trained on the same training and testing set. The output of baseline 2 and my method are refined with the post-optimization described in Section 3.3.5. 500 images sampled from the testing set are used as input and generate a new image with a target material from the test set. Quantitative results are provided in Table 3.2 and visual results in Figure 3.6. For baseline 2, the use of L2 loss only results in blurry normal prediction, hence the blurry target images. While baseline 1 addresses this issue by use of skip connections, implicit feature representations fail to capture the true intrinsic properties and result in a large appearance difference with respect to ground truth. The approach of Lombardi et al. [6] performs non-convex optimization over the rendering equation which is likely to get stuck in local minimum without a good initialization. The individual predictions from baseline 2 are quite far from the global minimum or any reasonable local minimum. Thus, no improvement is observed with post-optimization, in some cases the optimization in fact gets stuck in a worse local minimum. My initial network output performs in par with [6] with no assumption on known surface normals. Refining these predictions with the post-optimization outperforms all other methods significantly.

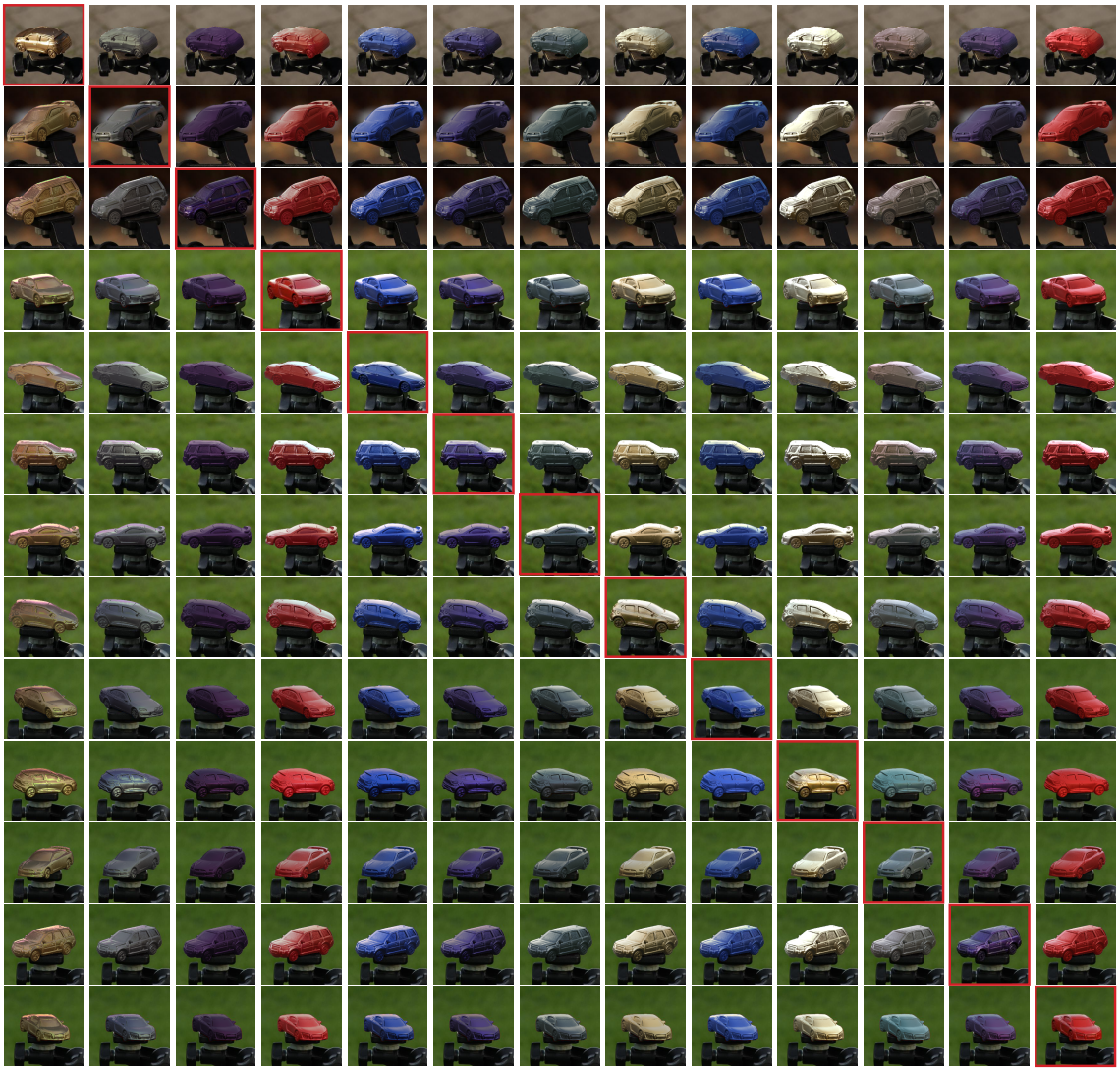


Figure 3.9: Given a set of images (in diagonal, in red boxes), new images are synthesized by using shape and light from its row and material from its column using my approach.

3.4.6 Evaluation on Real Images

Visual comparisons of my method with the method of Lombardi et al. [6] and baseline 2 on real product images[1] downloaded from the internet in Figure 3.8 are provided. Since the

[1]For real examples I finetune my network using all materials in the material dataset to better generalize to unseen materials.

method of Lombardi et al. [6] assumes surface normals to be known, the normal predictions of my method are provided to their optimization. Additionally, the results are fined for both baseline 2 (i.e. individual prediction of normal, material, and illumination) and my method with the post-optimization introduced in Section 3.3.5. Despite the fact that my network has been trained only on synthetic data with a limited set of environment maps and materials, the raw network results are promising and provide a good initialization to the post-optimization. Independent predictions, on the other hand, result in the post-optimization to get stuck in a local minimum which does not yield as visually plausible results.

Finally, in Figure 3.9, material transfer examples are provided on some real images provided by the SMASHINg challenge dataset [113] with the combination of my network and the post optimization.

3.5 Conclusion and Future Work

I propose an end-to-end network architecture for image-based material editing that encapsulates the image formation process in a rendering layer. The rendering layer enables the network to synthesize images with significant improvements in appearance. I demonstrate various plausible material editing results both for synthetic and real data. This work is also published in [5].

One limitation is that the MERL BRDF dataset [119] only contains 100 materials. Even though these materials can be combined to interpolate more materials, this interpolated space sometimes can't cover some of the real world's materials. For some real images, the material estimations would be quite different from the real underlying materials. Some failure examples can be found in Figure 3.10.

Another limitation of this work is the fact that lighting space is learned with a relatively small dataset, because of the lack of data. This may result in imperfect decomposition, specifically some lighting effects being predicted to be part of the surface normals.

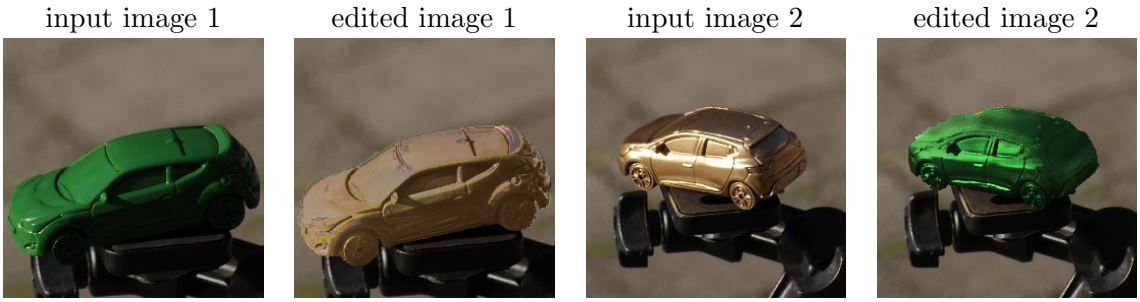


Figure 3.10: Failure cases: given two input images, new images are synthesized by swapping the material properties of the objects.

This is especially apparent in the relatively poor performance on real examples (see the supplementary material).

For multi-material cases, even though the normal and lighting can be estimated from the whole image globally and assign a desired target material for a either big or small region, directly estimating material for a small segment independently is difficult. For such scenarios, extending the network to perform per-pixel material predictions will be a promising direction. Incorporation of advanced light interactions such as subsurface scattering and inter-reflections in the rendering layer is crucial to generalize the method to scenes with multiple objects and real images. Finally, I expect the performance boost obtained by the encapsulation of the rendering layer to stimulate further research in designing neural networks that can replicate physical processes.

Chapter 4: Shape Synthesis - Fine Scale Normal Prediction

In this Chapter, I will focus on estimating fine-scale normal from 2D images. I will work on both synthetic dataset and real image dataset. The framework is similar to the framework used in the Chapter 3. Especially, I will show how this framework will be used for extracting detailed images from real scene images where the ground truth data is noisy. Comparisons will be shown to validate the effectiveness of this framework.

4.1 Introduction

As one type of shape properties, normal direction captures the local surface details. It has wide applications in image editing, visual understanding, geometry reconstruction etc. Image relighting is a representative task where the normal direction is useful. Naive image relighting may add or reduce the light intensity for each pixel equally; while knowing the normal direction helps to make each pixel's light changes to be more consistent with the pixel's geometry variation. David Marr [125] proposed that visual recognition takes 2.5D representation as the intermediate representation, where human recognition system tend to first infer 2.5D structure for the visible pixels from 2D image and then generates the 3D shape using the 2.5D representation. Surface normal is one of those important 2.5D representations. Normal prediction, which captures the local surface details, can also be used to generate denser and smoother geometry reconstruction.

There are mainly three sources to obtain the normal direction information:

1). One way to obtain normal is to use depth sensors to get the 3D point cloud data first and then normal direction is computed from the the local frame. However, the problems of this source are: a). **Inadequate calibration and inaccurate measurement of disparities.** Inadequate calibration is the source of systematic error in points' coordinates.

Inaccurate measurement of disparities also influence the accuracy [126]; b). **Measurement setup.** This is related to the lighting condition and the imaging geometry. With strong light, the laser speckles appear in low contrast in the infrared image leading to outliers or gap in the captured point clouds. The imaging geometry includes object's distance and surface orientation relative to the sensor. For instance, the operating range of a certain may be between 0.5m to 5m and the error increase when out of this range. Some part of the sense may not be seen by the infrared camera, but may have been by the laser pattern [126]; c). **Smooth and specular materials.** For the surfaces with these materials, sensors can't get accurate feedback to measure the depth. As a result, the computed normals usually are noisy or contains some missing regions, shown as the second image in Figure 1.3. On the other hand, normals computed from these depth data usually serve as the ground truth training data for some learning-based approaches [9, 127, 128]. In consequence, it is possible that inaccurate ground truth confuses the machine learning models and leads to model's worse performance during the deployment stage.

2). The second source is multi-view image based reconstruction [129]. However, it usually can only reconstruct sparse points and for most scenarios, there are no multiple views images of the same object or scene, like the images on the Internet.

3). The third source is the single image based normal estimation. Single image based normal estimation usually depends on model fitting or priors which usually are learning through learning approaches. Model fitting [130] requires a large dataset to cover enough geometry variety. Occlusion and camera pose are also big challenges for model fitting, which directly influence final fitting performance. Occlusion causes a lot of ambiguities; while camera pose estimation itself is an unsolved open problem. Moreover, model fitting approaches are not easy to generalize to scene. Some of the prior based approaches [131] usually over-simplify the problems with a lot of assumptions, which limit their application scenarios. Other priors based approaches [132] have few assumptions and constraints, but usually produce blurry normal predictions.

