STYLE-DRIVEN SHAPE ANALYSIS AND SYNTHESIS

A Dissertation Presented

by

ZHAOLIANG LUN

Submitted to the Graduate School of the

University of Massachusetts Amherst in partial fulfillment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2017

College of Information and Computer Sciences

STYLE-DRIVEN SHAPE ANALYSIS AND SYNTHESIS

A Dissertation Presented

by

ZHAOLIANG LUN

Approved as to style and content by:

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Evangelos Kalogerakis, for guiding me through my PhD study, advising me on various projects that pave the way for my dissertation, teaching me how to do research in a scientific way, how to coordinate with different projects and different collaborators, how to show off my work to different audience, and many more during my PhD life. I would also like to thank Prof. Rui Wang for getting me started in my graduate life, helping me for my first project, first internship, first conference, and providing advice on my research and life. I also thank Prof. Subhransu Maji as my collaborator and committee member, who is always energetic and enthusiastic and full of creative ideas during our collaboration. I thank Prof. Benjamin Jones for willing to become my committee member and offering cross-disciplinary suggestions for my dissertation as an expert. I would also like to thank everyone who gave me advices, suggestions or comments during my dissertation writing. I acknowledge support from the NSF grants CHS-1422441 and CHS-1617333.

I am lucky to collaborate with different people, including Alla Sheffer, Richard Zhang, Yahan Zhou, Haibin Huang, Changqing Zou, Matheus Abrantes Gadelha, with whom I got involved in cutting-edge research in this area and from whom I learn a lot in bringing up sparkling ideas, realizing rough concepts, tackling tough problems and summarizing fragmented pieces into publications. I also thank R. Manmatha for being a reader for my synthesis project. I thank Ren Liu and Brad Ek for mentoring me during my internship at Robert Bosch Research and Autodesk Inc. I thank Cheng Jin as my undergraduate advisor who opened the door to the graphics world for me and recommended me to join UMass.

During my six year of graduate life, I met many interesting people who brighten up my days. Many thanks to my past and current lab mates in the vision and graphics lab who built a warm, comfortable and delightful environment and offered daily laughings around this corner of the building. To all my buddies in or outside the college, with whom we had a wonderful time enjoying the beautiful life in the shiny and snowy New England. To all my friends during my internship bringing me endless joys, with whom I believe I will have intersection again someday in my future career life. To all my friends across the continent and the ocean who share every tiny piece of happiness to me no matter when and where.

Finally, I would like to give my greatest thanks to my family, especially my parents. For my toughest days, they are the ones who support me, relieve me and motivate me. My parents gave me everything. I dedicate this dissertation to my parents.

ABSTRACT

STYLE-DRIVEN SHAPE ANALYSIS AND SYNTHESIS

SEPTEMBER 2017

ZHAOLIANG LUN

B.Sc., FUDAN UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Evangelos Kalogerakis

In this dissertation I will investigate algorithms that analyze stylistic properties of 3D shapes and automatically synthesize shapes given style specifications. I will start by introducing a structure-transcending method for style similarity evaluation between 3D shapes. Inspired by observations about style similarity in art history literature, we propose an algorithmically computed style similarity measure which identifies style related elements on the analyzed models and collates element-level geometric similarity measurements into an object-level style measure consistent with human perception. To achieve this consistency we employ crowdsourcing to learn the relative perceptual importance of a range of elementary shape distances and other parameters used in our measurement from participant answers to cross-structure style similarity queries. I will then describe an algorithm that utilizes this learned style similarity measure to synthesize 3D models of man-made shapes. The algorithm combines user-specified style, described via an exemplar shape, and functionality, encoded by a functionally different target shape. We transfer the exemplar style to the target via a sequence of compatible element-level operations where the compatibility is a learned metric that estimates the impact of each operation on the edited shape. We use this metric to cast style transfer as a tabu search, which incrementally updates the target shape using compatible operations, progressively increasing its style similarity to the exemplar while strictly maintaining its functionality at each step. Finally I will propose a method for reconstructing 3D shapes following style aspects of given 2D drawings. Our method takes line drawings as input and converts them into surface depth and normal maps from several output viewpoints via a deep convolutional neural network with multi-view encoder-decoder architecture. The multi-view maps are then consolidated into a dense coherent 3D point cloud by solving an optimization problem that fuses depth and normal information across all output viewpoints. The output point cloud is then converted into a polygon mesh representation, which is further fine-tuned to match the input sketch more precisely.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

style *n. a distinctive appearance, typically determined by the principles according to which something is designed.*

— Oxford English Dictionary

1.1 Motivation

Style is an important property in object design and scene arrangement. Putting style coordination into consideration, e.g. setting up a Baroque style bed alongside with a Baroque style nightstand in a bedroom, can greatly increase the aesthetics of the environment, as well as improving the believability in a virtual environment. However, the definition of *style* is intentionally vague in terms of shape criteria in graphics literature, placing a challenge in codifying the concept of style in shape design.

In spite of the abstract nature of shape style, humans have an innate perception of shape style similarity. On one hand, humans can easily recognize style discrepancies within an arrangement of objects, e.g. identifying a stylistically incompatible cup among a tea set. On the other hand, human perception of style similarity transcends structure and function: we can meaningfully discuss style similarity between a cup and a coffee pot, a bed and a dresser, or a church and a castle. This inspires us the possibility of a machine learning approach to understand perceptual shape style similarity. Specifically, we would like to learn a metric to measure style similarity between structurally different models and detect models which share a similar style despite large functional differences. With the increasing amount of stylized 3D models available in online shape repositories, a robust style similarity measure can greatly facilitate the navigation through those shape databases. While previous work focused on evaluating style similarity between objects with similar overall structure, in this dissertation we introduce the first structure-transcending method for style similarity evaluation between 3D shapes.

A robust measure evaluating style similarity between structurally and functionally different models enables shape retrieval tasks in terms of shape style within a broad shape category. A more general use case scenario of shape style similarity is to populate virtual 3D scenes with stylistic coherent sets of objects. Unfortunately, while large databases of 3D man-made objects already exist, sets of functionally diverse shapes in a particular style are often not available. Manually creating shapes satisfying specific functional and stylistic requirements is time consuming and expertise intensive. Instead, we need an automatic method to synthesize shapes in different functionalities with style specifications. In this dissertation, we also describe a shape synthesis algorithm that transfers the style of an *exemplar* shape to structurally and functionally different *target* objects while maintaining target functionality. Our framework is inspired by the typical workflow for designing style coordinated environment: since all shapes in the environment share a coherent style, artists tend to first design one shape satisfying the style requirements and then use it as a reference to design other shapes in different functionalities. Following the idea of this workflow, our framework only requires users to locate or create a single *exemplar* shape as input, then our framework will automatically apply an algorithmic, cross-functional, exemplar-to-target style transfer to synthesize shapes in diverse functionalities while having a coherent style. This framework can significantly simplify and speed up the modeling of style coordinated virtual environments.

The style transfer approach can populate the virtual environment or object collections with stylistically coherent models. Nonetheless, all shapes being populated are constrained to an existing style; the artists still need to design the style for the exemplar shape in the first place. Furthermore, if there is a new requirement for style, the artists also need to modify the shapes with the new specifications. All these scenarios require a method to realize the style requirement rather than transferring the style. Going deeper in the design cycle, artists often prototype their style design with line drawings on common structure templates, and then translate these conceptual line drawings into shape designs. This motivates us to use line drawings as an intuitive medium to express style specification. The benefits of using line drawings as abstract representations of form and style of a shape are twofold: on one hand, line drawings convey important characteristics of the underlying shape such as its figure-ground boundaries, surface curvature, and occlusions [63, 118, 80]; on the other hand, sketching requires much less expertise comparing to the low-level 3D geometric operations required in modern 3D modeling software. The main challenge lies in converting 2D sketches into plausible 3D models. The fact that artists tend to prototype with line drawings in canonical views makes this problem more attainable since most style-related salient details are often preserved in canonical views. Moreover, the huge amount of 3D shapes available online enables a data-driven approach to infer the shape structure from unseen view projections. In this dissertation we also propose a learning approach utilizing the neural network architecture and post-processing optimization to infer a 3D shape that is consistent with input sketches from one or more views of an object.

1.2 Contributions

This dissertation proposes two novel research directions. The first research direction involves algorithmic understanding of geometric shape style and extending this understanding to applications that require evaluation of style similarity between shapes, and transfer of geometric style across objects. The second research direction aims to algorithmically extract style specifications from a more abstract domain, i.e. line drawings, and translate it into shape geometry.

In the context of these two novel research directions, I proposed three new methods with the following contributions:

- a new algorithm for evaluating stylistic similarity, which is well aligned with the human perception of style, is motivated by art history literature, and is learned from and validated against crowdsourced data
- a new algorithm for transferring style between models of man-made objects with different structure and functionality
- a deep learning architecture for reconstructing 3D shapes from 2D sketches

I will now present an overview of these new methods and the structure of this dissertation.

1.2.1 Learning Shape Style Similarity

In Chapter 3 we introduce the first structure-transcending style similarity measure and validate it to be well aligned with human perception of stylistic similarity. Our work is motivated by observations about human perception of style in art history and appraisal literature. Art history experts often classify objects as belonging to a particular geographic or temporal style by looking at salient geometric elements on the objects with recurring visual motifs [86, 9]. For instance, classical Byzantine churches are likely to have rounded domes and arches, while Gothic structures are dominated by steep gables and flying buttresses (Figure 1.1).

Our style similarity metric is therefore designed around the presence of pairs of similarly shaped, or matching, salient geometric elements on the evaluated models. Since style has no unified quantifiable, objective, definition, we resorted to crowdsourcing and followed a

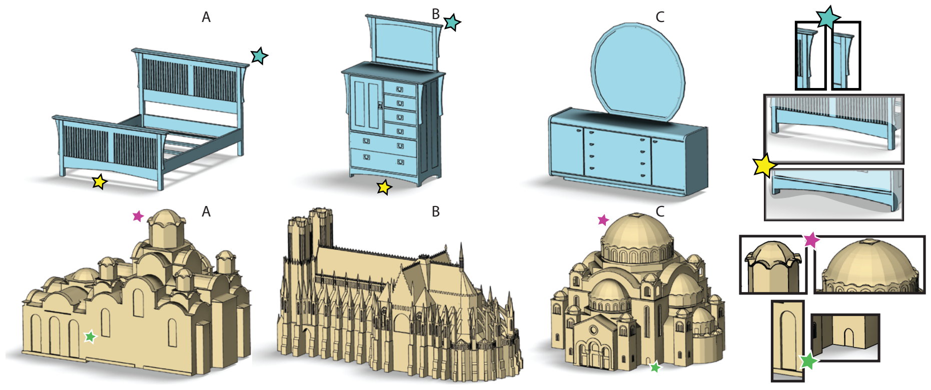


Figure 1.1. Style similarity transcends structure: in the top row, the bed A is pronouncedly more similar, style-wise, to dresser B than C; in the bottom row, building A and C are stylistically more similar (insets highlight some stylistically similar elements).

machine learning approach to quantify those geometric criteria inspired by art history literature and learn all the parameters in our measure. In collecting crowdsourced data, instead of asking participants for an absolute style similarity score which may lead to uncalibrated responses, we gathered participants responses on queries about *relative* style similarity comparisons, which are validated to be largely consistent in the evaluation.

1.2.2 Learning Shape Style Transfer

In Chapter 4 we describe the first algorithm for synthesizing shapes by transferring style between man-made objects with different structure and functionality. The underlying challenge in style transfer task is to implicitly separate style from function: while the output should stylistically look similar to the exemplar, it should also fully retain the functionality of the target. Inspired by the observations from understanding shape style similarity in Chapter 3, stylistically more similar shapes tend to have more geometrically similar elements and fewer unshared decorative features. Therefore, we search for a sequence of element-level operations (e.g. substitution, addition, removal and deformation) to gradually bring the target shape stylistically more similar towards the exemplar shape. It is impractical to exhaustively search for all possible combinations of element-level opera-

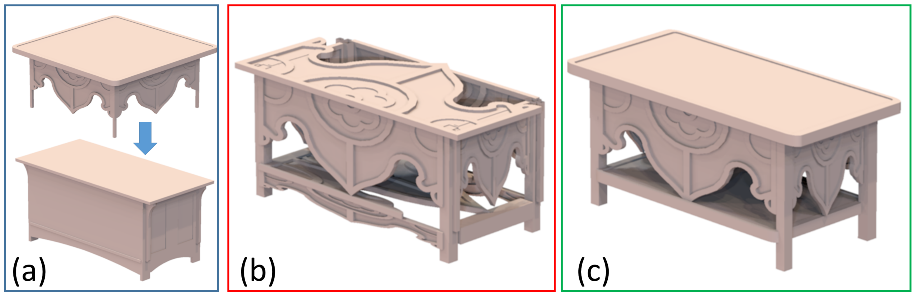


Figure 1.2. Transferring a style from a table to a TV stand (a) without (b) and with (c) functionality constraints in place.

tions that can improve the style similarity. Moreover, not all element-level operations lead to functionally compatible results (Figure 1.2). To this end, we introduce a new functional compatibility metric, which is also the main contribution of this work, to supervise the search over *compatible* element-level operations maintaining the original functionality of the output model. Our compatibility metric is inspired by design literature [85] and recent research [78] suggesting that gross form and arrangement of elements within a shape are strongly correlated to its functionality. Therefore we design our compatibility measure by leveraging the overall shape and context properties between elements. We employ a learning based approach to derive robust metric parameters using automatically generated training datasets following the observation that sets of coordinated same style shapes in online shape databases can provide insights on element compatibility.

1.2.3 Learning Shape Reconstruction from Stylized Drawings

In Chapter 5 we propose a new method for reconstructing 3D shapes from 2D sketches in the form of line drawings. Our approach appears to be the first that considers a learned, view-based representation for generating 3D shapes from sketches. The advantages of our approach are threefold. First, our learning model is based on a Convolutional Network trained to map sketches to 3D shape information, which is a powerful tool in modeling geometry and viewpoint transformation. Second, the view-based representation allows us to process 3D shape information such as depth and normals at a higher resolution than a voxel-based representation. Third, we carefully design the full optimization pipeline so that those multi-view shape information can be fused in a consistent manner which is a crucial step for getting plausible shape reconstruction results. We train our architecture on automatically generated, synthetic sketches of 3D shapes without requiring supervision in the form of human line drawings. Once trained, our method can generalize to reconstruct 3D shapes from human line drawings that can be approximate, noisy and not perfectly consistent across different viewing angles.

CHAPTER 2

RELATED WORK

In this chapter we discuss the most relevant prior works. In Section 2.1 we review literature on style analysis of shapes. In Section 2.2 we discuss prior methods on shape style transfer. In Section 2.3 we discuss related work on modeling 3D shapes from sketch drawings.