In this chapter, I will propose a framework to predict fine-scale normals from a single

image. The novelties contain two parts: 1). this framework will predict significantly more detailed normals than previous learning approaches with few assumptions or even without any assumption about the input image; 2). one mechanism is proposed to reduce the effects of inaccurate ground truth normal training data. Previous learning based approaches only defines the L2 loss function (equivalent to the angular loss) between the ground truth normal $\mathbf{n}_\mathbf{p}^*$ and predicted normal $\mathbf{n}_\mathbf{p}'$.

$$l_{normal} = \sum_p (\mathbf{n}_\mathbf{p}^* - \mathbf{n}_\mathbf{p}')^2, \tag{4.1}$$

where the problems include: 1). this type of loss function usually leads to blurry output; 2). as it uses the inaccurate ground normal directly, the inaccuracy may hurt the performance of the model.

The main motivation of my approach is that: during the training, it not only penalizes the difference between ground truth normal and predicted normal but also adds the regularization of whether the image generated from estimated normal matches with the image generated from the real normal. The image generated from real normal represents the original input image. Besides normal, the material and lighting will also be predicted from the image. It is equivalent to that my approach predict normals that is both close to the ground truth and generates the image close to the original input image, together with the predicted material and lighting. Particularly, the new formulation can be regarded as the following:

$$\sum_p P(\mathbf{I}_\mathbf{p}^*, f(\mathbf{n}_\mathbf{p}', \mathbf{m}_\mathbf{p}', \mathbf{l}_\mathbf{p}')) + \lambda \sum_p (\mathbf{n}_\mathbf{p}^* - \mathbf{n}_\mathbf{p}')^2, \tag{4.2}$$

where $\mathbf{I}_\mathbf{p}^*$ denotes the input image; $\mathbf{n}_\mathbf{p}'$, $\mathbf{m}_\mathbf{p}'$ and $\mathbf{l}_\mathbf{p}'$ are the predicted normal, material and lighting respectively; f is the rendering function; $\mathbf{n}_\mathbf{p}^*$ is the ground normal. P is perceptual loss function defined in [121]. The rendering equation is same with that in Chapter 3.

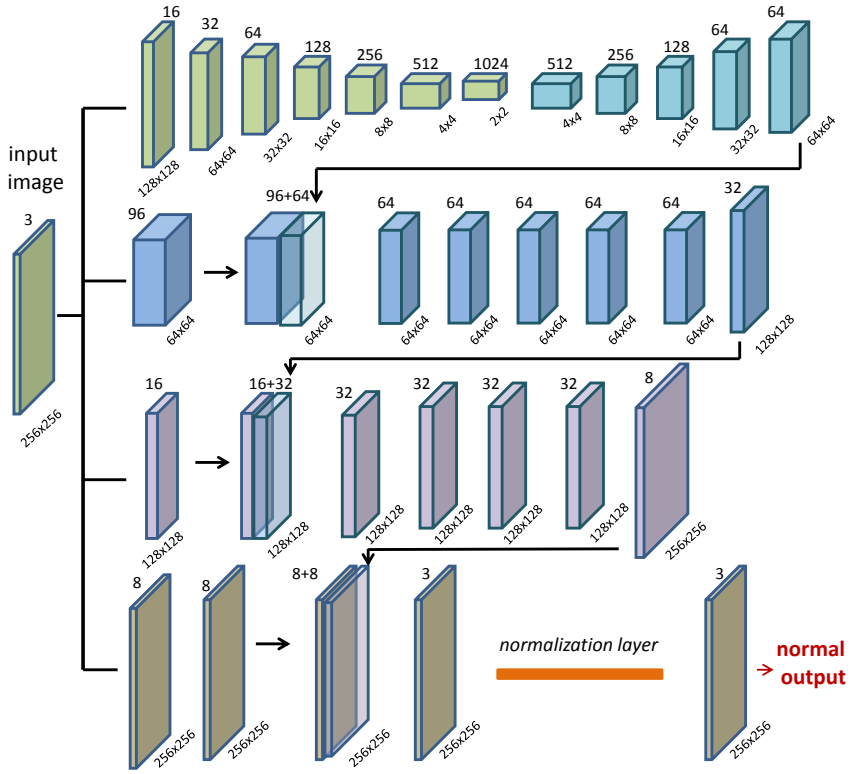


Figure 4.1: Multi-Branch Normal Prediction Architecture.

One main advantage of this formulation is that it can transfer the high resolution characteristics of image to normal prediction. Images serve as a better source to provide details than the ground truth normal computed from depth data acquired by depth sensors. Also there is a chance that it would improve the training for the region where ground truth normal is noisy or missing.

To validate the effectiveness of the formulation, I train and test the model on both single object images and scene images. For single image, both the cases of single material and multi-material are covered. For scene images, the model is trained and tested on RGB-d dataset.

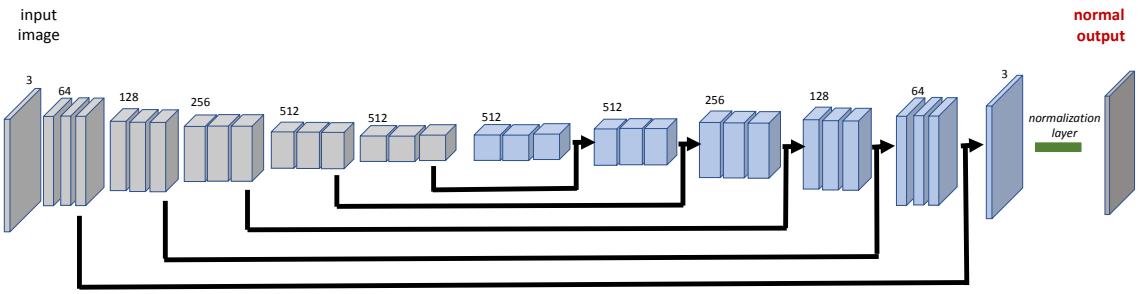


Figure 4.2: "U"-net Normal Prediction Architecture.

4.2 Single Object Normal Estimation

Deep neural network is chosen as the model and loss function is defined using Formulation 4.2. For the deep neural network architecture, two architectures are considered. The first architecture is a multi-branch network, similar to the work of Eigen et al. [9]. This type of network contains several sub-networks with smaller sizes to make the network have higher capability to generate detailed final output. The structure of this architecture can be found in Figure 4.1. Assuming the input image is with size 256×256, the first branch generates 64×64; the second branch use a single convolution layer to convolute the input image to 64×64 and concatenate it with the result produced by first branch, then it will generate the 128×128 output; similar to the second branch, the third branch will give the output with size 256×256; the fourth branch does all the convolution operations on size with 256×256 directly. The second architecture is similar to the work of Zhang et al. [133]. This architecture contains encoder part and decoder part. The encoder part is adapted from VGG-16 network [23] by removing the fully connected layers, same to the encoder of Zhang et al. [133]. The decoder is symmetric to the encoder with convolutional layers and upsampling layers (or unpooling layers). Figure 4.2 shows the details of this architecture. From the experiments, I find that two architectures perform similarly. For single object

Figure 4.3: Some examples of environment map used for training.

case, I mainly use the multi-branch network for single object normal prediction.

4.2.1 Single Material Case

For single object case, synthetic data are mainly used. The advantage of using synthetic data is that there are ground truth normal, material and light. Unlike the normal computed from depth data capture by depth sensors, the ground truth normal is accurate and detailed. Same to Chapter 3, the network is trained on *car, chair, sofa* and *monitor* categories, with a total of 280 3D models (130 cars, 50 chairs, 50 sofas and 50 monitors). 80 materials from MERL dataset [119] are used. 14 free HDR environment maps downloaded from the Internet are used as illumination. Some examples of environment maps are shown in Figure 4.3. The details of data generation and training process are discussed in Section 3.4 of Chapter 3.

Figure 4.4 shows the results of testing on the synthetic images of single object with single material. Figure 4.5 shows the results of testing on the real images using the model trained on synthetic images. "standalone" represents the results using the Formulation 4.1 as the loss function; "combined" represents the results using Formulation 4.2 as the loss

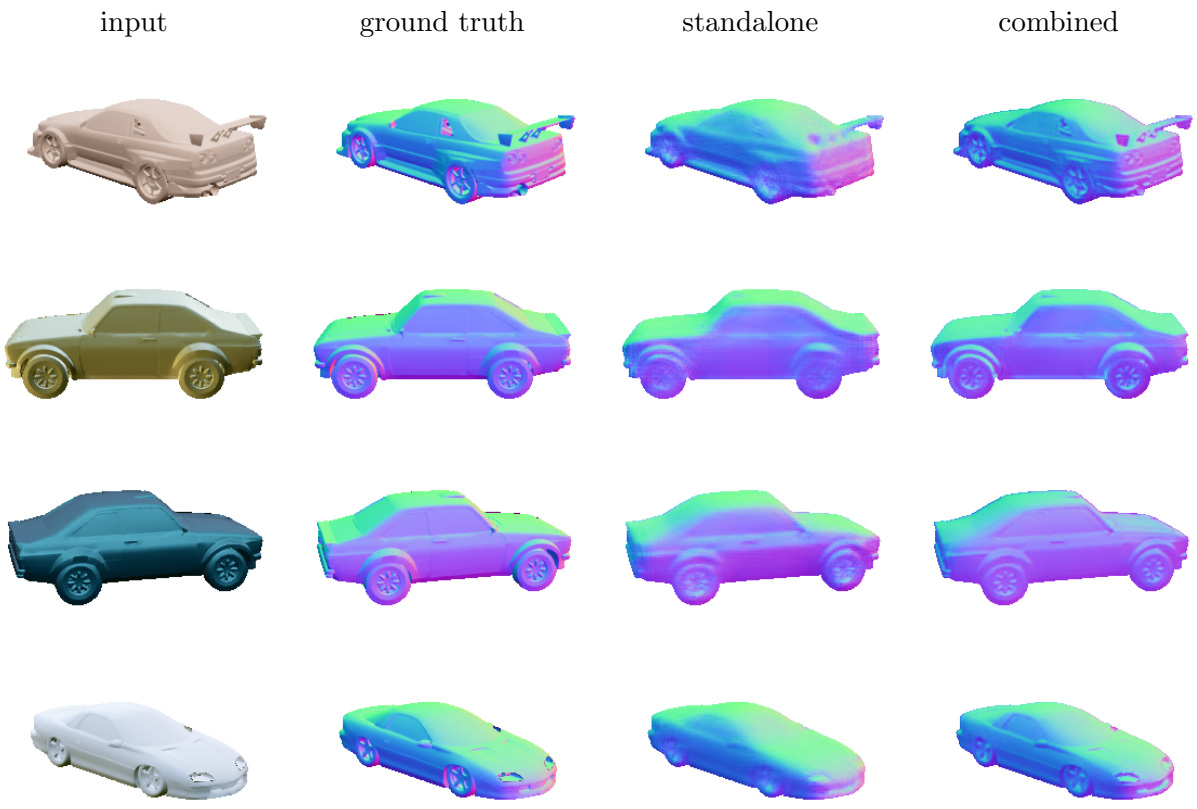


Figure 4.4: Comparison of standalone and combined formulation on synthetic dataset.

functions. It can be seen that the model trained using Formulation 4.2 as the loss function can produce more details normal prediction.

4.2.2 Multiple Material Case

I use the same setting as the Single Material Case as in Section 4.2.1. The only difference is that the shape will be segmented into several parts. The segmentation is obtained: 1). using the connectivity information to do graph search and. if possible, getting the original component information coming along with the model itself; 2). using the segmentation methods described in Chapter 2 to further decompose the model into several parts; 3). manually

Figure 4.5: Comparison of standalone and combined formulation on real images.

Figure 4.6: Comparison of standalone and combined formulation on single object with multiple materials.

merging the decomposed parts into more meaningful and semantic components. The normals are generated using the views same to the views in Section 3.4 of Chapter 3. I trained two models both using the network in Figure 4.1: one using standalone Formulation 4.1 as the loss function; the other one using combined Formulation 4.2 as the loss function. Two models are trained using the same dataset with same batch numbers. Figure 4.6 shows the comparison of the models trained using those two formulations as loss functions. It can be seen that the model trained using combined Formulation 4.2 as the loss function can also generate more detailed normal predictions than the one using standalone Formulation 4.1 as the loss function.

4.3 Scene Image Normal Estimation

Scene images contain a lot of textures and various objects with different material properties. Thus, a single scene image usually contains several material properties, especially when there a lot of various texture patterns in the image. As a result, predicting a single material from the whole image is not enough. Per-pixel material estimation is required. One challenge about the per-pixel material estimation using current material representation is that each pixel needs to predict 108 dimensional vector. As a result, if the input image is with size $256 \times 256 \times 3$, where height and width area 256 and channel size is 3, the network needs to produce an output with size $256 \times 256 \times 108$. This makes the network structure to be very complex and increases the parameter number by a large amount. Both the model size, computation time for forward pass and computation for backward pass will increase a lot. It would make the network very difficult to train. To reduce the complexity, following the work of Lombardi et al [6], the *functional* PCA (functional principal component analysis) is used as the representation. This *functional*-PCA allows to use fewer number of material representation. The basis of functional principal component analysis is computed from the 100 materials from MERL BRDF [119]. In this way, the material is represented using 32

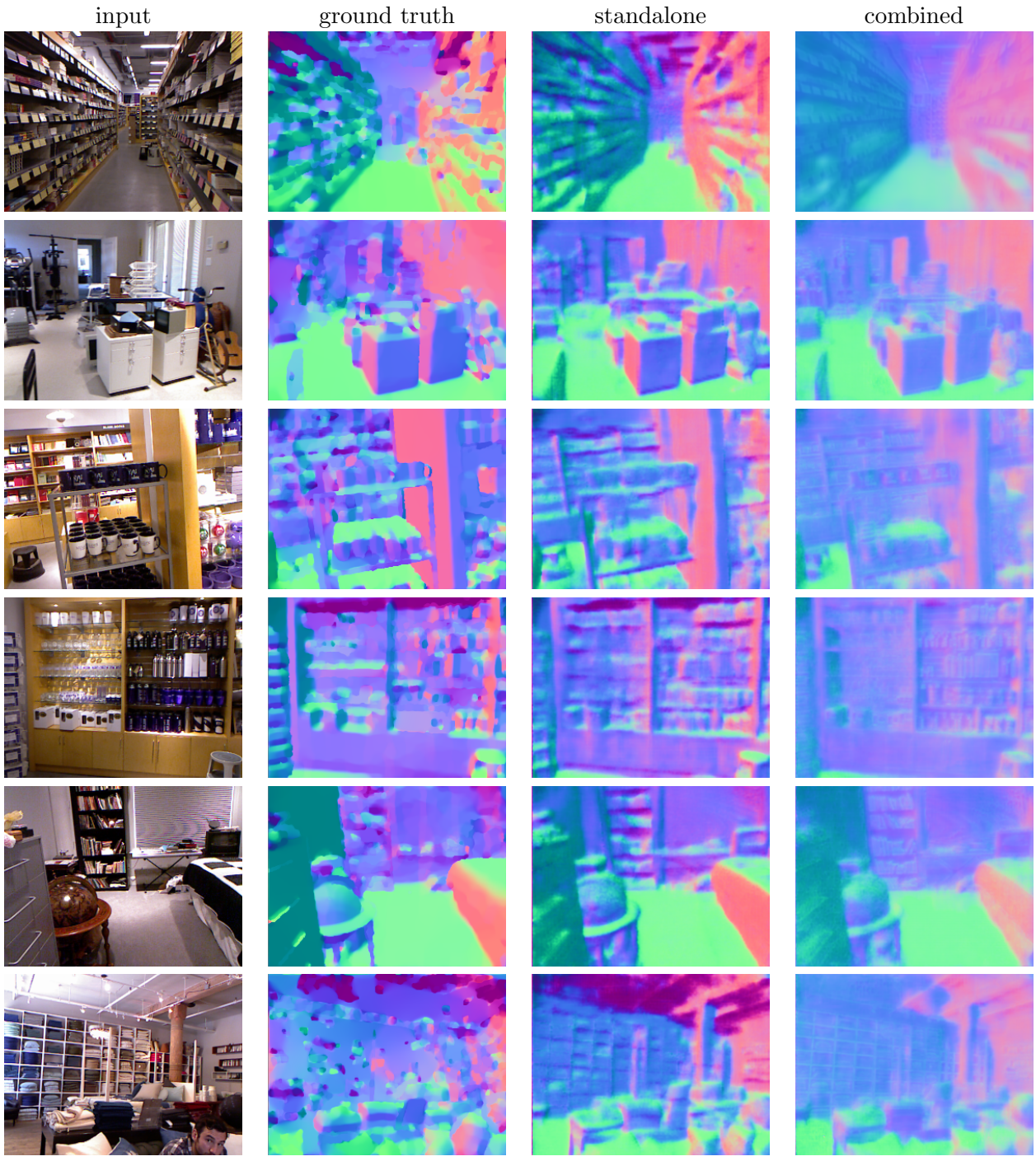


Figure 4.7: Comparisons of standalone and combined formulation on real scene images.

dimensions with the help of *functional*-PCA. The network design is the same with the "U"-shape structure, except that the output channel size 32. To train this network, I use the raw video of RGB-d data from [10] and use the method described in [127] to compute the ground truth normal. For the normal predictions on scene images, there are some existing works on this topic, such as the work of [133]. The network structure in [133] is the same with Figure 4.2. The same networks are used for the comparisons.

Figure 4.7 shows the comparisons between standalone network and combined formulation. The first column shows the input image; the second column shows the ground truth compared from the depth data, which is noisy; the third column shows the results from standalone formulation; the fourth column shows the results from combined formulation. It can be seen that the model trained using combined formulation produces much more detailed output than the ground truth and the model trained using standalone formulation.

4.4 Quantitative Evaluation

Accuracy of Surface Normals. For this evaluation, as there is no accurate or detailed normal ground truth data for real images, I mainly do the quantitative evaluation on synthetic images. Given an input image $\mathcal{I}$, ground truth material ($\mathbf{m}$) and illumination ($\mathbf{L}$) are provided and my network only predicts surface normals. The predicted surface normals along with $\mathbf{m}$ and $\mathbf{L}$ are used to synthesize $\mathcal{I}'$.

Quantitative results are provided in Table 4.1 respectively. It can be seen that L2 loss helps to get the orientation of the surface normals correct, and thus results in small errors, but produces blurry output. Adding the perceptual loss helps to preserve sharp features resulting in visually more plausible results. Also note that, although commonly used, L1 and SSIM metrics are not fully correlated with human perception.

	normals	**rendering**	
	cosine	L2	SSIM
standalone	0.9105	313.2058	0.9441
combined	0.9050	238.4619	0.9625

Table 4.1: The table shows the accuracy of the predicted surface normals and the images rendered using these together with ground truth material and illumination. The results of the normal prediction module trained standalone v.s. with the rendering layer are also provided.

4.5 Discussion & Future Work

In this Chapter, I show the formulation, which both penalizes the difference between ground truth normal and predicted normal but also regularize whether the image generated from estimated normal matches with the image generated from the real normal, is effective at predicting fine-scale normal prediction. Results on both synthetic and real images are shown. However, due to the lack of accurate ground truth normal for real scene images, the quantitative evaluation currently is only on synthetic images. It is still needed to explore how to evaluate the detailed normal predictions from real images.

Chapter 5: Motion Synthesis - Medial Axis based Motion Planning

Chapters 3 and 4 have discussed about extracting the independent properties for individual objects. In this chapter and paper [3], I will focus on analyzing the relative spatial and topological relationship among objects. More specifically, I will present a new method, called SVMA, to approximate the medial axis of robot's configuration space for motion planning problem. The 2D and 3D decompositions methods presented in Chapter 2 are critical to decompose the concave objects into nearly convex parts in order to extract the medial axis in the concave regions of those concave objects. To begin with, I will explore the similarity between medial axis and max-margin scheme in optimization, specifically, Support Vector Machine. Based on this similarity, I propose a new method to quickly construct the medial axis for the motion planning environment both in low and high dimensional space. However, directly applying the SVM classification on the large volume of uniformly sampled configurations suffers from huge computation and the medial axis is usually not the real medial axis due to SVM's optimization function's tolerance to the mis-classification. Instead, I show a method that can quickly push any configuration to the medial axis by using the characteristics of the Max-Margin's optimization function. Comparison will be presented to show how the proposed method can speed up the computation of the medial axis extraction for motion planning tasks.

5.1 Introduction

Medial axis (MA), a set of points that locally maximize the distance to obstacles, is known to be an important structure in robotic motion planning [134]. Maintaining clearance, or

distance from obstacles and sampling efficient enough configurations on the medial axises are a vital component for successful motion planning. By planning motions on the medial axis not only create safer paths for robots, but also offers higher possibility to find a path in complex environment where the feasible configuration space only occupies a small proportion of the whole space. However, computing the exact representation of the MA in configuration space whose dimensionality is usually greater than three is highly nontrivial [135] and can only be approximated. Some researchers have proposed methods to approximate the medial axis using sampling. Medial Axis Probabilistic Road Map (MAPRM) proposed by Wilmarth et al. [14] retracts the configuration (either collision free or not), to the medial axis. This is extended to high dimension spaces by Lien et al. [15], who used a large number of random query rays to approximate the clearance and penetration depth. Either of these two methods can't control the distribution of the samples on medial axis. Yeh et al. [16] tried to solve this issue using Uniform Medial-Axis PRM (UMAPRM), which samples configurations on medial axis more uniformly.

Even though, to some extent, those MAPRM-variants can tackle the "narrow passage" issue, their computation time is dominated by approximating clearance and penetration depth, which is very computationally expensive, especially in high dimensional space. Moreover, MAPRM-variants can't guarantee that the edges connecting the samples will be on or near the MA, which usually increase both roadmap connectivity and path clearance otherwise. Instead, they rely on another local planner to retract every discretization of connection to the medial axis. Such local planner will further cause huge computation cost.