2.1 Shape Style Analysis

The most relevant literary source for understanding human perception of 3D object style can be found in art history and appraisal literature. These texts discuss at length the geometric features of architectural structures, furniture and other artifacts associated with particular historic or geographic styles, e.g. [86, 9, 73], and frequently refer to characteristic "elements of design" or "motifs" to describe a particular style. For example, [9] states "starting from recognizing motifs you will soon recognize styles", "The purpose of this brief guide is to provide photographic illustrations of ... architectural details, elements, and forms to enable the user to ... recognize styles and elements". The book proceeds to describe a range of architectural styles based on the choice of architectural elements they employ, e.g. mansards, towers, or porches, as well as the characteristic shape of different building parts such as roofs or windows. Nutting [86] similarly catalogs European and American furniture styles based on the shape of different furniture elements such as feet, trims, or posts.

The style definitions employed in this literature are descriptive rather than constructive, motivating our search for a constructive style similarity measure. Our work on shape style similarity measure builds upon previous methods for shape style analysis, as well as methods for learning style measures in other types of data, discussed below.

2.1.1 Style analysis for same class models

A range of methods provide strategies for evaluating fine-grained similarity between shapes with similar structure [126, 56, 61, 46, 116, 132]. These methods rely on the shared structure to first extract either a dense correspondence, e.g. [61, 46] or a segmentation of the two models into corresponding, compatible, parts (a co-segmentation), e.g. [126, 56]. They then use these correspondences to evaluate fine-grained similarity measuring either point-wise or part-wise geometric differences with respect to pre-defined distance metrics whose relative weights are either hard-coded or learned from database distribution. For instance, Xu et al. [126] co-segment models into roughly corresponding parts and define the style distance between shapes based on differences in scales and orientations of part bounding boxes. Kalogerakis et al. [56] define object and part styles using dominant modes of a learned probability distribution across a database of models on a range of geometric descriptors. Kim et al. [61] and Huang et al. [46] classify shapes within the same class, e.g. chairs, as belonging to different fine-grained categories, e.g. office or rocking chairs. Kim et al. perform the categorization by first producing a set of probabilistic part-based templates and grouping the shapes based on the template they fit best. Huang et al. group the shapes based on partial and local similarity measured using a combination of spin images, distance, and deformation fields, with the importance of each term learned from database distribution. Yumer et al. [132] use co-analysis of shapes within the same class to learn geometric and spatial constraints among the different parts of an object, and use this information for style transfer and other applications.

2.1.2 Structure-transcending shape style analysis

So far, little has been done when it comes to analyzing style across different structures. Li et al. [74] highlight the difficulty in evaluating or defining structure-transcending styles for 3D objects. Thus, rather than considering style of 3D shapes, they focus on identifying styles on closed 2D curves. They segment the curves at curvature extrema, and evaluate style similarity between curves by comparing segment shapes and curvature histograms. Their conclusions highlight the need to perform a perception study to establish a style definition consistent with human intuition.

More recent works aim to address the much more challenging question of analyzing style of structurally different 3D objects. Liu et al. [78] propose a metric for stylistic compatibility between furniture learned from collections of 3D scenes. They observe that the functionality of objects is strongly correlated to the gross shape and arrangement of their major parts, while style is strongly linked to the fine geometric details of these parts. Lim et al. [75] propose a deep metric learning approach for style classification across different structures and functionalities. Instead of utilizing hand-crafted geometric descriptors, they represent shapes as rendered images in multiple views and train the neural network to identify high-level features for discriminating styles. However they treat the shape as a whole which potentially misses finer-detail features within sub-elements of the shapes. Instead, our work is driven by observations in art history literature [86, 73] that point to the presence of similarly shaped, salient, geometric elements across analyzed shapes as a key indicator of stylistic similarity.

2.1.3 Learning style in other domains

There is a significant effort to analyze style in other types of data, such as images, video and audio. Tenenbaum and Freeman [112] discuss ways to separate content and style factors in speech, typography, and face images. Significant effort has been made in learning style parameters from exemplar images and video and transferring them to other instances [38, 11]. Researchers have addressed style analysis, recognition and retrieval in 2D images [121, 49], music [4], and film [7]. Doersch et al [23] recognize the visual motifs, or style elements, that distinguish photos taken in different cities.

We differ from these works in the domain of application as well as in the use of crowd-sourced data to both facilitate and validate our style similarity measure. Our work is closer to that of Garces et al. [32], who employ crowdsourcing to learn a similarity measure for clip art styles. However, we target 3D shapes which require a very different style definition and a distinct measurement approach.

2.2 Shape Style Transfer

Besides the insights from previous methods on shape style analysis, our work on shape style transfer is also built upon previous work in part correspondence, part-based shape synthesis, and style transfer.

2.2.1 Part Correspondence

Numerous existing methods compute correspondences between compatible parts of objects within the same class or co-segment such objects into such corresponding parts, see [117, 125] for recent surveys. Recent methods leverage structural similarities between shapes to extract functional part correspondences. Zheng et al. [136] match specific types of shape substructures (called SFARR) that have the form of part triplets: two symmetric parts and a third part connecting the two. Such special substructures do not exist in many man-made shape categories (e.g., lamps, cutlery) and span only a small subset of compatible parts on others. Liu et al. [77] extract common substructures on shapes through manual annotation of corresponding parts. Our method instead uses more general structural relationships to measure functional compatibility between elements and does not require any user interaction.

Laga et al. [67] use graph kernels to evaluate functional part correspondences between shapes withing the same class. Our work adopts some ideas from this work, and in particular the use of graph kernels to measure element compatibility. As shown by our comparisons to [67], applying graph kernels successfully to evaluate element compatibility on structurally different shapes requires different geometric feature sets, different encoding of graph edge relationships, and different parameter estimation. In particular, instead of hand-tuning graph kernels, we employ a learning approach to automatically adapt weights of appropriate feature descriptors and graph walk parameters impacting cross-functional compatibility.

A number of recent works compute functional correspondences between points, parts, or fuzzy regions on different shapes by analyzing their potential interactions with either other objects in the scene [43, 42], or with posed human avatars [52, 60, 96, 140, 97]. The detected interactions relate parts with similar gross functionality across objects within the same broad shape class and with sufficient input can be extended to handle patch correspondences with similar interaction types across shapes with different functionality [42]. However, such correspondences can only be estimated for patches with specific types of object interactions and require large amounts of external context data. Consequently it is likely unrealistic to extend these methods beyond detecting gross functional part correspondences. Our work uses readily available coordinated object scenes as the only training input, and computes fine-level element compatibility, necessary for synthesis of detailed functional geometric shapes in a given style. We demonstrate that compatibility can be successfully evaluated without explicit functionality detection.

2.2.2 Part-based shape synthesis

Interactive methods for part-based shape synthesis rely on users to specify and edit parts to assemble a shape, either directly [29, 66] or indirectly through semantic handles, attributes and suggestion mechanisms [14, 133], and to explicitly control output style and function.

Shape grammars are used to generate new models, either by repeating structural patterns specified manually by the user, or through automatic inference of such patterns from a set of examples [10, 107]. The shape structure and functionality are determined by the grammar, which operates within a particular shape class, or sub-class.

Multiple methods generate new shapes by combining parts from objects within the same functional class [56, 44, 127, 48], employing part correspondences generated via co-segmentation and labeling methods designed to operate on shapes with common functionality and coarse structure. After assembling models using co-segmented same class-shapes as input, Huang et al. [48] subsequently deform them to best fit input images. [132] facilitates structure preserving shape deformation by providing users with deformation handles trained on co-segmented shapes within a class. All these methods rely on co-segmentation or correspondences between parts of objects within a single class. Our framework is designed for transferring element style between shapes in vastly different classes, requiring a cross-functional element compatibility measure and does not require any prior co-segmentation or part labeling.

2.2.3 Style transfer

Researchers have explored style transfer for 2D curves, e.g. [39, 74] and images [38]. While insights from these frameworks are useful for understanding the conceptual notion of style transfer, as explained by [126, 79], algorithms developed for the 2D setting cannot be readily extended to 3D space.

There had been no attempts to address generic style transfer for 3D shapes. However two recent methods address special cases of this problem [126, 79]. Given a pre-defined coarse segmentation of the exemplar and target shapes into corresponding parts, Xu et al. [126] use these correspondences to anisotropically scale target parts to fit the proportions of the matching parts on the exemplar. The method has limited applicability, as it assumes a meaningful part level correspondence between the exemplar and target, and cannot handle style properties beyond scale.

Following [38, 39], Ma et al. [79] require a triplet of inputs: an exemplar, a source, and a target, where the source and target are expected to share the same style but have different functionality, while the source and exemplar are expected to have the same functionality and structure. They assemble the output by combining exemplar and target surface patches guided by a combination of a dense exemplar to source mapping and a piece-wise similarity transformation between the source and target. Their method makes a number of strong assumptions, that rarely hold even when a source model which fits the generic criteria above is available. For the transfer to be successful, they implicitly assume that decorative elements on the exemplar and source are co-located, and assume most target surfaces to have meaningful source counterparts, related via simple similarity transformations (target surfaces with no source counterparts are left untouched by the transfer). Our style transfer framework has none of these limitations: it does not require a source or a compatible segmentation, and can handle a far wider range of inputs than either of the methods above.

2.3 Sketch-based Shape Modeling

There has been lots of researches focus on sketch-based shape modeling (see [88] and [22] for a survey on the sketch-based modeling methods and systems). Below we discuss some of the related work addressing the problem of shape modeling using sketches as input.

2.3.1 3D geometric inference from line drawings

Compared to using natural images, estimating 3D shape from line drawings is considerably more challenging due to the lack of shading or texture information. Early works [118, 80, 76, 134] formulate the process of inferring a 3D shape based on reasoning about local geometric properties, such as convexity, parallelism, orthogonality and discontinuity, implied by lines and their intersections ("junctions"), to find a globally consistent shape. These approaches produce reasonable geometry when applied to specific families of polyhedral objects, but are less effective for organic shapes with smoothly varying surfaces. For smooth shapes, hand-designed rules are usually devised to extrude or elevate a 3D surface from contours [50, 88]. More recent methods enable the creation of freeform surfaces by exploiting geometric constraints present in specific types of line drawings, such as polyhedral scaffolds, cross-section lines and curvature flow lines [99, 124, 90]. All these methods derive geometric constraints from specific types of lines, require very accurate input drawings, and can only reconstruct what is drawn.

On the other hand, various studies [64, 19] showed that humans can consistently interpret 3D shapes from sparse and approximate line drawings (up to a *bas-relief* transformation [6]). Although the exact mechanism of 3D shape perception in humans is not well understood, this indicates that pure geometric-based methods may not be able to mimic the human ability of shape understanding from sketches.

2.3.2 Learning-based methods for shape synthesis

In contrast to pure geometric methods, learning-based approaches argue that shape interpretation is fundamentally a learning problem, otherwise it is highly under-constrained. A large number of learning-based methods have focused on estimating 3D shapes from single, natural images that include color and texture. Early work was based on analyzing shading and texture cues within image regions [40, 98], while more recent work has employed ConvNets for predicting surface depth and normals from real images [25, 120]. Driven by the success of *encoder-decoder* architectures [69, 135, 54, 115, 51] that can effectively map inputs from one domain to another, newer methods use such architectures with convolutions in three dimensions to generate 3D shapes in a voxelized representation [122, 17, 129]. A different line of work has employed ConvNets to model geometric transformations of an object to predict novel viewpoints [24, 109, 130, 138]. The approach of Tatarchenko *et al*. [110] is most related to ours. Their approach takes as input a single natural image and a viewpoint and uses a ConvNet to predict the color and depth from the provided viewpoint. They show compelling 3D reconstructions for chairs and cars from a single color image by projecting the depth maps from multiple views into a 3D space. Our approach is inspired by this work, but differs in a number of ways. Our method operates on line drawings, a more challenging type of input due to the lack of shading or color information. It predicts both normals and depth across multiple viewpoints, which are then integrated into a high-quality surface mesh representation through a joint optimization procedure. It also adapts a U-net architecture [51] along with multi-view decoder branches and a structured loss function to resolve ambiguities in the input line drawing.

2.3.3 Sketch-based 3D shape retrieval

Sketch-based retrieval methods typically transform features of the input sketch and 3D shapes into a common space where comparisons can be made. Early work was based on hand-engineered descriptors [30, 92, 41, 70, 26, 128, 123, 100, 37], while more recently, ConvNets have been proposed to learn powerful representations for sketch-based retrieval [105, 119]. Unfortunately, these methods only allow retrieval of existing 3D shapes or parts. They provide no means to synthesize novel shapes or parts from scratch. A few recent approaches employ category-specific, predefined *parametric models* to guide shape reconstruction through ConvNets [84, 45]. These methods are only able to recover specific shape parameters or rules from input sketches. If a drawing depicts a shape that cannot be described by the parameters of these models, then the reconstruction fails. In contrast, our method learns a representation capable of predicting shapes from sketches without any predefined parametric model. We expect 3D shape priors to automatically emerge in our deep network.

CHAPTER 3

LEARNING PERCEPTUAL SHAPE STYLE SIMILARITY

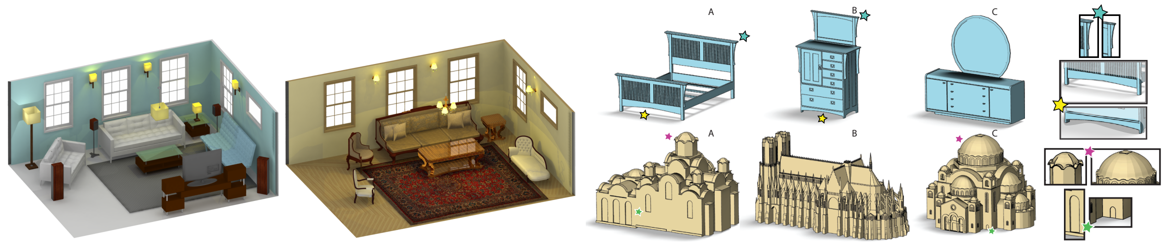


Figure 3.1. (left) Changing the style of objects in a scene influences the sense of time and place. (right) Style similarity transcends structure: in the top row, the bed A is pronouncedly more similar, style-wise, to dresser B than C; in the bottom row, buildings A and C are stylistically more similar (insets highlight some stylistically similar elements).