Proposed Work. This work proposes a solution that allows a configuration, free of collision or not, to be efficiently retracted to the MA. The proposed method achieves this without excessively expensive geometric proximity queries. The main idea is based an observation that constructing the MA from a set of obstacles can be viewed as a multi-class training and classification problem. Each obstacle is regarded as a label and the labels of samples in the configuration space (C-space) is determined according to which obstacle is the closest. Under this setting, the MA of the free C-space can be viewed as the classification

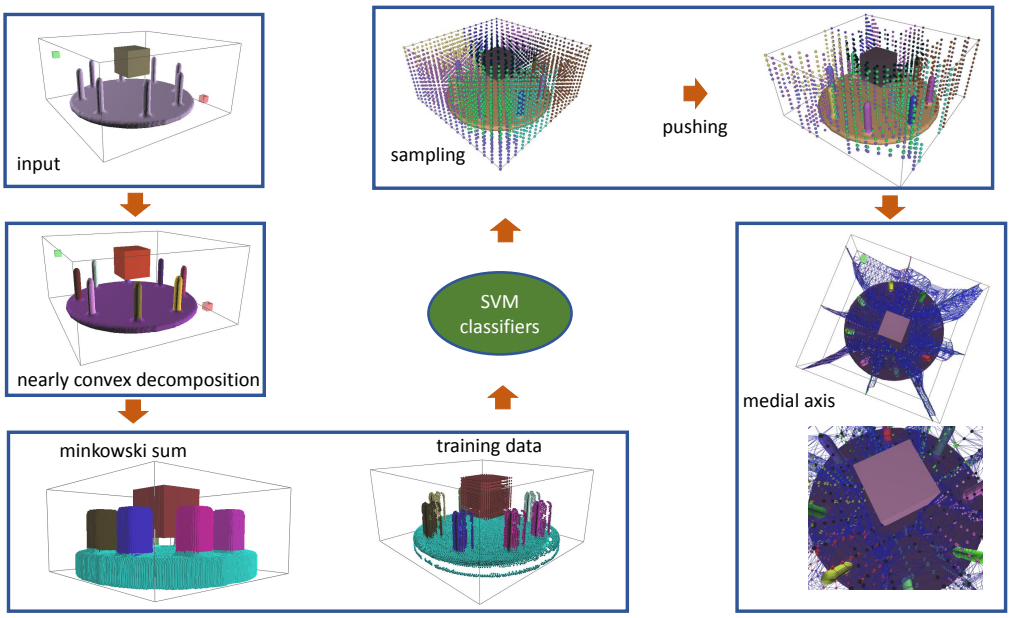


Figure 5.1: Overall pipeline of SVMA.

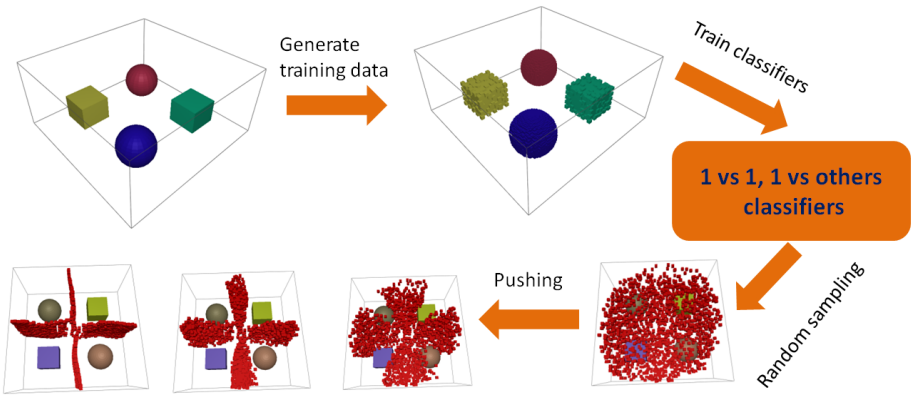


Figure 5.2: Training and Pushing pipeline of SVMA.

boundary. Therefore, approximating the MA can be transformed to finding the hyperplane that has the largest distance to the nearest training data point of any-class (known as the functional margin).

Fig. 5.1 illustrates the main steps of the proposed method called SVMA. Figure 5.1 shows the overview of the whole pipeline of my proposed framework:

- taking the environment as input, nearly convex decomposition methods in Chapter 2 will be first used to decompose the 3D models (obstacles) into nearly convex parts;
- compute the minkowski sum between each decomposed part and the robot;
- use the contact configurations from minkowski sum as the training data to train one v.s. one and one v.s. all others classifiers;
- sample random configurations in the working space and use the learned classifiers to push to the medial axis;
- build graph connection using the neighboring informations from the pushed configurations.

Figure 5.2 shows the more concrete explanations of the training and pushing stages and the step by step visualization of pushing. In the rest of this Chapter, I will focus more on describing the training and pushing in detail.

In machine learning, the widely-used Support Vector Machine (SVM) is a technique that achieves the binary classifier by constructing a hyperplane that has maximum margin. The proposed method SVMA uses the connection between SVM and the medial axis to accelerate the retraction process (discussed in Section 5.3).

The basic SVM only supports binary classification. Extensions to multiple-classes cases are required to handle motion planning problems with more than two obstacles. In Section 5.3.5, I will discuss an efficient multi-class classification approach by combining both the pairwise one-vs.-one SVM and one-vs.-others SVM. As shown in Fig. 5.1, SVMA has two stages: the training stage and the pushing (retraction) stage. In the **training stage**,

I train two types of classifiers: pairwise one-vs.-one SVM and one-vs.-others SVM. Given n obstacles, I would have n one-vs.-other SVM classifiers and $n(n-1)/2$ one-vs.-one classifiers. In the **pushing stage**, I first randomly sample a configuration in the environment, and one-vs.-other SVM classifiers are used to determine the closest obstacles, m and n, for this configuration. Then SVMA uses the pairwise one-vs.-one SVM classifier for m and n to push the configuration towards the medial axis. The last two steps are repeated until convergence. Details about training and pushing will be discussed in Section 5.3.5. My experimental results show that the proposed method provides a fast way to generate dense configurations on the medial axis with bounded approximation error controled by a user parameter.

5.2 Related work

Motion Planning with the Medial Axis Techniques to construct the medial axis (MA) have been widely explored, such as in graphics and robotics. In this section, I briefly review the use of MA in motion planning literature. Much research has been proposed for motion planning with the workspace medial axis. Few motion planning research has been done using the medial axis in arbitrary C-space. Surveys [134, 136] summarize earlier work in this area. Here I will concentrate on more recent work using probabilistic roadmap methods (PRMS).

Foskey et al. [137] propose a hybrid approach which uses a discrete approximation of the generalized Voronoi Diagram of the workspace to find an estimated path. Then they use a randomized path planner to bridge invalid segments along the path.

Hoff et al. [138] present a motion planning method that employs the approximate medial axis computed by graphics hardware. They successfully apply this technique to a 2D dynamic environment composed of more than 140,000 polygons. In [139], Holleman and Kavraki approximate the workspace medial axis by employing a similar method to [140].

Choset and Burdick [141] use the generalized Voronoi graph (GVG) to serve as a basis

for sensor based robot motion planning. The GVG is an extension of the medial axis. The medial axis is the set of all points that are equidistant to at least two obstacles. The GVG is the set of all points in m dimensions that are equidistant to m obstacles. The GVG is not guaranteed to be connected in dimensions greater than two, so additional structures are introduced to link the disconnected components of the GVG. The resulting structure is the hierarchical generalized Voronoi graph (HGVG). The HGVG is proven to be connected, and can be built incrementally using distance measurements from sensors.

The Equidistance Diagram (EDD) [142] is a roadmap method based on the medial axis of the workspace. Roadmap nodes are the local maxima of a clearance function defined in the workspace.

Multi-classifier using maximum margin. Learning the multi-classifier is one of the main goals in machine learning. Among the widely used classifier models, SVM [17], short for support vector machine is a supervised model used for classification, regression, or other tasks. SVM [143] constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space. Choosing the hyperplane is based on the intuition that a good separation is achieved by the hyperplane that has the largest distance to the nearest trainig data point of any-class (so-called functional margin), since in general the larger the margin in lower the generalization error the classifier. SVM is widely used due to its solid mathematic background support, its fast computation speed and high classification accuracy, support vector machine has been successfully applied for a lot of tasks including text classification [144], object detection [145], [146] etc. Pan et al. [147] used SVM to estimate the penetration depth, a challenging prolem in geometry and motion planning. Even though SVM is a binary classifier, it is also widely used for multi-classification task. The general way to extend to multi-class classification is to use the one-versus-all or one-versus-one [148]. Even though more complex methods have formulated the multi-classification problem as a one-shot problem and built a single optimization function [149], they usually require much longer training time and produce much more larger model size.

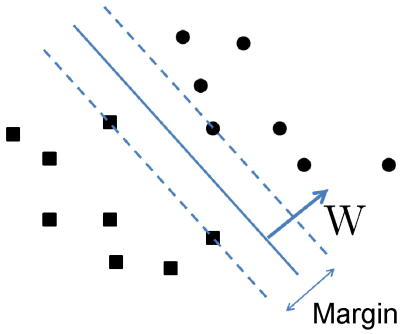


Figure 5.3: Linear separating hyperplane of SVM [150]. W is the normal vector of the separating hyperplane.

5.3 Max-Margin and Medial Axis

The medial axis of the free space partitions the C-space into Voronoi-cell-like regions. Each region corresponds to a connected component of C-obstacle. All configurations inside that region must be closer to its associated obstacle than to the other obstacles. If I view this property as a classification problem, I can also say that the label of these configurations is their corresponding obstacle. Thus, the MA is where a configuration has similar chance of being labeled from two or more obstacles. Based on this observation, the problem of approximating the medial axis can be transformed to a problem of finding the classification boundary for the labeling (classification) problem.

In this section, I will describe a method called SVMA that uses the characteristics of the max-margin optimization function to push a configuration onto the classification boundary. This is achieved by using the derivative of the SVM Max-Margin classification function as the pushing direction and the classification score as the step size.

5.3.1 Max margin and separating hyperplane

Support Vector Machine (SVM) finds a linear hyperplane to separate positive and negative data in feature (kernel) space. The hyperplane is parameterized by $\mathbf{w}$ and b, where $\mathbf{w}$ is the normal vector and b is the offset of the hyperplane. The distance from the closest point to the hyperplane in feature (kernel) space is called "margin". SVM, as shown in Fig 5.3 minimizes $||\mathbf{w}||^2$, subject to the constraint:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 >= 0 \ ,$$

where $y_i \in \{-1, 1\}$ indicates the label of $\mathbf{x}_i$. If the kernel function is used, the $f(\mathbf{x})$ becomes $f(\mathbf{x}) = \sum \alpha_i y_i K < \mathbf{x}_i, \mathbf{x} > +b$ (K is the kernel function). Fig 5.3 shows the visualization of the linear separating hyperplane for the separable case.

The testing data on the separating hyperplane, corresponding to $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$, would have equal chance to be assigned with positive label and negative label. In other words, the testing data has the equal "classification distance" to positive support vectors and negative support vectors. Thus, with the similar spirit, the separating hyperplane can be regarded as the "medial axis" between the positive support vectors and negative support vectors. If the positive training data and negative training data are from two geometric objects, the separating hyperplane can be regarded as the approximate medial axis. In motion planning, the positive and negative training data can come from the contact configurations of two obstacles.

5.3.2 Modeling the medial axis

The approximate medial axis (i.e., separating hyperplane) corresponds to the set of $\{\mathbf{x}\}$ where $f(x) = \mathbf{w}^T\mathbf{x} + b = 0$. To obtain a collection of medial axis configurations $\{\mathbf{x}\}$, directly solving $\mathbf{x} = f^{-1}(0)$ (the inverse function of $f(\mathbf{x}) = 0$) is impossible when some more complex kernel function like RBF is introduced.

Another approach is to partition the whole space into uniform grids and determine the labels for each grid cell. Then, the grid cells which have neighboring grid cells with different labels are retained and the rest grid cells are filtered out. The remaining cells must enclose the MA. Unfortunately, this approach requires to test many samples whose size is exponential to the robot's degrees of freedom, thus impractical for higher dimensional space. In the next section, I introduce an efficient pushing-based approach to approximate the medial axis.

5.3.3 Push configurations to the Medial Axis

Pushing a configuration $\mathbf{x}_0$ to medial axis is inspired by the intrinsic idea of maximal margin. The pushing strategy is similar to gradient decent algorithm, but with the step size automatically determined. Given the SVM classification $f(\mathbf{x})$, the pushing is achieved by iteratively updating $\mathbf{x}_{n+1} = x_n - \frac{f'(\mathbf{x})f(\mathbf{x})}{||f'(\mathbf{x})||^2}$ until $|f(\mathbf{x}_{n+1})| < \epsilon$. In this chapter, ϵ is set to be 0.02 throughout all experiments. Configurations generated in this way are approximately on the medial axis. From the view point of SVM classification, the approximation error of these configurations are bounded by this parameter ϵ. Since the classification score field is continuous, given enough samples of such kind, their connecting edges would be on the approximated medial axis too.