The human perception of stylistic similarity transcends structure and function. An algorithmically computed style similarity measure that mimics human perception can benefit a range of computer graphics applications. Previous work in style analysis focused on shapes within the same class, and leveraged structural similarity between these shapes to facilitate analysis. In contrast, we introduce a *structure-transcending* style similarity measure and validate it to be well aligned with human perception of stylistic similarity. Our measure is inspired by observations about style similarity in art history literature, which point to the presence of similarly shaped, salient, *geometric elements* as one of the key indicators of stylistic similarity. We translate these observations into an algorithmic measure by first quantifying the geometric properties that make humans perceive geometric

The work described in this chapter has been published as a full paper in *ACM SIGGRAPH 2015*. Please also see the accompanying video: `https://youtu.be/PWqZwpHQtnE`

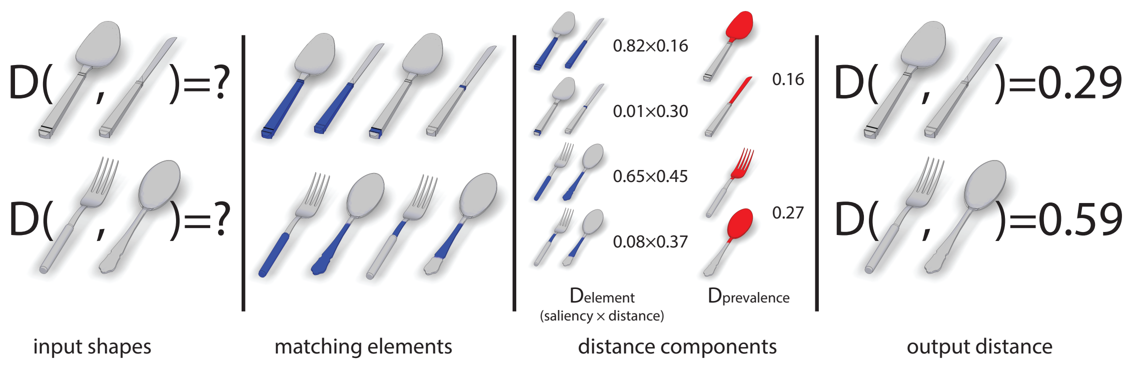


Figure 3.2. To evaluate style similarity, we identify potentially matching elements; and use those in a distance function that accounts for element similarity, element saliency and prevalence. The parameters of both steps are learned from crowdsourced perceptual similarity data.

elements as similarly shaped and salient in the context of style, then employing this quantification to detect pairs of matching style related elements on the analyzed models, and finally collating the element-level geometric similarity measurements into an object-level style measure consistent with human perception. To achieve this consistency we employ crowdsourcing to quantify the different components of our measure; we learn the relative perceptual importance of a range of elementary shape distances and other parameters used in our measurement from 50K responses to cross-structure style similarity queries provided by over 2500 participants. We train and validate our method on this dataset, showing it to successfully predict relative style similarity with near 90% accuracy based on 10-fold cross-validation.

3.1 Overview

Our goal is to obtain a structure-transcending style similarity measure for man-made 3D shapes (Figure 3.2).

While the notion of style extends beyond shape, we consider a purely geometry based measure; for most modeling applications properties such as texture can be easily changed once a shape is available. Moreover shape databases frequently contain only geometric information, making a measure which contains other properties less useful. Our framework for computing a style similarity measure consists of the key components outlined below.

3.1.1 Element Similarity

We first develop a method to measure element-level similarity in the context of style evaluation. Our measure is inspired by observations in art history literature about the types of geometric criteria that play a role in style identification (Section 3.2).

3.1.2 Matching Elements

We use this measurement method within a matching algorithm that detects similar geometric elements on the evaluated objects. We first search for paired regions on the processed models that satisfy the approximate mapping requirement. We then group neighboring region pairs together based on geometric similarity, both within each pair of regions, and between adjacent regions (Section 3.2.2).

3.1.3 Combined Style Measure

We seek a measure that reflects both the degree of similarity between the detected matching elements as well as the percentage of the surface area on both models covered with similar elements.Our overall style similarity measure balances these two terms.

3.1.4 Learning

In each of the steps above we face multiple parameter choices, such as "how to weigh different elementary distances when evaluating element similarity?", "how to decide when elements are similar enough for matching purposes?", or "how to evaluate saliency in the context of style measurement?" As we aim to obtain parameter values that lead to a style measure consistent with human perception, we elect to learn these parameters by studying human responses to style similarity queries and algorithmically tuning the parameters to best mimic these responses. Our training step is based on participant responses to relative style similarity queries, which we describe next, and is designed to maximize the agreement between our measure and participant responses.

3.1.5 Study of Style Perception

Our study was designed to achieve two goals. We wanted to examine our hypothesis that human perception of style similarity between differently structured objects is consistent. We also aimed to use the study results to facilitate parameter learning for our style measurement algorithm. Our study was designed around *relative* comparisons, with users asked to evaluate if an object A is more stylistically similar to object B or C.

The selection of queries, detailed in Section 3.3, was motivated by the two goals above. We pre-processed the raw participant input for training and algorithm validation, removing queries with non-discriminative majority responses and answers from participants deemed unreliable (see Section 3.3).

3.2 Measuring Style Similarity

Man-made shapes in online databases are typically represented as partially connected meshes (polygon soups). To evaluate style similarity we densely resample these models, representing them as point clouds with normals (normal direction is set to point outward using point visibility). We assume the models to be upright oriented. Most models in online repositories, and essentially all the inputs we downloaded, satisfy this assumption. Incorrect orientation can corrected using the method of Fu et al. [28].

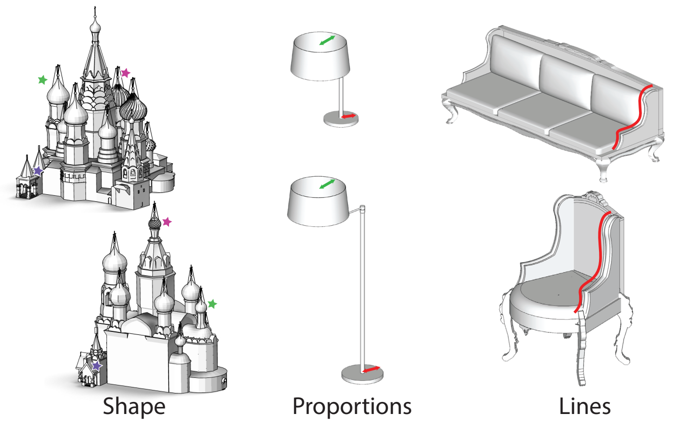


Figure 3.3. Literature highlights three element-level style similarity criteria: intrinsic element geometry or shape, relative proportions or scale, and dominant curve or line shape.

3.2.1 Geometric Similarity

Art-history literature [86, 9] and appraisal tutorials, e.g. [20], point to three separate geometric criteria that are useful when identifying a particular style and which are applicable across different structures: *shape*, *proportions*, and *lines* (Figure 3.3). This literature repeatedly stresses that objects with similar style are expected to have intrinsically similar, even if differently scaled, geometric elements - see the highlighted church domes in Figure 3.3, left or the skirts of the bed and dresser in Figure 3.1. It also indicated that relative and internal proportions of the elements play an important stylistic role - e.g. narrow vs square windows, sturdy or thin furniture legs, and so on (Figure 3.3, center). Finally, it emphasizes the importance of representative or noticeable surface curves in conveying style on the object's surface (Figure 3.3, right). Styles are often characterized by the use of straight versus curved, or clean versus ornate lines. While for some style comparisons all three criteria may come into play, it is the interaction or the relative weight of each criterion we seek to learn from training data.

We measure geometric similarity using elementary distances that relate to these criteria. When comparing intrinsic element geometry we employ both direct comparisons - measuring point-wise positional and normal distances computed after aligning the elements using an affine transformation, and indirect comparisons, measuring curvature distribution. We compare element proportions using their bounding box scales and shape diameter functions [102]. To explicitly account for line similarity we detect and compare feature curves and representative silhouettes. All distances are normalized to the interval $[0, 1]$. We detail the exact distance metrics in Appendix A.1.

We represent the distance between two elements $\{p, p'\}$ as a weighted combination of the elementary distances, using learned distance weights w_i,

$$D(p, p') = \sum_{i=1}^{F} w_i d_i(p, p'). \tag{3.1}$$

3.2.2 Extracting Matching Elements

Given a pair of input models, we need to detect elements of one model that match, or are geometrically similar, to elements on the other and vice versa. These matching elements may not share the same exact geometry, but are expected to share similar geometric features, as measured by the geometric similarity measure above. Detecting matching elements is challenging since we do not *a priori* know the size, location, or number of such elements. We make the problem tractable by observing that geometric elements are typically self-similar and can be *approximately* mapped to each other using an affine transformation. Following these observations, we first locate near-convex patches on the two models that approximately map to one another, we then locate dominant mapping transformations, and finally groups patches into elements by merging together adjacent patches that undergo a similar dominant mapping transformation. Our grouping aims to discard matched, yet dissimilar, patches and identify coherent geometric elements that share common geometric characteristics and which frequently stand apart from the surrounding surface. We now describe these steps in detail.

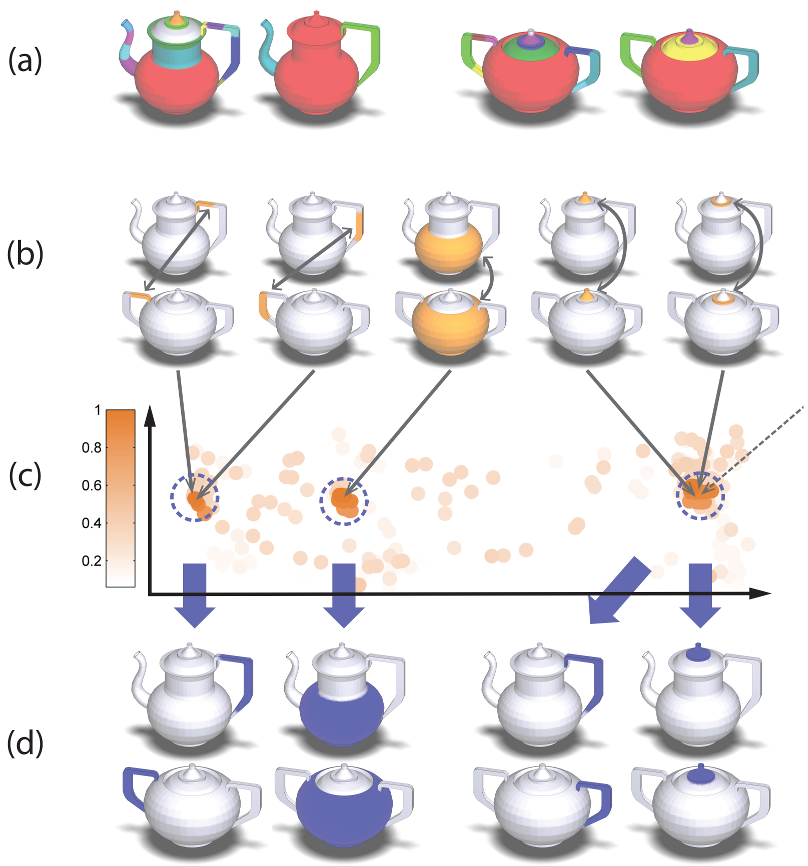


Figure 3.4. Extracting matching elements: (a) patch-segmentations (two levels visualized); (b) example matches; (c) transformation space (2D MDS projection) with clustering results; (d) extracted elements.

3.2.2.1 Patch Sampling

As a starting point for the matching process we sample each input model using a dense set of approximately convex patches, computed using the method of [3] (Figure 3.4, a). Operating on patches, instead of individual points, significantly reduces the time complexity of our element computation. Patches also provide a more reliable starting point for matching since we can immediately evaluate match quality using our geometric distance measure, assisting further analysis. We generate patches at a number of scales which enables us to detect similarly shaped, but differently scaled, elements.

3.2.2.2 Transformation Clustering

For each patch computed in the previous step, we compute a transformation that approximately maps it to every patch on the other shapevia an outlier-robust iterative closest point alignment algorithm [8]. To detect groups of adjacent patches that undergo similar transformations, we use a Hough transform based voting strategy [5, 81]. To imbue the transformation votes with geometric meaning, we represent each transformation as a point in a nine-dimensional space which consists of translation, rotation, and non-uniform scaling or reflection computed via Singular Value Decomposition. Each point is assigned a confidence weight based on the shape distance between the transformed patch p and its image p':

$$\omega = \frac{A(p) + A(p')}{2} \exp\Big(- D(p, p')/s \Big), \tag{3.2}$$

where $A(p)$ and $A(p')$ measure the percentage area of p and p' relative to their shape. The parameter s controls the confidence weight falloff as the distance increases, and is automatically estimated by using a grid search and selecting the value that maximizes the objective function (Equation 4.13) during model training (Section 3.2.4). To find the dominant transformations in the 9-dimensional voting space we perform mean-shift clustering.

Each local maximum of density yields a cluster of voting transformations, and each cluster centroid corresponds to a dominant transformation that approximately maps a number of patches of one shape to the other. The transformation computation and subsequent clustering, visualized in Figure 3.4, are performed twice, from the first shape to the second and vice versa.

3.2.2.3 Element Extraction

We use the dominant transformations $\mathcal{T}$ found in the previous step, to compute matching elements, where each element is defined as a group of contiguous patches and the matching element is defined by the image of these patches under $\mathcal{T}$. We formulate this task as a labeling problem to make similar inside/outside decisions for similar, contiguous patches. The labeling assigns each patch p a binary label c_p which is set to 1 if the patch is added to the group and 0 otherwise. We compute the labels by minimizing the following objective function over all the patch label assignments **c** per shape:

$$E(\mathbf{c};\mathcal{T}) = \sum_{p} E_1(c_p;\mathcal{T}) + \sum_{p,q} \frac{1}{|\mathcal{N}(p)| + |\mathcal{N}(q)|} E_2(c_p, c_q;\mathcal{T}) \tag{3.3}$$

where p, q are adjacent patches, $\mathcal{N}(p)$, $\mathcal{N}(q)$ are the sets of all patches adjacent to p and q respectively. The unary term in this function assesses the distance between p and its image T_p under the transformation $\mathcal{T}$, and the pairwise term assesses how likely a pair of adjacent patches p, q is to belong to the same element. Specifically, the unary term expresses the negative logarithm of the following probability for an individual patch p:

$$P(c_p = 1;\mathcal{T}) = \exp(-D(p, T_p)/s) \tag{3.4}$$

thus:

$$E_1(c_p = 1; \mathcal{T}) = D(p, T_p)/s \tag{3.5}$$

$$E_1(c_p = 0; \mathcal{T}) = -\ln\Big[1 - \exp(-D(p, T_p)/s)\Big] \tag{3.6}$$

where T_p denotes all patches on the other shape that are closest to the patch p when it is transformed under the transformation $\mathcal{T}$. We use the same learned parameter s as in the clustering step.

The pairwise term expresses the negative logarithm of the probability for pairs of neighboring patches to have different binary labels based on the geometric distance between them:

$$E_2(c_p, c_q) = -[c_p \neq c_q] \ln\Big[1 - \exp\Big(- D(p, q)/s\Big)\Big] \tag{3.7}$$

To compute the distances between the patches, we apply a translation to align their centroids. A small distance indicates that the two patches are likely to belong to the same geometric element, and that the cost for assigning different labels to them should be high. In this case the pairwise term will encourage them to be either grouped into the element associated with T_p , or have both removed from the group depending on their unary terms. We compute the labeling using the standard min-cut framework [36] for each shape. Each computation yields a group of patches on one shape that approximately map to patches on the other shape under the transformation $\mathcal{T}$ and are internally similar. We perform labeling separately for the two transformation directions (from the first shape to the second and vice versa).

3.2.3 Combined Style Similarity Measure

3.2.3.1 Element-level Similarity

Given a set $\mathcal{M}$ containing all the pairs of matching elements detected on the two input shapes, element-level similarity is computed as:

$$D_{element} = \sum_{\{p,p'\}\in\mathcal{M}} C(p,p')D(p,p') \tag{3.8}$$

where $D(p,p')$ is the distance between a pair of matching elements $\{p,p'\}$ on the two models, and $C(p,p')$ is the saliency of this pair of elements. As pointed out earlier, style elements are expected to be visually distinct, or salient, motivating the use of saliency to weigh the impact of individual element distances on the overall style similarity between shapes. We define saliency using a weighted combination of elementary saliency metrics suggested by recent literature [16, 72, 103] (see Appendix A.1). The saliency of a pair of elements is defined as the average of their individual saliences $C(p,p') = .5[C(p)+C(p')]$, and element saliency is expressed as a weighted sum of the saliences of its sample points. Specifically for the element p (and similarly for its matching element p'):

$$C(p) = \Big(\sum_{s\in p} \sigma\Big(\sum_{j=1}^{G} v_j x_{j,s} + v_0\Big)/M(s)\Big)/C(S) \tag{3.9}$$

where $\sigma(x) = 1/(1+exp(-x))$ represents the sigmoid, or logistic, function, $x_{j,s}$ are the elementary saliency metrics measured at the sample point s, v_j is a learned weight per metric, and v_0 is a learned bias weight shifting the sigmoid along the input axis. The sigmoid transformation non-linearly combines the elementary saliency metrics and scales the resulting point saliencies within the $[0,1]$ range. Our experiments (Section 3.4) show that using this formulation to combine elementary saliency metrics is more predictive than using a simple linear combination. We normalize the element saliency by the saliency integral across the entire input model S and the number $M(s)$ of matching elements the sample point s belongs to:

$$C(S) = \sum_{r\in S} \sigma\Big(\sum_{j=1}^{G} v_j x_{j,r} + v_0\Big) \tag{3.10}$$

3.2.3.2 Prevalence

To estimate the prevalence of the matching elements we consider the percentage of the area not covered by these elements on both models. For identical input shapes, this percentage will be zero, and will increase to one if no matching elements are found. As with element similarity, we take saliency into account. If the uncovered area contains salient features, it would indicate a poorer stylistic match between shapes than if it is nondescript. We penalize unmatched areas z and z' on the two objects using the saliency integral across these areas, normalized by the saliency integral across the relevant shape

$$D_{prevalence} = 0.5[C(z) + C(z')] \cdot t \tag{3.11}$$

where t is a learned penalty parameter.

3.2.3.3 Combined Distance Function

The distance between two shapes is defined as the sum of the two terms above:

$$D = D_{elements} + D_{prevalence} \tag{3.12}$$

We note that the distance between the two models is by definition symmetric. The impact of each term depends on the learned individual weights on the elementary distance and saliency metrics.

3.2.4 Parameter Learning

The input to our parameter learning step is a set of user responses to relative similarity queries based on triplets of shapes $\{A, B, C\}$. For each query we have answers from multiple participants whether the pair of objects $\{B, A\}$ is more stylistically similar than the pair $\{C, A\}$ or vice versa. The output is a set of learned parameters (total 99 parameters) for the distance function and the matching algorithm, which can then be used to compute style distances on other pairs of objects. We note that our problem setting is different from regression or classification, since our training data does not have the form of absolute, continuous or discrete, measurements of style. Instead, we use a probabilistic framework suited to handle relative comparisons for training. Since not all study participants are equally reliable in their answers, our training procedure weights each participant according to number of times they disagreed with the majority answer in each relative comparison.

3.2.4.1 Learning Distance Parameters

Our model expresses the probability a participant rates $\{B, A\}$ as more similar than $\{C, A\}$, or more compactly $B_A \triangleright C_A$ as :

$$P(B_A \triangleright C_A) = \sigma\Big(D(C, A) - D(B, A)\Big) \tag{3.13}$$

and similarly:

$$P(C_A \triangleright B_A) = \sigma\Big(D(B, A) - D(C, A)\Big) = 1 - P(B_A \triangleright C_A) \tag{3.14}$$

where $\sigma(x)$ is a sigmoid function that converts the shape distance differences into probabilities. This logistic-based probabilistic model follows [12], where it was used for learning model rankings in the context of information retrieval.

To promote sparsity of the weights, our model contains a L^1-norm regularization term which can be seen as expressing a prior probability for the weights of elementary distances and saliency features to be small:

$$P(\mathbf{w}, \mathbf{v}, t) = \exp\Big(- \lambda_1 ||\mathbf{w}||_1 - \lambda_2 ||\mathbf{v}||_1 - \lambda_3 |t|)\Big) \tag{3.15}$$

where $\mathbf{w} = \{w_i\}_{i=1...F}$, $\mathbf{v} = \{v_j\}_{j=1...G}$. The regularization parameters $\lambda_1, \lambda_2, \lambda_3$ control the degree of sparsification of the model and are automatically estimated through $10-$fold cross-validation on the training set. Given M training triplets, we learn the parameter values that maximize:

$$L(\mathbf{w}, \mathbf{v}, t) = \ln P(\mathbf{w}, \mathbf{v}, t) + \sum_{m=1}^{M} b[m] \cdot \ln P(B_A[m] \triangleright C_A[m]) + c[m] \cdot \ln P(C_A[m] \triangleright B_A[m])$$

where $b[m]$ and $c[m]$ represents our confidence that $B_A[m] \triangleright C_A[m]$ and $C_A[m] \triangleright B_A[m]$ respectively based on the user responses to the query m. The confidence per query is measured as follows. Each user is assigned a reliability weight that is equal to the percentage of times their answers agreed with the majority answer in the queries they were asked. The confidence $b[m]$ (and similarly $c[m]$) for a query m is measured as the sum of reliability weights for users that answered $B_A[m] \triangleright C_A[m]$ (or $C_A[m] \triangleright B_A[m]$) normalized by the total sum of reliability weights of the users who answered the query. We use bound constraints to enforce the parameters t and $\mathbf{w}$ to be positive.

We use Matlab's implementation of Sequential Quadratic Programming (SQP) for the optimization task of our objective function. To initialize the optimization, the weights are set to small random values. Finally, we note that all elementary distances are normalized to $[0, 1]$ during training by dividing them by their $90th$ percentile value computed across all training pairs and then truncating all higher values to 1. The percentile is used instead of the maximum to discard any outlier values in the training data.

3.2.4.2 Learning Matching Parameters

To learn the parameters of the overall distance function, we require the output of the matching step. However, our matching step requires an element-level distance measure to evaluate the shape differences between pairs of patches, creating a self-referential dependency between the two steps. To learn both sets of parameters we use an iterative procedure.

We start with a naive distance measure generated by computing the average closest point-to-point patch distance after ICP and use this measure to detect an initial set of matching elements. We then update the parameters of the distance function by training our model with the procedure above. We repeat both steps, each time using the just learned, more reliable, distance function in transformation clustering and min-cut labeling for element matching, resulting in better matches. In practice three iterations were sufficient for the method to converge to the results reported in Section 3.4.

3.3 Study of Style Perception

Category	# Shapes	# Total Queries	# (%) Q. (iii) plurality	# (%) Q. (iv) plurality	# (%) (i) & (ii) plurality	# (%) Q. majority
building	238	1000	0 (0.0%)	149 (14.9%)	798 (79.8%)	731 (73.1%)
furniture	278	1250	0 (0.0%)	134 (10.7%)	1088 (87.0%)	1065 (85.2%)
lamp	186	1250	1 (0.1%)	103 (8.2%)	1121 (89.7%)	1100 (88.0%)
column	74	800	0 (0.0%)	25 (3.1%)	760 (95.0%)	743 (92.9%)
coffee set	76	270	0 (0.0%)	32 (11.9%)	233 (86.3%)	224 (83.0%)
cutlery	74	200	3 (1.5%)	10 (5.0%)	184 (92.0%)	183 (91.5%)
dish	91	200	3 (1.5%)	18 (9.0%)	170 (85.0%)	162 (81.0%)
Total	1017	4970	7 (0.1%)	471 (9.5%)	4354 (87.6%)	4208 (84.7%)

Table 3.1. Study statistics per category. Left to right: category, number of models, number of queries, number and percent of queries with plurality “Both B and C” response, number and percent of queries with plurality “Neither B nor C” response, number and percent of queries with plurality discriminative response, number and percent of queries with a *majority* discriminative response (majority formed by more than 50% participants).

Our study tests the hypothesis that human observers are persistent and consistent in evaluating relative style similarity across structurally different objects, and provides data for training our algorithmic style similarity measure. We gathered our study data using online questionnaires released through the Amazon Mechanical Turk (MTurk) service.

Category	**% persistence** all/reliable	**% consistency** all/reliable	**% consistency** majority	**% consistency** (i) vs (ii)
building	73.8% / 76.7%	76.8% / 79.0%	86.6%	91.3%
furniture	89.5% / 90.8%	86.0% / 87.2%	91.2%	97.4%
lamp	92.5% / 93.4%	89.8% / 90.6%	94.4%	97.8%
column	86.4% / 88.8%	87.3% / 88.9%	91.7%	96.8%
coffee set	82.3% / 84.2%	83.5% / 85.0%	90.3%	94.5%
cutlery	88.1% / 89.8%	89.4% / 91.4%	93.7%	97.7%
dish	83.7% / 86.1%	78.7% / 81.6%	88.1%	92.6%
Total	85.7% / 87.7%	85.0% / 86.5%	91.3%	95.8%

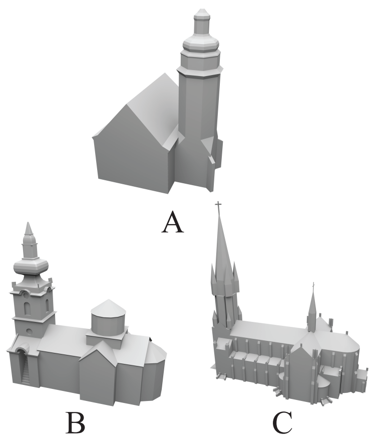
Table 3.2. More study statistics per category. Left to right: category, participant persistence across all participants and across reliable participants only, participant consistency across all participants and across reliable participants only, consistency for queries with a majority response, consistency for queries considering only discriminative responses ((i) B or (ii) C).

3.3.1 Study Format

The queries used in our questionnaires were based on triplets of models, laid out as visualized in Figure 3.5, left. Subjects were asked the question "Which of the two shapes on the bottom (B or C) is more similar, style-wise, to the shape on the top (A)?" and were required to select one of the following answers: "(i) B, (ii) C, (iii) can't tell - Both B and C, (iv) can't tell - Neither B nor C".

The models used for the study were organized into seven structurally diverse categories: buildings, furniture, lamps, coffee sets, architectural columns (pillars), cutlery and dishes. To focus on structure-transcending style similarity, for categories with clear fine-grained structural sub-categories (furniture, lamps, coffee-sets and cutlery) B and C were selected to have similar structure, different from that of A (e.g two dressers and a bed, Figure 3.1). In categories with no clear structural sub-classes, the triplets were assembled based solely on similarity bias.

Since our primary goal was to train our style similarity measure, we assembled most of the queries with the goal of obtaining *discriminative* responses, where participants clearly rank the degree of similarity between shapes, by introducing subjective bias. Specifically,



(i) B - 60%

(ii) C - 0%

(iii) Both - 0%

(iv) Neither - 40%

Figure 3.5. (left) Study query layout. (right) response distribution for this query.

we generated 120 out of 4970 queries at random to validate the observation above. For the rest of the study queries, roughly half of them were constructed such that A and one of B or C were selected from a single database scene, or arrangement, (e.g. tableware set), and the remaining object was drawn from a different arrangement. The remaining half queries were constructed such that one pair was classified by the authors as being in the same geographic or temporal style, while the third shape was subjectively classified as belonging to a different style.

3.3.2 Questionnaire and Participant Information

Each questionnaire released via the Mechanical Turk contained 25 unique queries. Each question was repeated twice, with B and C flipped, to measure participant persistence. To collect a diverse set of answers per query and avoid any individual bias, we allowed each participant to complete *only one* questionnaire per category. Participants were rewarded $\$0.50$ for each questionnaire completion.

3.3.3 Query Response Processing

Any large-scale study faces the risk of attracting unreliable respondents. We detected and discarded outlier responses from participants who gave two different answers to more than 6 out of the 25 unique queries in the questionnaire, or took less than 3 minutes to complete it. For all other participants, we ignore answers which are provided differently from the same participant. For learning purpose we only used queries with a majority discriminative ((i) B or (ii) C) response. Table 3.1 lists the number and percentage of discarded and remaining queries as well as those of queries with discriminative responses. For algorithm training and validation we detected and discarded outlier responses using a two stage filter. Participants who gave two different answers to more than 6 out of the 25 unique queries in the questionnaire, or took less than 3 minutes to complete it, were classified as unreliable and all their answers were discarded. For all other participants, we ignored non-persistent answers, where a participant answered the same question differently. To form a statistically significant majority we gathered answers to each query by 10 different, reliable users. For learning purposes we only used queries with a majority discriminative ((i) B or (ii) C) response. While the answer "(iii) Both B and C" could potentially be used in a learning procedure, the percentage of queries with such plurality answers is negligible (0.1%) and does not justify the extra effort required to incorporate them into the training algorithm. The number and percentage of discarded and remaining queries are listed in Table 3.1 columns four through six. The number and percentage of queries with discriminative majority responses used for learning are listed in Table 3.1 column seven.

3.3.4 Hypothesis Validation

We hypothesize that participants' consistency and persistence in this study can be considered as a measure of human performance for comparing the style similarity of shapes. Consistency is measured as the percentage of times that MTurk participants' answers agree with the plurality answer per query, i.e., the percentage, or size, of the plurality. Table 3.2

lists those values. The fact that on average 8.5 out of 10 users agree on the response for a query confirms that human observers are consistent in evaluating relative style.

3.3.5 Representative Sample

An interesting question to ask is how the consistency of MTurk participants compares to a curated participant set, and how their perception of style compares to that of experts. To answer these questions, prior to conducting our large-scale study, we performed a pilot study which included a mix of participants: 55 unique participants found through the MTurk service, 32 casual participants located based on personal contacts, and 5 Arts Ph.D students. The last group can be considered as experts for our task. This study had 250 queries. Across the casual user group the average plurality size was 93.5% based on all answers and 99.3% excluding the non-discriminative answers. Within the expert user group, the pluralities were very similar - 95.3% and 99.5% correspondingly. For MTurk participants in this smaller study, the pluralities were 88.6% and 98.5% correspondingly, slightly smaller and similar to the ones in our large study. We conclude that the overall consistency across the different user groups is similar, and the consistency rate we observe among MTurk participants serves as a plausible estimate of such consistency in the general population. When comparing the majority responses across all three groups taking only discriminative responses into account, the percentage of times that casual users or MTurk participants disagree with the plurality answer provided by experts were negligible, 0.6%, and 1.2% respectively. In other words when participants were able to provide a ranking these rankings were essentially identical. This observation indicates that our learning method, which relies only in discriminative majority answers of MTurk participants, is likely to be consistent with expert perception of style. We observed a larger difference in the percentage of time one group of participants chose the non-discriminative ‘Neither B nor C’ (iv) response while the other one chose a discriminative one. Overall, MTurk participants agree with expert plurality 83% of the time, while casual participants agree with expert plurality 87% of

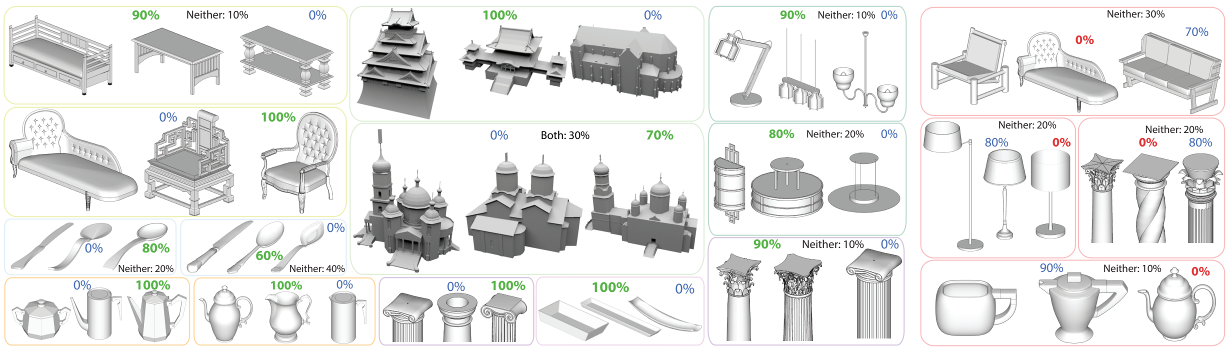


Figure 3.6. (left) Examples of the 89% of queries where our method agrees with study majority response(shared answer in green), numbers show percentages of participants who selected each answer (not listed answers received zero votes). (right) Some representative failure cases (majority response blue, ours red).