I proved the correctness of this gradient-based approach from the perspectives of the intrinsic idea of SVM and from the perspective of gradient decent based optimization. Details of the proofs are shown in the supplemental material.

5.3.4 Handle workspace with more than two obstacles

The SVM is a binary classifier. When there are more than two obstacles in the environment, the followings parts discuss the limitations in using either one-vs.-one classifier or one-vs.-others classifier in this section.

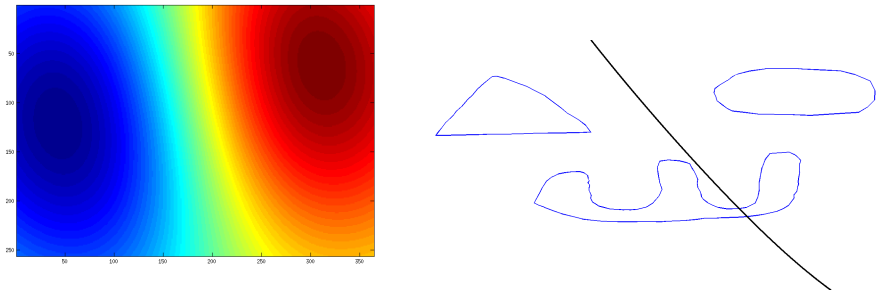


Figure 5.4: The figure shows the drawback of extracting medial axis using one-vs.-one classifier only. Left: the classification score $f(\mathbf{x})$ field of the one-vs.-one classifier between the two obstacles on the top. Right: the extracted medial axis of this one-vs.-one classifier.

one-vs.-one classifier. Let $\mathbf{x}$ be a set of configurations on the separating hyperplane between object A and object B. I observe that some configurations in $\mathbf{x}$ may be not on the medial axis. For example, some points on the separating hyperplane between A and B can be inside a third object C as shown in Fig. 5.4.

one-vs.-others classifier If I extract the medial axis by find the points where $|g| \approx 0$, for a one-vs.-others classifier g. The extracted medial axis underestimates the region that should 'belong' to this obstacle. Take Fig 5.5 as an example. The score field of the one-vs.-others classifier is shown in the left two and top-right images of Fig 5.5. The bottom right image is the medial axis extracted using one-vs.-others classifier for the bottom obstacle. It can be seen that the medial axis for the one-vs.-others classifiers is too close to the classifier.

Combining one-vs.-one and one-vs.-others classifiers. From the discussions above, it's known that both only using one-vs.-others classifier and one-vs.-one classifier do not extract the correct medial axis. To solve the problem, SVMAneed to find the answers for these questions: for a random configuration or a random region in the space, SVMAneed to know which obstacles are closer, which part of the medial axis SVMAshould push to and which classifier SVMAneeds to use for pushing. The solution to these question is to combine one-vs.-one classifier and one-vs.-others classifier. SVMAuse one-vs.-others classifiers to identify which obstacles are closer and whose one-vs.-one classifier SVMAshould use for pushing; then SVMAuse the identified one-vs.-one classifier for pushing. To validate this solution, score

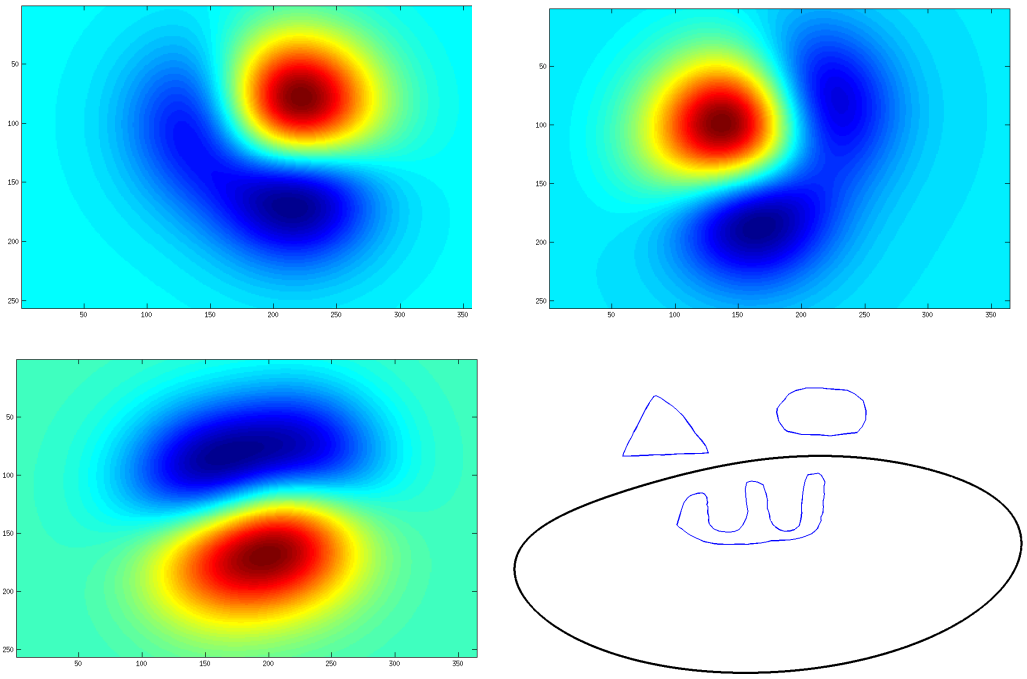


Figure 5.5: This figure shows the drawback of extracting medial axis using one-vs.-others classifier. From left to right, top to bottom: three images show the classification score field of three one-vs.-others classifier for each of the three obstacles. Bottom-right: The extracted medial axis using the bottom obstacle's one-vs.-others classifier.

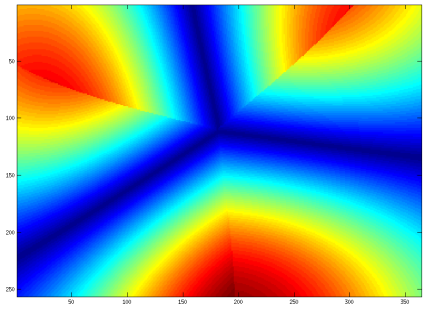


Figure 5.6: The medial axis field of combining one-vs.-one and one-vs.-other classifiers. Apply the one-vs.-other classifier to get the top two nearest obstacles m and n. Then compute the $|f_{m,n}(x)|$. The blue means lower value and red represents higher value. Note that there are some discontinuity in the red regions. These discontinuity belongs to two different pairwise one-vs.-one SVM classifier; the second topmost label (obstacle) has changed between the two sides of the discontinuity. For example, for the left most discontinuity edge, the discontinuity edge' up-side's top two labels are triangle obstacle and the ellipse obstacle; the bottom side for triangle obstacle and $w - shape$ obstacle. The discontinuity is caused by the classification score's scale difference between the two pairwise SVM classifiers on the either side.

field of this combining solution is visualized in Fig 5.6. SVMAfirst use one-vs.-other classifiers to get the right one-vs.-one classifier $f_i(\mathbf{x})$ and then compute and visualize the value $|f_i(\mathbf{x}_0)|$ for the current configuration $\mathbf{x}_0$. The general procedure for generating configurations on medial axis is: *a.* for a configuration $\mathbf{x}_0$, SVMAfinds the "closest" 2 obstacles using all the one-vs.others classifiers' classification score as the measurement; higher classification score means closer. *b.* get the one-vs.-one classifier corresponding to those two "closest" obstacles, and apply the pushing strategy discussed in section 5.3.3.

5.3.5 svma Push Algorithm

In this section, I summarize the whole algorithm of SVMA pushing. Given a random configuration $\mathbf{x}$, the pushing is an iterative process. For each iteration, SVMAuse all one-vs.-others classifiers to get two obstacles s, t with the two top classification scores; then use the one-vs.-one classifier $f_{st}(\mathbf{x})$ of these two obstacles to get pushing direction $f'_{st}(\mathbf{x})$ and pushing step size $f_{st}(\mathbf{x})$; repeat the pushing until the step size is slower than the threshold. The

overall procedure can be described as Aglorithm 4:

Algorithm 4 Pushing Algorithm

Input: a configuration $\mathbf{x}$, stopping pushing threshold ϵ, one-vs.-one classifiers f and one-vs.-others g

Output: updated $\mathbf{x}$ on the approximate medial axis

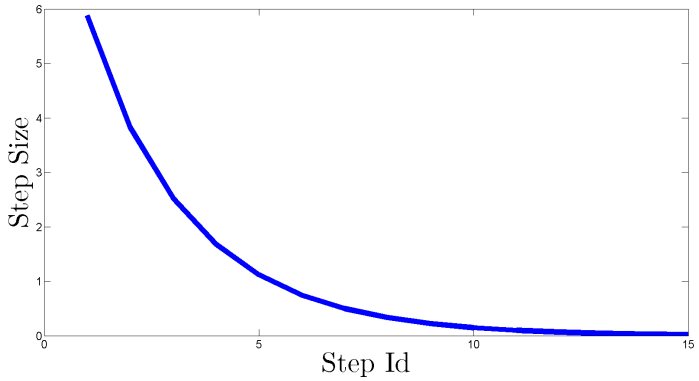
1: **while** $d > \epsilon$ **do**
2: $\quad \mathbf{x} = \mathbf{x} - \frac{f'_{st}(\mathbf{x})}{||f'_{st}(\mathbf{x})||^2} d$
3: $\quad$ find s and t such that $g_s \geq g_t \geq g_i, \forall i \in \{1..n\} \setminus \{s, t\}$
4: $\quad$ step size $d = f_{st}(\mathbf{x})$
5: **end while**

The Fig 5.7(a) shows how the step size changes during the pushing. The bottom part of Fig 5.1 shows the pushing process of an environment containing four obstacles.

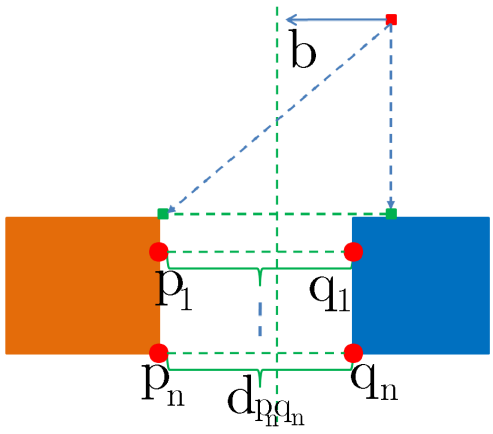
5.4 Step Size Optimization

I have discussed using the SVM score as pushing step size. In this section, I will discuss its limitation and alternative approaches that use workspace clearance to obtain better step size.

The training process of SVM is to minize the overal optimization function. So if the positive and negative sample are not lineary separable, some may be misclassified serving as compromise. As a consequence, the classification boundary is not the real medial axis. If I directly use the SVM's score as the step size and stopping criteria for pushing, some of the pushing result is not on the medial axis. For example, Fig. 5.8(a) is the constructed medial axis with SVM classification score as the step size. However, part of the medial axis goes through the obstacles.