Category	**all terms**	no prevalence term	linear saliency	no saliency term	LFD
building	**81.4%**	77.4%	79.3%	79.9%	70.7%
furniture	**91.4%**	87.9%	90.8%	90.6%	74.6%
lamp	**95.0%**	86.1%	94.5%	94.7%	61.6%
column	**90.2%**	87.2%	87.8%	87.9%	55.5%
coffee set	**90.6%**	87.1%	89.3%	86.2%	62.1%
cutlery	**85.8%**	72.7%	82.5%	81.4%	61.2%
dish	**89.5%**	86.4%	87.0%	87.0%	88.9%
average	**89.1%**	83.5%	87.3%	86.8%	66.6%
mixed	**86.6%**	82.1%	86.3%	85.9%	67.8%

Table 3.3. Prediction accuracy, Left to right: category, our prediction accuracy, prediction accuracy with alternate formulations. Rows one to seven show results where training and validation were done per category, bottom row shows results where both were done on the entire database.

the time. Such discrepancy is to be expected, as experts may look for different style cues beyond those noticeable by laymen. Since the difference remains small and is limited to non-descriptive answers, we believe that the MTurk participant responses can be relied on to derive an accurate picture of human perception of relative style similarity and to train a robust style similarity measure.

3.4 Algorithm Validation

We validate our style similarity measure by performing ten-fold cross-validation on queries with a majority discriminative response among the study participants. Queries tested during validation excluded all pairs of models present in the training queries. The percentages of queries on which our algorithm agrees with the majority response are reported in Table 3.3, second column. Across all categories our method agrees with the majority response 89.1% of the time. This number is comparable to the agreement level between the individual reliable participants for these queries (91.3%). Table 3.3, rows one through seven, report the predictive accuracy for the scenario where the algorithm was trained and validated separately against each model category. Performing these two tasks on all the categories at once, the average accuracy slightly drops to 86.6% since, as one would expect, the importance of different measure components may vary across different object categories.

3.4.1 Algorithmic Choices

We evaluated a number of alternative approaches for style measurement, summarized in Table 3.3, columns three to five. We evaluated the impact of dropping the prevalence term, using linear vs sigmoid saliency models, and ignoring saliency altogether. As expected each change led to drop in prediction accuracy, with the omission of prevalence leading to the largest drop. While one could expect an even larger drop without the prevalence term, such a drop is prevented by our use of approximate element matching: we purposefully classify elements as approximately matching even if the distance between them is significant. This choice assists style similarity evaluation when the input shapes do not share identical style elements. In the last column of Table 3.3, we provide comparisons against using the popular LightField shape descriptor alone (LFD) as a distance measure between shapes [15]. Our learning method has significantly higher average prediction accuracy compared to using LFD for style similarity. As explained in Section 3.2.4, our algorithm employs an iterative scheme that alternates between matching elements using our distance function and then

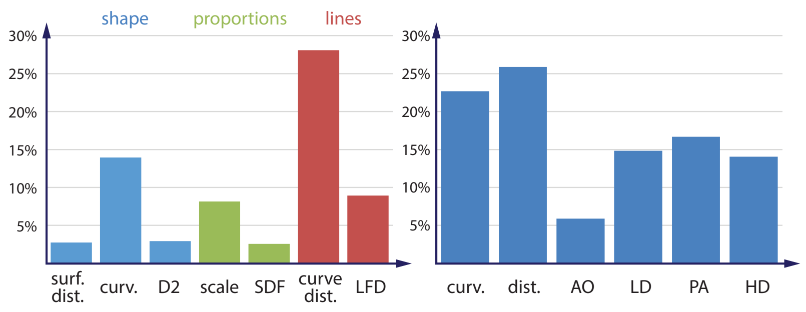


Figure 3.7. (left) Similarity weights, features from left to right are: surface distance, curvature difference, shape distribution D2 difference, scale difference, shape diameter difference, curve distance, light field descriptor difference, (right) saliency weights, features from left to right are: curvature metrics, location metrics, ambient occlusion, and three levels of shape distinctness from [103].

updating its parameters. After the initial iteration, the prediction accuracy of our algorithm averaged over our seven datasets is 87.9%. At the second iteration, the accuracy increases to 89.0%, and at the third iteration the accuracy converges to 89.1% as reported in Table 3.3.

3.4.2 Elementary Distances

Figure 3.7(left) shows the relative importance of each elementary distance as reflected by its learned weight in the element similarity term (normalized by the sum of all elementary distance weights), averaged over all our seven categories. We observe that the distances between feature curves are contributing the most to our style measure. Figure 3.7(right) similarly shows the relative importance of each saliency feature as reflected by the absolute scale of its learned weight in the saliency model (normalized by the sum of all absolute saliency weights). We note that the relative importance for saliency features is less direct since we employ a non-linear sigmoid-based model for measuring saliency. Curvature- and location-based features appeared to have higher contribution.

3.4.3 Complexity and Runtimes

Our distance computation has two main time-consuming steps: computation of per sample point geometric features used for elementary distances and saliency metrics, and element matching. Feature computation takes 40 seconds on average for a pair of shapes. Element matching consists of patch sampling, transformation clustering and element extraction steps which take 115, 85 and 35 seconds respectively for a shape pair on average. In total, evaluating the distance function takes about 4.5 minutes. We note that several parts of our algorithm could be implemented much more efficiently e.g., patch segmentation is implemented on a single thread. Regarding computational complexity, the distance function evaluation has quadratic complexity in the number of patches in shapes and linear in the number of point samples. We note that the number of patches is relatively low, ranging from 20 to 80 at most.

Optimizing the objective function for learning the parameters of our distance function requires 30 seconds per 100 training queries. The complexity of the parameter learning stage is linear in the number of triplets. Learning requires evaluating distance functions for all shapes pairs in the training queries. For our largest dataset, the learning stage requires about 50 hours. We note that the learning stage is an offline procedure; once the measure is learned, applying it for a shape pair requires only a few minutes, as discussed above. All running times are reported on an Intel E5-2697 v2 processor. Our data and source code is available on our project page: `https://people.cs.umass.edu/~zlun/papers/StyleSimilarity`.

3.5 Applications

We describe three novel applications of our learned style similarity measure.

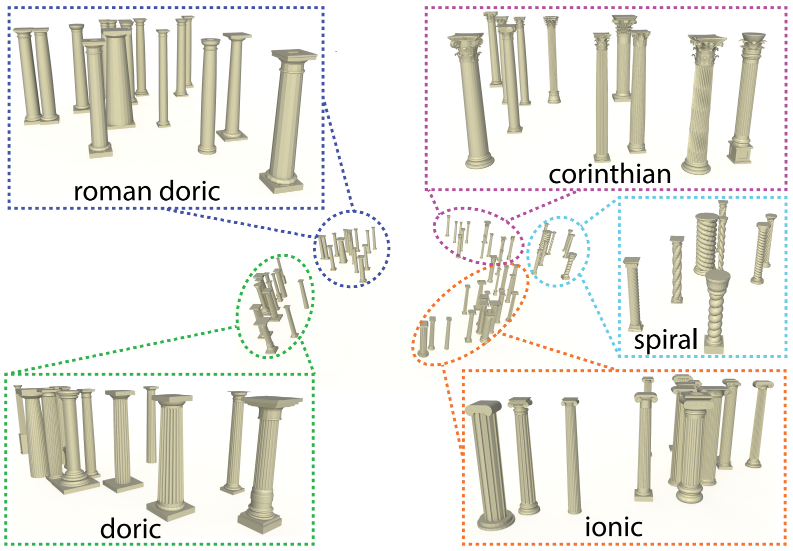


Figure 3.8. Embedding the column dataset in a 2D space based on learned pairwise distances yields distinct Gaussian-like clusters that correspond to known architectural orders visualized by colored boxes.

3.5.1 Organizing shape collections

We first constructed a graph where nodes are the shapes belonging to the triplets described in Section 3.3 and the edges represent triplet relationshaip with edge lengths equal to their computed pairwise distance. Then we embed the graph in 2D using the Isomap technique [111]. Figure 3.8 shows the resulting embedding for columns. Interestingly, the discovered groups are largely correlated to architectural orders commonly used by art historians to describe column styles.

3.5.2 Style-based shape tagging

Given a set of shapes with style labels provided by an expert, we can train a classifier that infers, or propagates, labels to the rest of the shapes in a collection. To reliably perform classification, we embed shapes in a high-dimensional space using Isomap technique and concatenate the embedding coordinates per shape as the feature vector, which is provided as input to the classifier for training and evaluation. While training the classifiers, we performed hold-out validation on the training sets to choose the dimensionality D of the embedding for each collection.We stop once the hold-out validation error increases more than 10% with respect to the best previous value of D, or when $D = 20$. On average, using single nearest neighbor classifier through ten-fold cross-validation, the labeling accuracy in the test sets was 95.6% for columns, 86.6% for buildings, and 94.1% for coffee sets.

3.5.3 Style-based suggestions for scene modeling

Finally, our learned measure can be used to help designers during interactive scene composition by providing stylistic suggestions of shapes. The input to this application is a collection of shapes and a scene being modeled. The application compiles an ordered list of shapes from a collection according to their style distance to the shapes in the scene, or selected shapes of interest (query shapes) specified by the designer. but evaluated across the entire shape, sidestepping explicit element detection. In this manner, the shape that is most structurally and geometrically similar to the query shape is found first. To generate the ordered list, we use the geodesic distances from that shape to all other shapes in the database through a precomputed graph, which is constructed using the process described for organizing shape collections. We demonstrate the application of stylistic suggestions for furniture in the accompanying video. Instead of computing the distance of the query shape to all collection shapes which is computationally expensive, we perform a nearest neighbor search in the space of features we use for element matching. Then we generate the ordered list using the geodesic distances from that nearest neighbor shape to all other shapes in the database through a precomputed graph, which is constructed using the process described for organizing shape collections.

3.6 Discussion

We have described the first algorithm for computing a structure-transcending style similarity measure between objects. As demonstrated, our measure is well aligned with human perception of style, owing to our novel use of parameter learning from crowdsourced style similarity queries. Since understanding style is fundamentally important for analysis of man-made objects, our method directly benefits a range of applications such as the ones described in the paper.

CHAPTER 4

FUNCTIONALITY PRESERVING SHAPE STYLE TRANSFER

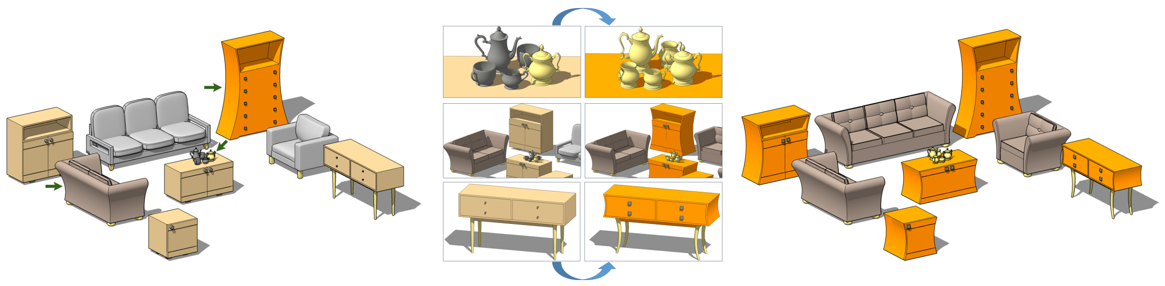


Figure 4.1. Our algorithm transfers the geometric style of three exemplars, cabinet, loveseat, and sugar pot (left, highlighted with arrows), to the rest of the objects in the scene, while preserving the target functionality (right). Please zoom-in to see more details.

When geometric models with a desired combination of style and functionality are not available, they currently need to be created manually. We facilitate algorithmic synthesis of 3D models of man-made shapes which combines user-specified style, described via an *exemplar* shape, and functionality, encoded by a functionally different *target* shape. Our method automatically transfers the style of the exemplar to the target, creating the desired combination. The main challenge in performing cross-functional style transfer is to implicitly separate an object's style from its function: while stylistically the output shapes should be as close as possible to the exemplar, their original functionality and structure, as encoded by the target, should be strictly preserved. Recent literature point to the presence of similarly shaped, salient *geometric elements* as a main indicator of stylistic similarity

The work described in this chapter has been published as a full paper in *ACM SIGGRAPH ASIA 2016*. Please also see the accompanying video: `https://youtu.be/R-1jhH4sbRk`

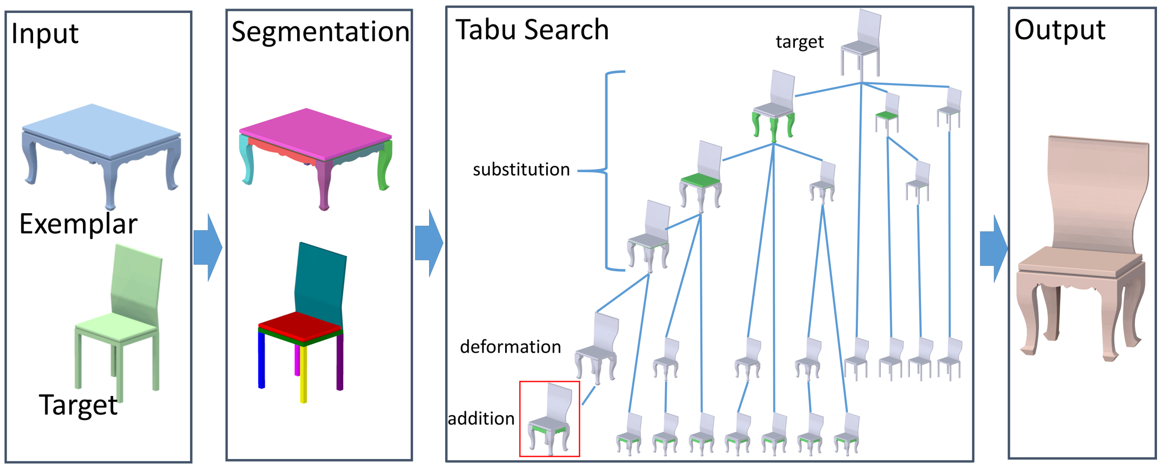


Figure 4.2. Framework overview.

between 3D shapes. We therefore transfer the exemplar style to the target via a sequence of element-level operations. We allow only *compatible* operations, ones that do not affect the target functionality. To this end, we introduce a cross-structural element compatibility metric that estimates the impact of each operation on the edited shape. Our metric is based on the global context and coarse geometry of evaluated elements, and is trained on databases of 3D objects. We use this metric to cast style transfer as a tabu search, which incrementally updates the target shape using compatible operations, progressively increasing its style similarity to the exemplar while strictly maintaining its functionality at each step. We evaluate our framework across a range of man-made objects including furniture, light fixtures, and tableware, and perform a number of user studies confirming that it produces convincing outputs combining the desired style and function.

4.1 Style Transfer Framework

Our framework takes an exemplar shape and a target shape in a different functional class as input. Given the exemplar and target shapes, we search for modifications to the target shape which bring it stylistically closer to the exemplar. It begins by hierarchically segmenting both the exemplar and the target into potential geometric elements (Section 4.2) and then employs a set of element-level target modifications that reduce the style distance between the output and the exemplar (Figure 4.2). To measure this distance we use the style measure discussed in Chapter 3.We perform only compatible operations, ones that do not violate the target functionality. We evaluate compatibility as discussed in Section 4.3.

4.1.1 Operations

The simplest and most common operation we employ to reduce style distance between an exemplar and a current shape is substituting elements on the current shape with (appropriately scaled) exemplar elements. By definition any such substitution reduces the style distance between shapes; however, it is rarely possible to replace every single portion of the target shape with exemplar elements without violating functionality. We therefore use three additional operations that can improve style similarity once substitution is no longer possible: curve-based element deformation, element addition, and element removal. The element deformation operation embeds the exemplar curve into the candidate shape by replacing its target counterpart and adapting the surrounding surface, see Section 4.5. For element addition and removal operations, We only perform those operations on elements deemed decorative, i.e. having only marginal impact on functionality as measured by our per-object compatibility function. To ensure that our operations preserve target shape symmetries, we detect all replicated elements and curve handles on the target shape, and apply each operation to an entire symmetry group instead of to a single element.

4.1.2 Tabu Search

Our optimization procedure is designed to select modifications providing maximal style adaptation while preserving target functionality, which follows the concept of tabu search [34]. A detailed pseudocode of our method is provided in Figure 4.3.

```
input  : Exemplar shape $E$ in class $c'$, Target shape $T$ in class $c$
output: An output list $\mathcal{O}$ of new shapes
Initialize search list $\mathcal{L} = \{T\}$
repeat
    Choose shape $S = \arg\min_{T' \in \mathcal{L}} D_{style}(E, T')$
    Remove shape $S$ from search list $\mathcal{L}$

    // Search for element substitutions
    for each element (or symmetric group of elements) $G_S$ in $S$ do
        Find elements $G_E$ in shape $E$ with $D_{func}(G_E, G_S) < \epsilon_{c,c'}$
        for each retrieved element $G_E$ do
            if replacing $G_S$ with $G_E$ drops $D_{style}(E, S)$ then
                Construct new shape $S'$ by aligning $G_E$
                if $D_{func}(S', T) < \epsilon_{c,c}$ and alignment is successful then
                    insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                    (unless a copy of $S'$ already exists in the output list)
                end
            end
        end
    end

    // Search for curve-based deformation
    for each curve (or symmetric group of curves) $C_S$ in $S$ do
        Find curve $C_E$ in shape $E$ with $D_{curve}(C_E, C_S) < \epsilon^{curve}_{c,c'}$
        for each retrieved curve $C_E$ do
            Construct new shape $S'$ by deforming $S$ to align with $C_E$
            if the constructed new shape $S'$ drops $D_{style}(E, S)$ then
                if $D_{func}(S', T) < \epsilon_{c,c}$ then
                    insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                    (unless a copy of $S'$ already exists in the output list)
                end
            end
        end
    end

    // Search for element additions
    for each non-used element (or group) $G_E$ in $E$ do
        if adding $G_E$ to $S$ drops style distance then
            Construct new shape $S'$ by aligning $G_E$ with $S$
            if $D_{func}(S', T) < \epsilon_{c,c}$ and alignment is successful then
                insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                (unless a copy of $S'$ already exists in the output list)
            end
        end
    end

    // Search for element removals
    for each non-substituted/added element (or group) $G_S$ in $S$ do
        if removing $G_S$ from $S$ drops style distance then
            Construct new shape $S'$ by removing $G_S$
            if $D_{func}(S', T) < \epsilon_{c,c}$ then
                insert shape $S'$ in search list $\mathcal{L}$ and output list $\mathcal{O}$
                (unless a copy of $S'$ already exists in the output list)
            end
        end
    end
until search list $\mathcal{L}$ is empty
```

Figure 4.3. Tabu search pseudo-code.

Throughout the optimization, we maintain a list of seed shapes which is initialized with the given target shape. At each iteration, we remove a seed shape from the list and attempt to bring it closer, style-wise, to the exemplar using one of the four supported element operations. To explore the most promising solutions first, we always select the seed shape currently closest to the exemplar in terms of style. If the attempt to improve it through allowable operations succeeds, then the improved shape is inserted into the seed list. If the attempt yields a shape which has been generated before, or if all evaluated operations violate the functionality preservation constraints, then we discard and forbid the output. We use the compatibility measure and the threshold learned from training data discussed in Section 4.3 to examine if an operation is compatible, and only proceed if the compatibility is higher than the threshold. We also avoid operations predicted to reduce the style distance by less than a minimum threshold. In this manner, the tabu search only searches the space of valid target shape modifications (a much smaller subset of the set of all possible modifications), and avoids performing operations on functionally implausible shapes. Once a seed shape can no longer be improved it is considered to be a terminal solution, or in other words it corresponds to a local minimum. In this case, we store it in an output list and proceed to examine the next best seed shape in our search list, terminating when this list is empty.

To reduce search space and avoid redundant operations we first perform tabu search using only element substitutions, then once compatible style-distance-reducing substitutions are exhausted we repeat the search using curve-based deformation, and finally use the same process for adding and then for removing decorative elements.

4.1.3 Improvement Step

Given a seed shape, we aim to perform a compatible editing operation that will maximally reduce the style distance from the seed to the exemplar. We thus cycle through all possible operations of the currently examined type and select the one that reduces style distance the most. We can only compute the exact impact of each operation after it is performed, since most operations change the geometry of both the elements involved and their surroundings. However, for both substitution and deformation we can reasonably predict beforehand if the operation is incompatible, or if it does little to reduce style distance. We rely on such predictions to avoid unnecessary computations. To examine if an operation is *a priori* incompatible, we use the compatibility measure and the threshold learned from training data discussed in Section 4.3, and only proceed with full computation if the predicted compatibility is higher than the threshold. We also avoid operations predicted to reduce the style distance by less than a minimum threshold. Style distances are normalized to the $[0, 1]$ interval and we use 1% as a threshold. Once the operation is performed we reassess both compatibility and style distance, rejecting operations that after the fact violate compatibility or do not sufficiently reduce style distance. We add the compatible result with the smallest distance to the exemplar to the list of seeds.

4.1.4 Substitution

Give a pair of compatible elements in the exemplar and seed shape, we seek to replace the seed shape element, including any of its symmetric counterparts, with the corresponding element on the exemplar shape. This step requires alignment or coverage of *slots* [56], i.e. areas on the elements which are in contact with the rest of the shape (Figure 4.5). The alignment step, Section 4.4, balances two potentially contradictory goals: it seeks to preserve output functionality and to minimally change the style, and specifically the proportions, of both the substituted-in and pre-existing output elements (Figure 4.5).

4.1.5 Curve-Based Deformation

Our curve based deformation step only considers seed elements originating from the target, and evaluates the compatibility of each possible deformation using a variant of the compatibility measure designed for curves (Section 4.3). The deformation embeds the exemplar curve into the candidate shape replacing its target counterpart and adapting the surrounding surface, see Section 4.5.

4.1.6 Element Additions and removals

Addition and removal are performed after substitution and deformation, the two operations expected to be most effective at minimizing exemplar-output style distance. We only perform addition of exemplar elements and removal of target elements, deemed decorative, i.e. having only marginal impact on functionality as measured by our per-object compatibility function. To add a new element to a model one needs to know the attachment point or slot to place it at. We thus only consider adding elements that on the exemplar are immediately adjacent to an element previously substituted-in or added to the seed. To add the element we first place it next to this prior neighbor using the slots they shared on the exemplar.

4.1.7 Element removals

Finally, we perform removal of target elements that are decorative as also measured by our compatibility function. We only remove elements if the slots they share with the rest of the model are closed surfaces, i.e. if no gaping holes are left after removal. Typically only a tiny fraction of elements fits these constraints.

4.1.8 Termination

Once a seed shape can no longer be improved it is considered to be a terminal solution, or in other words it corresponds to a local minimum. In this case, we store it in an output list and proceed to examine the next best seed shape in our search list, terminating when this list is empty. The tabu search typically computes and evaluates a few dozen solutions.

4.1.9 output

If the user seeks a single output, then at the end of the process we return the shape in the output list closest to the exemplar in terms of style distance. To provide multiple results with different emphasis on target functionality preservation versus exemplar style adaptation, we add to the output list all the seed models produced during the tabu search, as well as models generated with more lax compatibility thresholds.

4.2 Pre-Processing and Segmentation

Our style transfer framework requires that models are first hierarchically segmented into meaningful geometric elements. We assume that the input shapes are represented as partially connected meshes (polygon soups), are consistently scaled and oriented (have the same upright orientation, and face the same front direction when that direction is well defined). Consistent orientations can be found manually or by using the automatic method by Huang et al. [46]. Consistent scalings are computed through bounding box-based alignment.

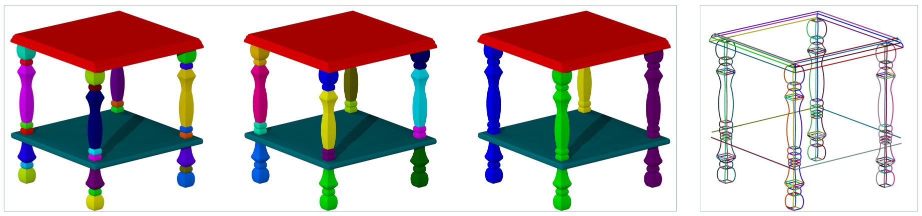


Figure 4.4. Hierarchical segmentation and extracted curves.

4.2.1 Segmentation

We follow previous works that use convexity as a criterion for shape segmentation into meaningful parts or elements. At the finest hierarchy scale, we generate geometric elements by using the approximate convex decomposition technique by [3]. We generate elements at a number of scales by repeating the segmentation with different convexity thresholds (0.3, 0.5 and 0.7). Since when designing 3D models of man-made objects artists often leave functionally meaningful parts as separate components we also add such separate connected components as potential elements. We introduce additional larger elements by merging neighboring elements when they jointly approximate a portion of a primitive (box, sphere, or cylinder). The primitive-based element grouping is based on [131]. The result of this step is a collection of a few dozen elements at a range of scales (Figure 4.4, left). We also detect symmetry groups of elememts by approximately matching them through ICP.

For each shape we build a graph representing its structure, which is consequently used for compatibility evaluation (Section 4.3). The nodes of the graph are the different elements (typically 10-50 elements per shape). The graph has three types of edges: we connect nodes by an *adjacency* edge if their corresponding elements are adjacent; we create a *symmetry* edge connecting nodes whose corresponding pairs of elements are related under a reflective, translational or rotational symmetry; and we create *containment* edges between nodes corresponding to pairs of elements where one element directly contains the other in the hierarchical segmentation.

4.2.2 Curve Handles

Our element deformation operation uses matching curve handles on target and exemplar elements. We extract two types of curve handles (Figure 4.4, right): view-independent ridges and valleys [87], and occluding contours. To compute the latter we use 12 views uniformly distributed about the upright axis at elevation angles of 0, 30, and 60 degrees above the horizontal plane. We extract the feature curves as described in [58], and hierarchically segment them along element boundaries.

4.3 Compatibility

To effectively transfer style we need to evaluate the impact of each editing operation on the functionality of the edited shape. We answer this question by using a set of compatibility measures that predict the impact of each operations and also assess the a posteriori impact of any operation on the output functionality. We formulate substitution and deformation compatibility by considering the contextual similarity between the substituted elements or deformation handles, and formulate shape-level similarity by analyzing compatibility between pairs of elements on them.

4.3.1 Formulation

Previous research, e.g. [78] as well as insights from design literature [85] point to the context and gross shape of geometric elements as important features in determining an object's functionality. While we do not aim to detect functionality, we speculate that elements with similar context and shape features are more likely to be compatible, i.e. replacing one with the other is less likely to negatively affect the functionality of the resulting shape. Our metric is designed to reflect these similarities and differences.

We encode each element's *functional* and *contextual* properties using the element relation graph contracted as described in Section 4.2. We then use a graph kernel-based similarity evaluation framework, inspired by [67] to combine those into a single compatibility metric. In contrast to Laga et al. we design our graph kernels to measure cross-class functional compatibility by choosing a different set of feature descriptors and then learning their individual importance and kernel parameters from training data. Our procedure offered dramatic improvements in performance compared to using the formulation of [67] as-is (Section 4.7).

4.3.1.1 Per-Element Descriptors

We encode each element's gross geometry and context within the overall shape using the following set of descriptors: the element's relative position with respect to its containing shape, encoded by the location of its markers such as its center of mass, lowest and highest points with respect to global object's markers (center of mass and its projections on supporting planes); its relative dimensions with respect to the object's dimensions; its mass distribution; and the relative orientation of the element's major axis with respect to the object's coordinate axes. The full set of the detailed 13 descriptors is provided in Appendix B.1.

4.3.1.2 Pairwise Descriptors

For each pair of elements connected by an edge in our graph we compute two sets of relative pairwise descriptors, using the same measurement as for individual elements, but computed for each element with respect to its graph neighbor rather than with respect to the containing object.

We assemble these descriptors into an element compatibility measure and learn their respective weights as discussed in Section 4.3.2. Intuitively, the learned weights indicate which geometric descriptors are more relevant for evaluating functional compatibility between elements on functionally different shapes.

4.3.1.3 Element Compatibility

We evaluate compatibility between pairs of elements on two shapes by comparing the graph walks initiated at their corresponding nodes in the respective graphs. Given an element p in a shape S, an n^{th} order (length) walk $W_S^{(n)}(p)$ is defined as a finite sequence of $n + 1$ vertices and n edges forming a continuous path in the graph. Given another element q in another shape E, the n^{th} order similarity $K^{(n)}(p,q)$, defined for the n^{th} order walks starting at p and q, is given by the recursive formula:

$$K^{(n)}(p,q) = K_{node}(p,q) \cdot \sum_{\substack{p' \in \mathcal{N}(p) \\ q' \in \mathcal{N}(q)}} K_{edge}(e_{pp'}, e_{qq'}) \cdot K^{(n-1)}(p',q') \tag{4.1}$$

where $K_{node}(p,q)$ is a kernel function comparing node descriptors for elements p and q; $\mathcal{N}(p)$ and $\mathcal{N}(q)$ represent the set of neighboring elements for p and q respectively; and $K_{edge}(e_{pp'}, e_{qq'})$ is a kernel function comparing edge (i.e. pairwise) descriptors that represent relationships between elements. For $n = 0$ (0^{th} order walk), the kernel function only evaluates $K_{node}(p,q)$.