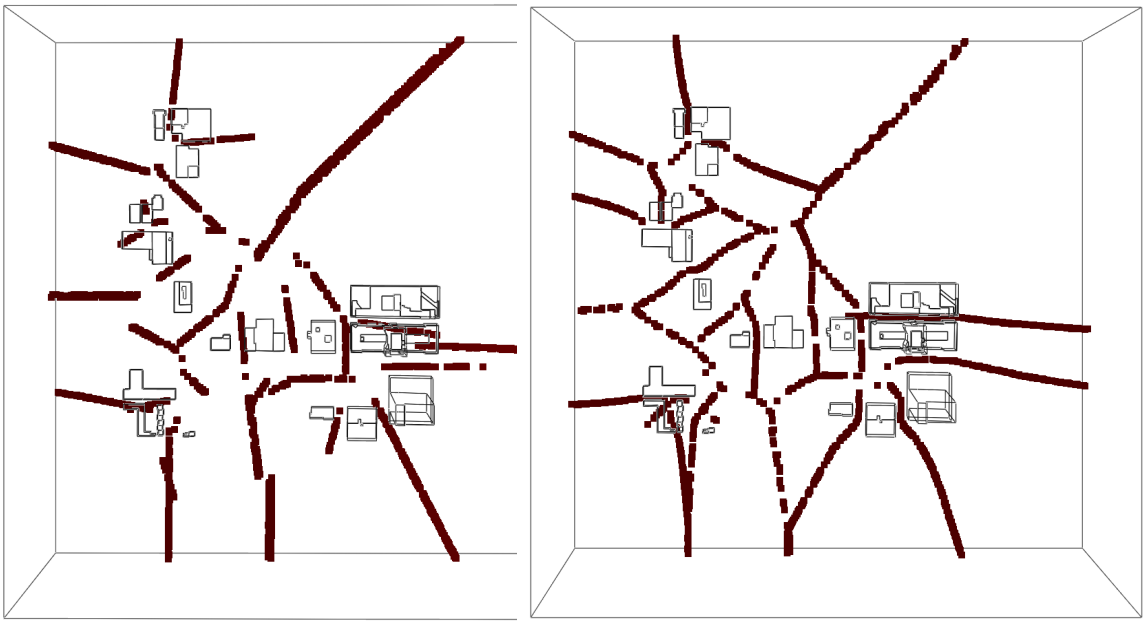


(a)



(b)

Figure 5.7: (a): the changing of step size during pushing; (b): the two cubes represent two obstacles. p_1, p_n, q_1, q_n are the support vectors.



(a) (b)

Figure 5.8: The medial axes (a) using SVM score as step size and (b) using clearance-based difference as step size.

5.4.1 Workspace clearance difference as step size

The distance measurement and the SVM score are in different domains and have different units. In order to utilize the distance difference for the step size, first I need to find the relationship between the distance and SVM score. Support vectors are used for computing it. The procedure is below: 1. for each positive support vector p, I find the nearest negative support vector q. 2. place the robot on that support vector (the support vector is a configuration); compute the distance $d_{pq}$when the robot is placed on these two configuration. 3. repeat 1 and 2 for all the positive support vectors. The relation between score difference and distance is $\frac{1}{n}\sum_{i=1}^{n}\frac{f(p_i)-f(q_i)}{d_{p_iq_i}}$.

To determine the pushing step size for any configuration, shown as Fig 5.8(b), I place the robot onto that configuration; then I use PQP [151] to get the point that is the closest to the robot for those two obstacles; then get the perpendicular bisector l of these two

Table 5.1: Comparison with MAPRM

Env	ObN	Method	Number			Time(s)				
			MKSum Num	Train Num	Pushing Num	MKSum	Train	Push	Conn	Total
2D	354	MAPRM	-	-	40000	-	-	439.16	275.96	715.2
		SVMA	70800	70800	35000	0.33	82.23	323.01	40.14	448.24
		SVMA	70800	70800	40000	0.33	86.86	391.11	61.30	542.32
3D	148	MAPRM	-	-	20000	-	-	904.65	431.72	1337.06
		SVMA	69469	59200	20000	0.59	160.02	87.40	73.89	323.42
6D	2	MAPRM	-	-	2000	-	-	19.03	27.61	47.09
		SVMA	27478	3000	2000	3.23	0.47	3.61	11.96	19.33

points; later compute the distance b between the robot center and l; the pushing step size is $b\frac{1}{n}\sum_{i=1}^{n}\frac{f(p_i)-f(q_i)}{d_{p_iq_i}}$. And in the same way, the pushing stopping criteria is when the step size is smaller than a threshold ϵ.

5.4.2 Combining SVM score and clearance difference

I also provide another option that is to use the SVM score as the step size to roughly push the configuration to the medial axis and then change the way to determine the step size by using clearance difference (PQP distance difference). That is to combine the SVM score and clearance difference for pushing. However, I observed that sometimes the improvement of combination of two over only using clearance is insignificant. One reason is that for some very complex environment, SVM classification would have a lot of misclassification. Thus, for the combination approaching, the initial pushing results would be farther from the real medial axis.

5.5 Experiments and results

In order to validate my method, I implemented SVMA in C++. All experimental results are collected from a Linux Virtual Machine with 4G RAM hosted on Windows 7 with Intel Core i7-2640M CPU 2.8G. The SVM implementation is adapted from libsvm [148].

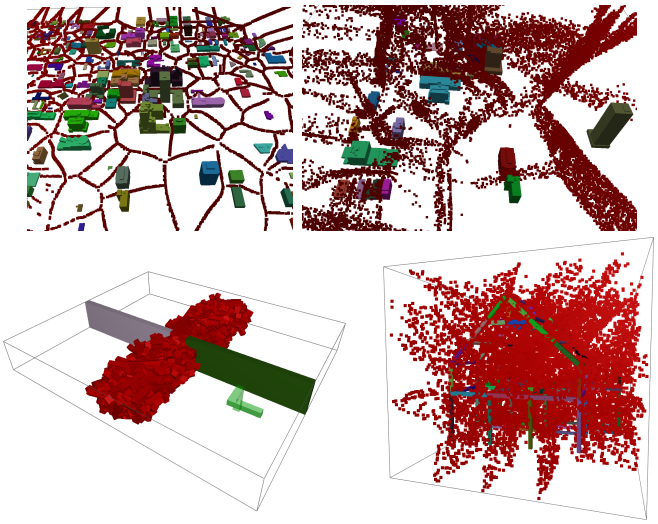


Figure 5.9: Configurations generated by SVMA

5.5.1 Training data

The training data are obtained from two sources: 1. the configurations from the contact space, and 2. the configurations that are in-collision with the obstacle. I used Minkowski-sum to geneate configurations on contact space. Generating configurations on contact-space is more time-consuming compared to doing it in collision space. Intuitively, configurations on contact space would be better for max-margin. In 2D and 3D, generating the sample configurations on the contact-space does not cost too much time comparing to training the SVM models. Thus, I used the configurations in contact space as my training data. For higher dimension, I can use either contact space or obstacle space.

5.5.2 Experiment Setting

As shown in Fig. 5.9, I tested four environments with 2D, 3D, 6D C-space. The first environment contains 354 3D buildings from Oklahoma City. In my 2D experiment, I use their 2D footprints on the ground. I also use these models to construct the medial axis in 3D. Another 3D model is a skeleton model of a house, which is decomposed into 148 cylinders

using the nearly convex decomposition methods. The environment with 6D has a robot made up of 3 parts and can translate and rotate, enabling the total degree of freedom to be 6D. Figure 5.9 shows the built medial axis configurations for those environments. For the 2D and 3D examples, my goal is to have enough samples that can approximate the medial axis of all the obstacles in the space. The robot for these two environment is a point robot. However, for the 6D environment, my goal is to find the path for the robot through the narrow passage. The robot's geometry contains three parts that are perpendicular to each other and needs to rotate when it goes through the narrow passage. From the Table 5.1, I found that the main time cost are computing Minkowski-sum in 6D and connecting the configurations. While for this environment, two obstacles are separable, thus, training the SVM is very fast.

5.5.2.1 Comparison with MAPRM

And another observation is that the medial axis from MAPRM is generated with the help of approximation using rays. The pushed result is less accurate than the pushing result of my method. In other words, my medial axis configurations are more concentrated. In this way, the length of the connecting edges would be shorter too. Thus, less interpolation is needed to check whether this edge is valid or not. As a result, the connecting time of my method is also shorter than the MAPRM. And pushing using my method is faster than MAPRM.

5.5.3 Step Size Strategy

As mentioned above, the constructed medial axis using clearance difference as step size has better quality than the SVM score. I will discussing the running time difference.

I set the maximum pushing steps for each method to be the same. However, 80% to 90% of the configurations can be pushed to the MA in 10 steps. The running time is usually related to the density of obstacles and the SVM model's complexity. The former

Table 5.2: The running time comparisons for step size using SVM score, clearance difference and their combination. The combination approach would have 30% steps using SVM score and the rest 70% using clearance difference.

Env	OB N	Sample N	SVM	Clearance	Combine
2D	354	10000	94.69	100.41	101.03
3D	148	10000	56.84	49.48	52.08
6D	2	10000	20.74	29.85	22.01

will influence the speed of collision checking and computing clearance difference, while the latter relates to the time cost for classifying a configuration. Generally, using SVM as the step size is faster than using clearance difference as step size, because running one time of the classification is often faster than doing one time collision detection. This is also one advantage of my method, especially when the geometry structure of the obstacle is complex. However, when the geometry of the obstacle is simple and only approximated by few vertices, it is not an advantage any more. For example the obstacles in 2D and 3D examples in Fig 5.9, the obstacles are simple geometry object. On the other hand, for 2D examples, the obstacles have very different scales and some of them are very close at their concave parts. This makes the SVM models more complex. Thus, the time-cost difference between using SVM score as step size and using clearance difference as step size is very small. The same to the combination of them. While in 6D examples, two obstacles doesn't touch each other. As a result, the learned SVM model is less complex. Using SVM score as step size is faster than using clearance difference as step size. The combination of them is between them.

5.6 Discussion & Future Work

In summary, I presented a method that can generate approximate medial axis method based on SVM model. The building processing is based on pushing, where the pushing direction and step size can be automatically determined. This pushing scheme can fast generate configurations on the medial axis. The proposed method can address the medial

axis motion planning problem for both two obstacles cases and multiple obstacle cases. Experiments show that SVMAcan generate medial axis faster than previous methods. This work also appeared in [3].

Chapter 6: Conclusion & Future Work

In this dissertation, the common theme has been to develop methods to estimate the shape and physical properties from 2D or 3D data. As these estimations usually suffer from several difficulties. I presented three representative topics of predicting shape and physical properties. Each of them represents how one of these difficulties can be addressed.

In the appearance synthesis topic, I have proposed methods to tackle the issue of under-constrained setting. The rendering layer is introduced to enable an end-to-end physical based rendering network. The model trained through this network has produced plausible and robust material editing results for both synthetic data and real data. In the shape synthesis part, I have shown a deep learning model using the formulation which penalize both the difference between predicted normal and ground truth normal and the difference between images generated using the estimated normal and the original input images. This formulation enable us to get final-scale normal prediction even when there is the difficulty that ground truth normal data is noisy or missing. In the motion synthesis part, a pushing approach based on support vector machine is introduce to approximate the medial axis of a robot's configuration space, which original has the issue of expensive computation cost. The efficiency and effectiveness of such approximation has been validated by the experiments and comparisons with other alternatives.

One prerequisite of applying these methods is to prepare feasible and sufficient data. In Chapter 2, I have proposed a new part-aware shape feature called continuous visibility feature and nearly convex decomposition approaches for 2D and 3D shapes. These approaches and feature can produce segmentation components which are semantically meaningful and with bounded geometric constraints. These semantically meaningful parts are used to simulate the rendering data for appearance synthesis and shape synthesis tasks. Decomposition results with bounded geometric constraints will also be used to train the support vector

machine models for the motion synthesis task.

There are several ways to extend the approaches proposed in this dissertation. For the material editing part, my current pipeline assumes the whole object (image) contains the same material or the segmentation masks for material are given. It would be valuable to explore some ways which enables interactive material editing easily. For the normal prediction part, although detailed normal information can be predicted from real images, how to evaluate the detail and correctness of the predicted normal is still an unsolved problem due to the fact that there is no accurate and detailed normal for real images.

On the other hand, this dissertation only presents three tasks to show how the corresponding three difficulties of physical and shape property estimation can be addressed. There are other difficulties that need to be explored in those property estimation problems. I hope this dissertation inspires new directions for addressing other difficulties and also exploring other property estimation research.

Bibliography

[1] G. Liu, Z. Xi, and J.-M. Lien, "Dual-space decomposition of 2d complex shapes," in *27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH: IEEE, Jun. 2014.

[2] G. Liu, Y. Gingold, and J.-M. Lien, "Continuous visibility feature," in *28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA: IEEE, Jun. 2015, pp. 1182 – 1190.

[3] G. Liu and J.-M. Lien, "Fast medial axis approximation via max-margin pushing," in *IEEE/RSJ International Conference on Intelligent Robot and System (IROS)*, 2015.

[4] G. Liu, Z. Xi, and J.-M. Lien, "Nearly convex segmentation of polyhedra through convex ridge separation," in *Symposium on Solid & Physical Modeling (SPM); also appears in Journal of Computer-Aided Design*, vol. 78, Sep. 2016, pp. 137–146.

[5] G. Liu, D. Ceylan, E. Yumer, J. Yang, and J.-M. Lien, "Material editing using a physically based rendering network," in *Internation Conference on Computer Vision (ICCV)*, Venice, Italy, Oct. 2017.

[6] S. Lombardi and K. Nishino, "Reflectance and natural illumination from a single image," in *IEEE ECCV*. Springer, 2012, pp. 582–595.

[7] J. T. Barron and J. Malik, "Shape, illumination, and reflectance from shading," *IEEE PAMI*, 2015.

[8] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, "Deep convolutional inverse graphics network," in *NIPS*, 2015, pp. 2539–2547.

[9] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *IEEE ICCV*, 2015, pp. 2650–2658.

[10] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," *Computer Vision–ECCV 2012*, pp. 746–760, 2012.

[11] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[12] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[13] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[14] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, vol. 2, 1999, pp. 1024–1031.

[15] J.-M. Lien, S. L. Thomas, and N. M. Amato, "A general framework for sampling on the medial axis of the free space," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, September 2003, pp. 4439–4444.

[16] H.-Y. Yeh, J. Denny, A. Lindsey, S. L. Thomas, and N. M. Amato, "Umaprm: Uniformly sampling the medial axis," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2014.

[17] V. Vapnik, S. E. Golowich, A. Smola *et al.*, "Support vector method for function approximation, regression estimation, and signal processing," *Advances in neural information processing systems*, pp. 281–287, 1997.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing*

Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States., 2012, pp. 1106–1114. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 580–587. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.81

[20] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," *CoRR*, vol. abs/1505.04366, 2015. [Online]. Available: http://arxiv.org/abs/1505.04366

[21] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, 2013, pp. 6645–6649. [Online]. Available: http://dx.doi.org/10.1109/ICASSP.2013.6638947

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1538–1546.

[25] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for cnn: Viewpoint estimation in

images using cnns trained with rendered 3d model views," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2686–2694.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[27] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *The Visual Computer*, vol. 24, no. 4, pp. 249–259, 2008.

[28] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3d mesh segmentation," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, pp. 73–73, 2009.

[29] J. M. Keil, "Polygon decomposition," in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Amsterdam: Elsevier Science Publishers B.V. North-Holland, 2000, pp. 491–518.

[30] B. Chazelle, "A theorem on polygon cutting with applications," in *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, 1982, pp. 339–349.

[31] D. H. Greene, "The decomposition of polygons into convex parts," in *Computational Geometry*, ser. Adv. Comput. Res., F. P. Preparata, Ed. Greenwich, Conn.: JAI Press, 1983, vol. 1, pp. 235–259.

[32] J. M. Keil, "Decomposing a polygon into simpler components," *SIAM J. Comput.*, vol. 14, pp. 799–817, 1985.

[33] B. Chazelle and D. P. Dobkin, "Optimal convex decompositions," in *Computational Geometry*, G. T. Toussaint, Ed. Amsterdam, Netherlands: North-Holland, 1985, pp. 63–133.

[34] M. Keil and J. Snoeyink, "On the time bound for convex decomposition of simple polygons," in *Proceedings of the 10th Canadian Conference on Computational Geometry*, M. Soss, Ed. Montréal, Québec, Canada: School of Computer Science, McGill University, 1998, pp. 54–55. [Online]. Available: citeseer.nj.nec.com/keil98time.html

[35] K. Siddiqi and B. B. Kimia, "Parts of visual form: Computational aspects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 3, pp. 239–251, 1995.

[36] H. Liu, W. Liu, and L. Latecki, "Convex shape decomposition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, june 2010, pp. 97 –104.

[37] W. L. Zhou Ren, Junsong Yuan, "Minimum near-convex shape decomposition," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 10, 2013, pp. 2546–2552.

[38] Z. Ren, J. Yuan, C. Li, and W. Liu, "Minimum near-convex decomposition for robust shape representation," in *Proc. of ICCV*, 2011.

[39] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polygons," *Comput. Geom. Theory Appl.*, vol. 35, no. 1, pp. 100–123, 2006.

[40] R. Juengling and M. Mitchell, "Combinatorial shape decomposition," in *Proceedings of the 3rd international conference on Advances in visual computing-Volume Part II*. Springer-Verlag, 2007, pp. 183–192.

[41] X. Mi and D. DeCarlo, "Separating parts from 2d shapes using relatability," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.

[42] M. Attene, S. Katz, M. Mortara, G. Patané, M. Spagnuolo, and A. Tal, "Mesh

segmentation-a comparative study," in *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on.* IEEE, 2006, pp. 7–7.

[43] A. Shamir, "A survey on mesh segmentation techniques," in *Computer graphics forum*, vol. 27, no. 6. Wiley Online Library, 2008, pp. 1539–1556.

[44] A. Golovinskiy and T. Funkhouser, "Consistent segmentation of 3d models," *Computers & Graphics*, vol. 33, no. 3, pp. 262–269, 2009.

[45] E. Kalogerakis, A. Hertzmann, and K. Singh, "Learning 3d mesh segmentation and labeling," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, p. 102, 2010.

[46] Q. Huang, V. Koltun, and L. Guibas, "Joint shape segmentation with linear programming," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 125.

[47] L. Lu, Y. Choi, W. Wang, and M. Kim, "Variational 3d shape segmentation for bounding volume computation," in *Computer Graphics Forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 329–338.

[48] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of polyhedral surfaces using decomposition," in *Computer Graphics Forum*, vol. 21, no. 3. Wiley Online Library, 2003, pp. 219–228.

[49] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 954–961, 2003.

[50] H. Yamauchi, S. Lee, Y. Lee, Y. Ohtake, A. Belyaev, and H. Seidel, "Feature sensitive mesh segmentation with mean shift," in *Shape Modeling and Applications, 2005 International Conference.* IEEE, 2005, pp. 236–243.

[51] R. Liu and H. Zhang, "Mesh segmentation via spectral embedding and contour analysis," in *Computer Graphics Forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 385–394.

[52] J. Zhang, J. Zheng, C. Wu, and J. Cai, "Variational mesh decomposition," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 3, p. 21, 2012.

[53] Y. Wang, M. Gong, T. Wang, D. Cohen-Or, H. Zhang, and B. Chen, "Projective analysis for 3d shape segmentation," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 192, 2013.

[54] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3d mesh analysis," in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 5. ACM, 2008, p. 145.

[55] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, "Shape distributions," *ACM Transactions on Graphics (TOG)*, vol. 21, no. 4, pp. 807–832, 2002.

[56] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3d scenes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 5, pp. 433–449, 1999.

[57] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 4, pp. 509–522, 2002.

[58] A. Elad and R. Kimmel, "On bending invariant signatures for surfaces," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 10, pp. 1285–1295, 2003.

[59] V. Jain, H. Zhang, and O. van Kaick, "Non-rigid spectral correspondence of triangle meshes," *International Journal of Shape Modeling*, vol. 13, no. 1, pp. 101–124, 2007.

[60] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and computational harmonic analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[61] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," in *Computer Graphics Forum*, vol. 28, no. 5. Wiley Online Library, 2009, pp. 1383–1392.

[62] R. M. Rustamov, "Laplace-beltrami eigenfunctions for deformation invariant shape representation," in *Proceedings of the fifth Eurographics symposium on Geometry processing*. Eurographics Association, 2007, pp. 225–233.

[63] C. Li and A. B. Hamza, "A multiresolution descriptor for deformable 3d shape retrieval," *The Visual Computer*, vol. 29, no. 6-8, pp. 513–524, 2013.

[64] S. Asafi, A. Goren, and D. Cohen-Or, "Weak convex decomposition by lines-of-sight," in *Computer Graphics Forum*, vol. 32, no. 5. Wiley Online Library, 2013, pp. 23–31.

[65] O. van Kaick, N. Fish, Y. Kleiman, S. Asafi, and D. Cohen-Or, "Shape segmentation by approximate convexity analysis," *ACM Trans. on Graphics*, vol. to appear, 2014.

[66] R. Liu, H. Zhang, A. Shamir, and D. Cohen-Or, "A part-aware surface metric for shape analysis," in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 397–406.

[67] A. Lingas, "The power of non-rectilinear holes," in *Proc. 9th Internat. Colloq. Automata Lang. Program.*, ser. Lecture Notes Comput. Sci., vol. 140. Springer-Verlag, 1982, pp. 369–383.

[68] A. Lingas, R. Pinter, R. Rivest, and A. Shamir, "Minimum edge length partitioning of rectilinear polygons," in *Proc. 20th Allerton Conf. Commun. Control Comput.*, 1982, pp. 53–63.

[69] M. Tănase and R. C. Veltkamp, "Polygon decomposition based on the straight line skeleton," in *Proceedings of the nineteenth conference on Computational geometry (SoCG)*. ACM Press, 2003, pp. 58–67.

[70] T. K. Dey, J. Giesen, and S. Goswami, "Shape segmentation and matching with flow discretization," in *Proc. Workshop on Algorithms and Data Structures*, 2003, pp. 25–36.

[71] L. Wan, "Parts-based 2d shape decomposition by convex hull," in *Shape Modeling and Applications, 2009. SMI 2009. IEEE International Conference on.* IEEE, 2009, pp. 89–95.

[72] Y. Lu, J.-M. Lien, M. Ghosh, and N. M. Amato, "α-decomposition of polygons," *Computer & Graphics*, vol. SMI 12 special issue, 2012.

[73] E. Behar and J.-M. Lien, "Dynamic minkowski sums under scaling," *Computer-Aided Design*, Nov. 2012.

[74] Z. Lian, A. Godil, P. L. Rosin, and X. Sun, "A new convexity measurement for 3d meshes," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 119–126.

[75] O.-C. Au, Y. Zheng, M. Chen, P. Xu, and C.-L. Tai, "Mesh segmentation with concavity-aware fields," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 7, pp. 1125–1134, 2012.

[76] H. Zimmer, M. Campen, and L. Kobbelt, "Efficient computation of shortest path-concavity for 3d meshes," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on.* IEEE, 2013, pp. 2155–2162.