We define the node and edge kernels as a weighted combination of Radial Basis Function (RBF) kernels with learned parameters which are evaluated respectively as follows:

$$K_{node}(p,q) = \sum_k w_k \cdot \exp\left\{ -\frac{D_k^2(p,q)}{2\sigma_k^2} \right\} \tag{4.2}$$

$$K_{edge}(e_{pp'}, e_{qq'}) = \delta(e_{pp'}, e_{qq'}) \sum_l w_l \cdot \exp\left\{ -\frac{D_l^2(e_{pp'}, e_{qq'})}{2\sigma_l^2} \right\} \tag{4.3}$$

where $D_k(p,q)$, $D_l(e_{pp'}, e_{qq'})$ are distances between the descriptors of nodes and edges respectively, $\delta(e_{pp'}, e_{qq'})$ is a binary function that returns 1 when the edges $e_{pp'}, e_{qq'}$ are of the same type (symmetry, containment, or adjacency) and 0 otherwise.

Compatibility between elements p and q is then defined as a weighted combination of n^{th} order similarities between them across a range of walk lengths n:

$$K_{func}(p,q) = \sum_n w_n K^{(n)}(p,q) \tag{4.4}$$

where w_n is a learned weight for each different walk length. For computational efficiency, in our implementation we use walks up to length 5, as in our experiments the learned weights assigned to longer walks were negligible. The above similarity function is a kernel function itself, and can therefore be normalized to ensure consistent similarity values for graphs of different size [68]:

$$\hat{K}_{func}(p,q) = \frac{K_{func}(p,q)}{\sqrt{K_{func}(p,p) \cdot K_{func}(q,q)}} \tag{4.5}$$

Given positive weights $\{w_k\}$, $\{w_l\}$, $\{w_n\}$, our kernel is guaranteed to be positive definite, thus distances between elements can be derived from the above kernel as follows [101]:

$$D_{func}(p,q) = \sqrt{K(p,p) - 2K(p,q) + K(q,q)} \tag{4.6}$$

where $K(p,p), K(q,q)$ represent the self-similarities of elements in the graphs used for normalization.

To evaluate compatibility between pairs of symmetric group of elements G_E, G_S, we find the best pairwise element match. If the best match is compatible for substitution or deformation, this indicates that at least one pair of elements are interchangeable. The rest of the elements within their respective symmetric group can be substituted or deformed under symmetry constraints. Thus, we use the compatibility of the best element match for measuring group compatibility $D_{func}(G_E, G_S)$:

$$D_{func}(G_E, G_S) = \min_{p \in G_E, q \in G_S} D_{func}(p,q). \tag{4.7}$$

4.3.1.4 Shape Compatibility

We employ the shape compatibility measure after each editing operation, to evaluate whether the resulting new shape S' is functionally compatible with the original target one T . We define compatibility as the maximal compatibility distance between corresponding elements on the two shapes, seeking the worst-case influence on shape compatibility:

$$D_{func}(S', T) = \max_{p \in T, p' \in S'} D_{func}(p, p') \tag{4.8}$$

where p is an element on the original target shape, and p' is its corresponding substituted or original element on the generated shape. Note that while added or removed parts are not explicitly accounted for by this metric, their presence or absence will be reflected in the graph kernels of their neighboring elements.

4.3.1.5 Curve Compatibility

To evaluate curve compatibility for curve-based deformation, we take into account the compatibility of the elements they belong to, and the similarity between the shape, relative location and size of the curves within the overall shape. The list of curve descriptors is provided in Appendix B.1.

For a pair of *view-independent* curves e, f belonging to elements p, q respectively, their compatibility is expressed as follows:

$$K_{curve}(e, f) = K_{func}(p, q) + \sum_m w_m \cdot \exp\left\{ -\frac{D_m^2(e, f)}{2\sigma_m^2} \right\} \tag{4.9}$$

and $D_m(e, f)$ represent distances between curve descriptors and $\{w_m\}, \{\sigma_m\}$ are learned parameters. We note that the curves are segmented according to our hierarchical element segmentation such that the curve segments can be associated with the corresponding element compatibilities.

For a pair of *view-dependent* curves e, f , we additionally take into account the distance between the views they are generated from:

$$K_{viewcurve}(e, f) = K_{curve}(e, f) + w_v \exp\left\{ -\frac{||\mathbf{v}(e) - \mathbf{v}(f)||^2}{2\sigma_v^2} \right\} \tag{4.10}$$

where $\mathbf{v}(e)$ and $\mathbf{v}(f)$ represent given 3D viewpoint location for these two curves, and w_v, σ_v are learned parameters.

Curve compatibility is defined by converting kernel similarity to distance (as in Equation 4.6). Our deformation only pairs same-type curves; that is, we do not match view-dependent curves on one shape with view-independent curves on the other.

4.3.2 Parameter Learning

We algorithmically learn the parameters of our element and curve compatibility measures with the same learning procedure. For element compatibility these include the kernel weights $\{w_k\}_{k=1...K}$, $\{w_l\}_{l=1...L}$, $\{w_n\}_{n=1...N}$ and RBF variances $\{\sigma_k\}_{k=1...K}$, $\{\sigma_l\}_{l=1...L}$ (58 parameters in total). For curve compatibility these include the weights $\{w_m\}_{m=1...M}$, $\{\sigma_m\}_{m=1...M}$ and w_v, σ_v (8 parameters in total). We use the same learning procedure for both. We note that these parameters can vary across object classes - compatibility criteria for chairs and sofas may differ from those for beds and cabinets. We consequently learn these parameters separately for each pair of shape classes. In our experiments we used coarse class classification with up to five classes per broad shape category (e.g. tables, chairs, sofas, cabinets, and beds for furniture).

Clearly, learning requires training data. One possibility to create a training dataset is to manually specify pairs of compatible or incompatible elements or curves across shapes. However, creating such a dataset requires human labor and supervision. Instead, we developed an automatic procedure to create training data. Specifically, we observe that online repositories such as Google Warehouse already contain a significant number of coordinated sets of objects in the same style. Since these shapes are designed to have the same style, many of the objects in these scenes contain elements which are identical up to an affine transformation. By construction these elements are compatible, since they can be clearly substituted (subject to appropriate scaling) without affecting shape functionality. Consequently, detected pairs of such compatible elements across different models yield valuable training data for learning compatibility parameters. We clearly detect only a subset of compatible pairs since compatible elements may have different geometry even on same set shapes. However, our compatibility function is based on coarse-scale element properties and context and does not consider fine-level element geometry. Thus, restricting our training data to elements identical up to affine transformation, does not, in our experience, bias our learning setup. On the assumption that most random element substitutions would lead to structurally or functionally invalid results, we complement our compatible pairs with less compatible ones using random pair assignment.

Given a dataset of scenes downloaded from Google Warehouse, we first segment each model, extracting elements and curves (Section 4.2); we then use an ICP based alignment to compute all pairs of elements approximately identical up to an affine transformation. Given these training pairs, the goal of parameter learning is to compute the set of parameters with which our compatibility function will, on average, deem these pairs p, q more compatible than element pairs which contain one of the elements in our compatible pair and a randomly selected one - p, r or q, r. We use a probabilistic framework that is well suited to handle such relative comparisons for training and is known to be robust to outliers [12, 108]. We express the probability that a pair $\{p, q\}$ is more compatible than $\{p, r\}$ (or more compactly $p_q \triangleright p_r$) as:

$$P(p_q \triangleright p_r) = \sigma\Big(D_{func}(p, r) - D_{func}(p, q)\Big) \tag{4.11}$$

where $\sigma(x)$ is a sigmoid function that converts the functionality differences into probabilities. We also include an L^1 norm as regularization term that minimizes the weights assigned to the different descriptors. The L^1-norm regularization, proposed by Tibshirani [113], promotes sparsity by allowing some weights to dominate while pushing others toward zero. In addition, when the number of training pairs is small relative to the number of parameters, the regularization encourages more zero weights, leading to a simpler model with better predictive performance. Our regularizer is formulated as follows:

$$P(\mathbf{w}) = \exp\Big(-\lambda ||\mathbf{w}||_1\Big) \tag{4.12}$$

where the weight vector $\mathbf{w}$ includes all kernel weights. The regularization parameter λ controls the degree of regularization and is automatically estimated through 10-fold cross-validation on the training set.

Given T training triplets p, q, r, we learn the parameter values that maximize the following likelihood:

$$L(\mathbf{w}, \boldsymbol{\sigma}) = \sum_{t=1}^{T} \ln P(p_q[t] \triangleright p_r[t]) \tag{4.13}$$

where vector $\boldsymbol{\sigma}$ includes all variances, $p_q[t] \triangleright p_r[t]$ refers to the automatically generated training triplet t. For element correspondences, we train the weights and variances by maximizing the above likelihood function on the element training data for input shapes per each pair of classes. Then for curve correspondences, we train the weights and variances by again maximizing the same likelihood function, but this time using curve training data for input shapes per each pair of classes. We use bound constraints to enforce all parameters to be positive. To maximize our regularized likelihood function, we use the L-BFGS-B method [139]. We note that analytic gradients of our kernel functions with respect to weights can be derived following a recursive formulation explained in Appendix B.2. In our datasets, the number of our training inputs based on the ICP-aligned pairs varied from 25 to 300 depending on the pair of classes (most were above 100). To encourage more zero weights for a simpler model with better predictive performance, we also include an L^1 norm as regularization term that minimizes the weights assigned to the different descriptors.

4.3.2.1 Automatic Threshold Selection

We use the detected element and curve correspondences to algorithmically select the compatibility threshold ϵ used in our tabu search. For each pair of classes c, c', we set the threshold $\epsilon_{c,c'}$ for element correspondence to the maximum distance between correspond-

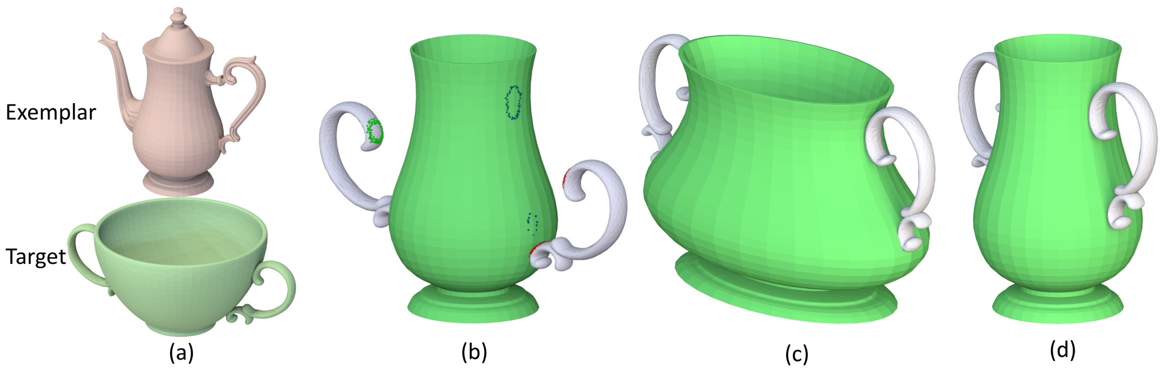


Figure 4.5. Element alignment: (a) exemplar and target; (b) seed model and substituted-in element with identified slots and their covers; (c) alignment using non-uniform scaling across the board; (d) style and structure aware alignment.

ing elements in the training data. We similarly use the maximum distance between corresponding curves in our training data as the threshold for curve compatibility. We note that we can safely use these maximum distances as thresholds since any outlier matches are pruned by the ICP matching step.

4.4 Element Alignment

Part and element adjacencies within an object obviously impact its functionality. In particular the locations of contact areas, or slots, connecting each element to the rest of the model are likely to reflect on this element's role within the larger whole. To preserve target functionality when adding or substituting elements into an edited seed shape we aim to, whenever possible, preserve all previously existing slots on both the incorporated element and the seed model, i.e. to keep previously covered, or in contact areas, similarly covered. We detect all slots on the seed shape and exemplar element, using the algorithm developed by [56] for part-based model synthesis. The identified slots include shared boundary loops, in-contact surfaces, and part-intersections. To preserve functionality, we treat object contacts with the ground plane as additional slots. By construction, within a model each slot has an opposite matching, or *cover*, slot. To assemble the new model, we need to compute such covers for slots at the interface between the seed shape and the new element, and then transform the elements to bring all pairs of matching slots into contact (Figure 4.5).

Aligning, or bringing slots into contact, often requires changes to element geometry, e.g. incorporating an armchair back into a sofa requires stretching it. Yet, unconstrained changes to element shape, can decrease the output functionality and negatively affect style similarity with the exemplar. Thus in performing alignment we seek to achieve the balance of changing element geometry enough to provide coverage but with minimal style and function degradation. While the method of Kraevoy et al. [65] seeks to preserve geometric features when non-uniformly resizing models, adapting it to our setting and applying it on a per element basis using coverage constraints is too computationally expensive. Instead, we facilitate an effective yet efficient alignment computation using the following framework. We first restrict the set of allowable per-element transformations to translation, rotation, and axial scaling. By preventing non-axial shear, and penalizing deviations from pure translation we seek to weakly preserve element proportions and orientation. However, applying a penalty approach to all elements uniformly is insufficient. Even small non-uniform scaling can lead to visible artifacts by breaking element symmetry (Figure 4.5 c); and even small rotations of anisotropic elements can affect their look and functionality. We therefore disallow symmetry violating scaling and rotations that change the direction of the major axis on anisotropic elements. To detect both scenarios we use the element's oriented bounding box (OBB) . When an element has two or more OBB axes with roughly similar length we constrain our transformations to maintain their length ratio (we use 20% deviation as conservative threshold). We only allow rotations for elements that are either isometric or that have two near identical axis lengths, in the later case rotation is allowed only within the plane span by these axes, using the same threshold as above to detect similar axis lengths. We also note that from a style perspective changes in thickness of thin elements are particularly undesirable, and disallow such changes (an element is considered thin if one of its axes is shorter than 10% of the sum of all axis lengths).

As in many alignment settings we face a chicken-and-egg problem, we need correspondences to perform the desired transformations, but correspondences computed when two objects are far apart are not reliable. We consequently use an iterative-closest-point (ICP) strategy, iterating alignment and correspondence steps. We first approximately align the new element to the seed shape. For substitution we transform the incoming element to align its OBB with that of the outgoing one. During addition, the added-in element by construction has at least one adjacent exemplar element that had been incorporated into the seed. We therefore similarly transform the added-in element to align the slots it shares with those elements. We then locate and pair seed and element slots nearest to one another. For any unpaired slot we treat the closest points on the opposite model as the matching covers.