[77] K. Mamou and F. Ghorbel, "A simple and efficient approach for 3d mesh approximate convex decomposition," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, Nov. 2009, pp. 3501 –3504.

[78] M. Attene, M. Mortara, M. Spagnuolo, and B. Falcidieno, "Hierarchical convex approximation of 3d shapes for fast region selection," in *Computer Graphics Forum*, vol. 27, no. 5. Wiley Online Library, 2008, pp. 1323–1332.

[79] V. Kreavoy, D. Julius, and A. Sheffer, "Model composition from interchangeable components," in *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on.* IEEE, 2007, pp. 129–138.

[80] M. Ghosh, N. M. Amato, Y. Lu, and J.-M. Lien, "Fast approximate convex decomposition using relative concavity," *Computer-Aided Design*, vol. 45, no. 2, pp. 494–504, 2013.

[81] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[82] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[83] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[84] H. Y. F. Feng and T. Pavlidis, "Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition," *IEEE Trans. Comput.*, vol. 24, pp. 636–650, June 1975.

[85] D. Hoffman and W. Richards, "Parts of recognition," *Cognition*, vol. 18, pp. 65–96, 1984.

[86] I. Biederman, "Recognition-by-components: A theory of human image understanding," *Psychological Review*, vol. 94, pp. 115–147, 1987.

[87] G. Borgefors and G. S. di Baja, "Methods for hierarchical analysis of concavities," in *Proceedings of the Conference on Pattern Recognition (ICPR)*, vol. 3, 1992, pp. 171–175.

[88] D. Attali, P. Bertolino, and A. Montanvert, "Using polyballs to approximate shapes and skeletons," in *ICPR94*, 1994, pp. 626–628.

[89] G. Borgefors and G. S. di Baja, "Analyzing nonconvex 2d and 3d patterns," *Computer Vision and Image Understanding*, vol. 63, no. 1, pp. 145–157, 1996.

[90] D. Hoffman and M. Singh, "Salience of visual parts," *Cognition*, vol. 63, pp. 29–78, 1997.

[91] M. Singh, G. Seyranian, and D. Hoffma, "Parsing silhouettes: The short-cut rule," *Perception & Psychophysics*, vol. 61, pp. 636–660, 1999.

[92] O. E. Badawy and M. Kamel, "Shape representation using concavity graphs," *ICPR*, vol. 3, pp. 461–464, 2002.

[93] L. Latecki, R. Lakamper, and T. Eckhardt, "Shape descriptors for non-rigid shapes with a single closed contour," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 1. IEEE, 2000, pp. 424–429.

[94] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational shape approximation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 905–914, 2004.

[95] J. Hershberger and J. Snoeyink, "Speeding up the Douglas-Peucker line simplification algorithm," in *Proc. 5th Internat. Sympos. Spatial Data Handling*, 1992, pp. 134–143.

[96] M. Müller, N. Chentanez, and T.-Y. Kim, "Real time dynamic fracture with volumetric approximate convex decompositions," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 115, 2013.

[97] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.

[98] C. Xian, H. Lin, and S. Gao, "Automatic cage generation by improved obbs for mesh deformation," *The Visual Computer*, vol. 28, no. 1, pp. 21–33, 2012.

[99] K. Crane, C. Weischedel, and M. Wardetzky, "Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow," *ACM Trans. Graph.*, vol. 32, 2013.

[100] H. G. Barrow and J. M. Tenenbaum, "Recovering intrinsic scene characteristics from images," AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Tech. Rep. 157, Apr 1978.

[101] G. Patow and X. Pueyo, "A survey of inverse rendering problems," in *CGF*, vol. 22, no. 4. Wiley Online Library, 2003, pp. 663–687.

[102] S. Lombardi and K. Nishino, "Reflectance and illumination recovery in the wild," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 129–141, Jan 2015.

[103] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah, "Shape from shading: A survey," *IEEE PAMI*, vol. 21, no. 8, pp. 690–706, 1999.

[104] M. K. Johnson and E. H. Adelson, "Shape estimation in natural illumination," in *Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 2553–2560.

[105] L. F. Yu, S. K. Yeung, Y. W. Tai, and S. Lin, "Shading-based shape refinement of rgb-d images," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, June 2013, pp. 1415–1422.

[106] E. A. Khan, E. Reinhard, R. W. Fleming, and H. H. Bülthoff, "Image-based material editing," in *ACM SIGGRAPH*, 2006, pp. 654–663.

[107] I. Boyadzhiev, K. Bala, S. Paris, and E. Adelson, "Band-sifting decomposition for image-based material editing," *ACM TOG*, vol. 34, no. 5, pp. 163:1–163:16, Nov. 2015.

[108] Y. Tang, R. Salakhutdinov, and G. Hinton, "Deep lambertian networks," in *ICML*, 2012.

[109] S. R. Richter and S. Roth, "Discriminative shape from shading in uncalibrated illumination," in *IEEE CVPR*, June 2015, pp. 1128–1136.

[110] T. Narihira, M. Maire, and S. X. Yu, "Learning lightness from human judgement on relative reflectance," in *IEEE CVPR*, June 2015.

[111] T. Zhou, P. Krhenbhl, and A. A. Efros, "Learning data-driven reflectance priors for intrinsic image decomposition," in *IEEE ICCV*, 2015, pp. 3469–3477.

[112] M. M. Takuya Narihira and S. X. Yu, "Direct intrinsics: Learning albedo-shading decomposition by convolutional regression," in *International Conference on Computer Vision (ICCV)*, 2015.

[113] K. Rematas, T. Ritschel, M. Fritz, E. Gavves, and T. Tuytelaars, "Deep reflectance maps," in *CVPR*, 2016.

[114] S. Georgoulis, K. Rematas, T. Ritschel, M. Fritz, L. V. Gool, and T. Tuytelaars, "Delight-net: Decomposing reflectance maps into specular materials and natural illumination," arXiv:1603.08240 [cs.CV], 2016. [Online]. Available: http://arxiv.org/abs/1603.08240http://arxiv.org/pdf/1603.08240v1.pdf

[115] J. Shi, Y. Dong, H. Su, and S. X. Yu, "Learning non-lambertian object intrinsics across shapenet categories," *arXiv preprint arXiv:1612.08510*, 2016.

[116] C. Innamorati, T. Ritschel, T. Weyrich, and N. J. Mitra, "Decomposing single images for layered photo retouching," in *Computer Graphics Forum*, vol. 36, no. 4. Wiley Online Library, 2017, pp. 15–25.

[117] M. M. Loper and M. J. Black, "OpenDR: An approximate differentiable renderer," in *Computer Vision – ECCV 2014*, ser. Lecture Notes in Computer Science, vol. 8695. Springer International Publishing, Sep. 2014, pp. 154–169.

[118] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis, "Radiometry," L. B. Wolff, S. A. Shafer, and G. Healey, Eds. USA: Jones and Bartlett Publishers, Inc., 1992, ch. Geometrical

Considerations and Nomenclature for Reflectance, pp. 94–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=136913.136929

[119] W. Matusik, H. Pfister, M. Brand, and L. McMillan, "A data-driven reflectance model," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 759–769, Jul. 2003.

[120] K. Nishino, "Directional statistics brdf model," in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 476–483.

[121] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *IEEE ECCV*, 2016.

[122] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[123] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[124] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. on Image Proc.*, vol. 13, no. 4, pp. 600–612, 2004.

[125] D. Marr, "Vision: A computational approach," 1982.

[126] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[127] B. Zeisl, M. Pollefeys *et al.*, "Discriminatively trained dense surface normal estimation," in *European conference on computer vision*. Springer, 2014, pp. 468–484.

[128] X. Wang, D. Fouhey, and A. Gupta, "Designing deep networks for surface normal estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 539–547.

[129] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski, "Towards internet-scale multi-view stereo," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1434–1441.

[130] H. Su, Q. Huang, N. J. Mitra, Y. Li, and L. Guibas, "Estimating image depth using shape collections," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 37, 2014.

[131] J. T. Barron and J. Malik, "Shape, illumination, and reflectance from shading," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 8, pp. 1670–1687, 2015.

[132] G. Oxholm and K. Nishino, "Shape and reflectance from natural illumination," in *European Conference on Computer Vision*. Springer, 2012, pp. 528–541.

[133] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser, "Physically-based rendering for indoor scene understanding using convolutional neural networks," *arXiv preprint arXiv:1612.07429*, 2016.

[134] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[135] T. Culver, J. Keyser, and D. Manocha, "Exact computation of the medial axis of a polyhedron," *Computer Aided Geometric Design*, vol. 21, no. 1, pp. 65–98, 2004.

[136] F. Aurenhammer, "Voronoi diagrams: A survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, Sep. 1991.

[137] M. Foskey, M. Garber, M. Lin, and D. Manocha, "A voronoi-based hybrid motion planner," in *Proc. IEEE/RSJ International Conf. on Intelligent Robots and Systems (IROS 2001)*, 2001.

[138] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha, "Interactive motion planning

using hardware-accelerated computation of generalized voronoi diagrams," in *Proceedings of the 2000 International Conference on Robotics and Automation (ICRA 2000)*, 2000.

[139] C. Holleman and L. E. Kavraki, "A framework for using the workspace medial axis in prm planners," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 1408–1413.

[140] N. Amenta, M. Bern, and M. Kamvysselis, "A new Voronoi-based surface reconstruction algorithm," in *Proc. SIGGRAPH '98*, ser. Computer Graphics Proceedings, Annual Conference Series, Jul. 1998, pp. 415–421.

[141] H. Choset and J. Burdick, "Sensor-based exploration: The hierarchial generalized voronoi graph," *Int. Journal of Robotics Research*, vol. 19, no. 2, pp. 96–125, 2000.

[142] C. J. Ong, S. S. Keerthi, E. Huang, and E. G. Gilbert, "Equidistance diagram — a new roadmap for path plannning," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1999, pp. 682–687.

[143] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[144] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of machine learning research*, vol. 2, no. Nov, pp. 45–66, 2001.

[145] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[146] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using gmm supervectors for speaker verification," *IEEE signal processing letters*, vol. 13, no. 5, pp. 308–311, 2006.

[147] J. Pan, X. Zhang, and D. Manocha, "Efficient penetration depth approximation using active learning," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 191:1–191:12, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508363.2508305

[148] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[149] T. Joachims, T. Finley, and C.-N. Yu, "Cutting-plane training of structural svms," *Machine Learning*, vol. 77, no. 1, pp. 27–59, 2009.

[150] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[151] M. C. Lin and D. Manocha, "Collision and proximity queries," 2003.

Curriculum Vitae

Guilin Liu received his Bachelor's Degree of Engineering from Wuhan University in 2012. His research interests include geometry & graphics, computer vision, applied machine learning etc. He has worked in Toyota Technological Institute at Chicago and Adobe Research as a intern in 2015 and 2016 respectively.

List of Publications

1. Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. "Material editing using a physically based rendering network," *International Conference on Computer Vision (ICCV).* Venice, Italy, Oct. 2017.

2. Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. "Nearly convex segmentation of polyhedra through convex ridge separation," In: *Symposium on Solid & Physical Modeling (SPM); also appears in Journal of Computer-Aided Design.* Berlin, Germany, June 2016.

3. Guilin Liu, Yotam Gingold, and Jyh-Ming Lien. "Continuous visibility feature," In: *28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Boston, MA: IEEE, June 2015.

4. Guilin Liu and Jyh-Ming Lien. "Fast medial axis approximation via max-margin pushing," In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Hamburg, Germany, Sept. 2015.

5. Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. "Dual-space decomposition of 2d complex shapes," In: *27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Columbus, OH: IEEE, June 2014.