At each subsequent alignment step, to minimize changes in element proportions and orientations we first solve for closest slot alignment using only translations. If this step is unsuccessful, we use the set of permissible scaling constraints per element to perform a restricted scale plus translation closest-point alignment of all participating elements. For each element we restrict the scalings to the permissible ones, while seeking to distribute the amount of scaling evenly between all elements. If and when this step fails we repeat the closest-point alignment allowing restricted element rotations and scales. For all symmetric groups of incorporated or seed elements, we constrain the transformations to preserve these symmetries. We iterate between correspondence computation and alignment till distances no longer improve or full coverage is achieved.

4.5 Curve Based Deformation

The input to our curve based deformation is a handle curve on the currently processed seed shape and a corresponding exemplar curve. Our deformation step modifies the seed by replacing the handle with the exemplar curve while smoothly deforming the seed surface so as to conform to the new curve geometry while preserving local surface details

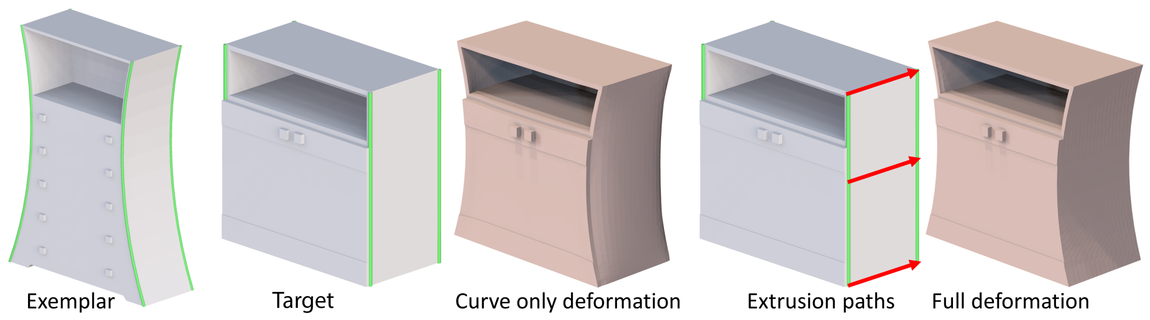


Figure 4.6. Curve based deformation without (center) and with (right) swept surface edits.

(Figure 4.6). While multiple surface deformation methods exist, we found that the ARAP framework [104] works well in our setup, as it supports curve deformation handles and preserves local geometric features under significant handle deformations. To facilitate deformation, we first align the endpoints of the exemplar curve with those of the handle curve through translation and uniform scaling and use arc-length parameterization to define curve-to-curve correspondences. We then deform the seed model by moving handle vertices to corresponding locations on the transformed exemplar curve. Using, the original, surface-based ARAP formulation as-is for large curve deformations can cause surface self-intersections. We therefore implemented ARAP on a volumetric graph, following the graph construction described in [137] shown to prevent self-intersections in the case of Laplacian deformations. In our experiments, this modification allows for the intersection-free large deformations necessary to modify curve style.

We seek to impact not only the features but also the contours of the output shape, and note that man-made objects are frequently dominated by swept surfaces. We implement the desired contour changes by editing the sweep profiles on these surfaces (Figure 4.6, right). For each handle contour curve we examine whether its underlying surface is well defined by sweeping the contour handle along a path curve, and maintain these swept surfaces during deformation. For simplicity we only implemented this mechanism for the most common sweep cases, revolution and extrusion, where this structure is easiest to detect and preserve.

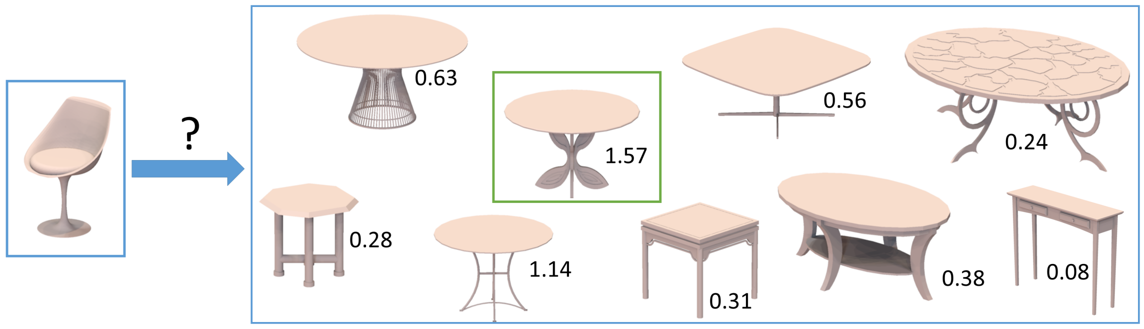


Figure 4.7. Among all possible tables on the right we selected the highlighted one as most compatible target for the exemplar chair.

Specifically, for each pair of similarly shaped and oriented handle curves on an element, we interpolate the curves and compare the distance from the resulting surface to the element. If the generated surface is close to the mesh surface, then we infer that it is a swept surface. To detect extrusions we use linear interpolation, and to detect surfaces of revolution we interpolate handle normals and positions.

4.6 Automatic Target Selection

Our output is dependent on the choice of a particular target shape (Figure 4.7). Typically the more similar the exemplar and the target are structurally, the more compatible their elements are, and the more complete, or compelling, the style transfer. Thus when users specify a database of shapes within a particular class as a target for style transfer, we use structural compatibility as a criterion for selecting the target shape to operate on within the database.

Intuitively one shape is more compatible with a given exemplar than another when a larger share of its elements are more compatible with exemplar elements. Given the exemplar shape E and a shape D within a target class, we compute their compatibility by first locating for each shape element the most compatible exemplar element, and then summing up the degrees of compatibility between them using the normalized kernel of Equation 4.5:

$$\hat{K}(D,E) = \sum_{p \in D, q \in E, q = s(p)} \hat{K}_{func}(p,q)$$

where p is an element on the database shape, and q is its most compatible element on the exemplar shape. A simple brute-force approach for selecting the most compatible shape is to evaluate these similarities across all database shapes and select the best one.

However, for large shape collections, this brute force approach is too slow. We speed up the process by leveraging the observation that in practice shape databases frequently contain clusters of structurally similar shapes. We first find such clusters, then select a representative shape per cluster, and finally perform the above computation only for those representative shapes, selecting one of them as the target. We perform clustering using affinity propagation [27] with the following similarity metric between two database shapes D_1, D_2:

$$\begin{aligned}\hat{K}(D_1,D_2) = &\frac{1}{|P|} \sum_{p \in D_1, q \in D_2, q = s(p)} \hat{K}_{func}(p,q) + \\ &\frac{1}{|Q|} \sum_{q \in D_2, p \in D_1, p = s(q)} \hat{K}_{func}(p,q)\end{aligned}$$

where $|P|$ is the number of elements in shape D_1, $|Q|$ is the number of elements in shape D_2, $s(p)$ returns the most similar element in D_2 to element p, $s(q)$ returns the most similar element in D_1 to element q. The affinity propagation method automatically infers both the number of clusters and their representative shapes.

4.7 Validation

We evaluate our method by synthesizing over a hundred new shapes using style transfer, see Figures 4.1 and 4.8 for representative examples.We tested our method on four broad categories of everyday objects: furniture, lamps, cutlery, and coffee and tea sets. Our

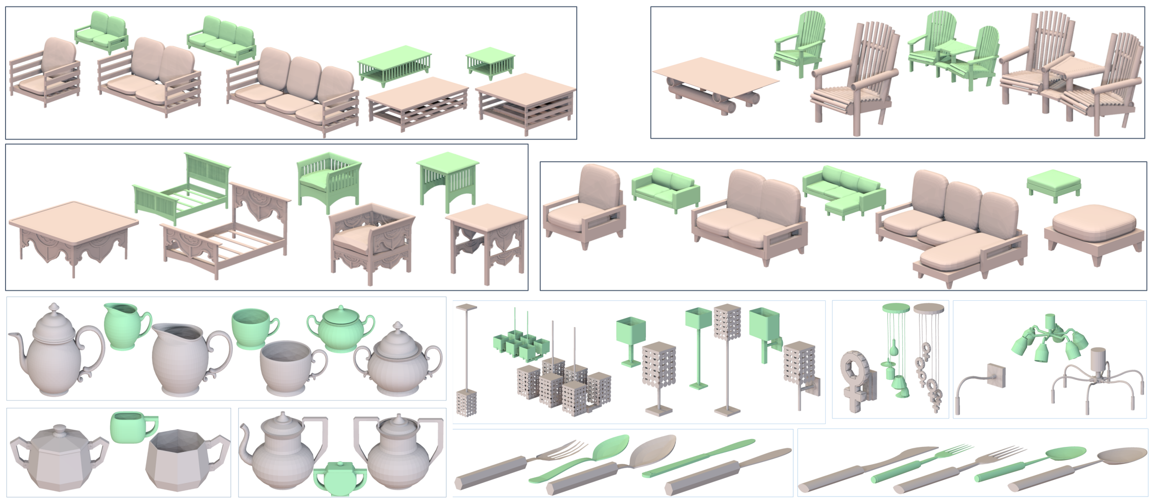


Figure 4.8. Typical style transfer results. For each group we show the exemplar first then, multiple synthesized outputs in the same style with targets shown as insets.

choice of categories was motivated both by availability and by diversity of functions and styles within each category. We use as inputs models from publicly available databases, 3D warehouse and TurboSquid. Throughout the paper we demonstrate a diverse range of style transfer results which convincingly combine exemplar styles with target functions.

4.7.1 Perceptual Validation

We validate the key properties of our method via three user studies: one designed to evaluate the degree of style similarity between the outputs and the exemplars, one designed to evaluate the functionality of the output models, and one designed to specifically evaluate our compatibility metric against the most similar prior work [67]. We summarize those below.

4.7.2 Style similarity

Style similarity is an inherently relative notion, thus asking if two shapes have the same style is often inconclusive. We consequently use relative comparison to assess our results. We asked participants to compare style similarity between an exemplar model and our

	plurality					raw votes			
	T	O	both	neither	draw	T	O	both	neither
top (T) vs target (O)	100.0%	0.0%	0.0%	0.0%	0.0%	93.2%	0.4%	0.5%	6.0%
top (T) vs third (O)	78.9%	1.4%	9.9%	5.6%	4.2%	68.6%	5.5%	15.1%	10.8%
top (T) vs original (O)	38.8%	26.9%	19.4%	1.5%	13.4%	39.1%	32.5%	21.2%	7.2%

Table 4.1. Style similarity study results: per-query plurality responses (left) and raw vote percentages (right).

output generated from it, against style similarity between the exemplar and a range of alternatives, aiming to ascertain the degree of success our method has at believably transferring style. We used questionnaires based on triplets of models, laid out with one shape image on the top and two on the bottom. The shape on the top (A) is an exemplar shape and one of the two shapes on the bottom (B or C, assigned randomly) is the top result synthesized by our method using this exemplar and a target in a different functional class (top). The second shape on the bottom is in the same functional class as the output and is randomly selected among the following alternatives: a shape from a style-coordinated pre-existing scene which included the exemplar A (original) - these shapes can be viewed as plausible ground truth for style transfer; a shape synthesized by our method using the same exemplar, but ranked as third in terms of its stylistic similarity to the exemplar (third) - this shape is useful to evaluate the meaningfulness of our ranking; and the target shape used for style transfer (target) which serves as a random baseline, expected to be arbitrarily different style-wise from the exemplar. Subjects were asked the question “Which of the two shapes on the bottom (B or C) is more similar, style-wise, to the shape on the top (A)?” and were asked to select one of the following answers: “(i) B, (ii) C, (iii) can’t tell - both B and C, (iv) can’t tell - neither B nor C”.

We assembled a total of 264 queries, up to three per each of our generated outputs comparing each output to all available alternatives. We gathered answers to each query from 10 different, reliable users.Vote distribution by query and raw vote percentages for each answer are listed in Table 4.1. Participants perceived our synthesized shapes as at least as

	plurality			raw votes	
	yes	no	draw	yes	no
target	96.0%	1.6%	2.4%	89.8%	10.2%
top	92.1%	3.2%	4.8%	86.7%	13.3%
third	90.1%	7.0%	2.8%	86.3%	13.7%
lax compatibility	69.8%	23.0%	7.1%	68.4%	31.6%
Laga et al.	65.3%	29.8%	5.0%	65.2%	34.8%
exemplar	12.2%	86.1%	1.7%	17.0%	83.0%

Table 4.2. Functionality study results: per-query plurality responses (left) and raw vote percentages (right).

similar style-wise to the exemplars as the ground truth models. Furthermore the top-ranked shapes were perceived as more style-wise similar to the exemplar compared to the third-ranked ones, and drastically more similar when compared to the baseline target shapes. These results strongly validate our claim of consistently successful style transfer across shapes with different functionality.

4.7.3 Functionality

Functionality is a largely boolean property, thus to evaluate how well our outputs preserve target functionality we show participants one model at a time and ask "Is this a functional X?" where X is the name of the specific, narrow, target class used for synthesis, e.g. coffee table, loveseat, side table, etc. Users were asked to choose either "yes" or "no". To provide a baseline to compare against, in addition to showing participants our top and third ranked results, we also included equal numbers of models from the following groups: original target models - intuitively one would expect a near 100% positive response on these models, with the actual positive response rate providing a good upper bound to compare against; top-ranking results synthesized using our framework but with either our compatibility metric but with a 10-times more lax compatibility threshold, or with the original threshold but with the similarity metric of Laga et al. [67] (based on the graph encoding, edge relationships and kernel parameters described in their paper) - intuitively we expect these two sets

	plurality	raw votes
ours	93.3%	91.8%
Laga et al.	5.0%	8.2%
draw	1.7%	

Table 4.3. Element compatibility study results: per-query plurality responses (left) and raw vote percentages (right).

of results to produce less positive responses than ours; and last exemplar models - these serve as the lower bound, as they do not share target functionality.

We assembled a total of 611 queries and gathered answers to each query from 10 different, reliable users. The responses are reported in Table 4.2. The results demonstrate that our synthesized shapes are deemed to fulfill their function at nearly the same rate as the ground-truth target models. If we relax our learned threshold for element compatibility, the functional plausibility of shapes drops significantly. Similarly, over a third of the shapes synthesized using Laga et al.'s metric are found to violate functionality considerations. The results validate the second goal of our method - the ability to reliably preserve target functionality during transfer. The comparisons to alternative methods also confirm that our compatibility metric and the automatic threshold setting we employ (Section 4.3) are key to this success.

4.7.4 Element compatibility metric

We directly evaluate our metric's effectiveness by comparing the correspondences it computes against those produced using the metric of [67]. To compare the methods we randomly selected pairs of an exemplar and a target across our inputs, and then selected a random element on the exemplar. We ran both methods to find its corresponding, or most compatible element on the target.

Our user study focused on the queries where both methods disagree with the correspondences. We used questionnaires based on triplets of models, laid out with one shape image on top and two on the bottom. The shape on the top (A) is an exemplar shape with the selected element highlighted, one of the two shapes on the bottom (B or C, assigned randomly) is the compatible target element selected by our method and the second shape is the element selected by the method of Laga et al. Subjects were asked "Which of the two highlighted parts on the bottom (B or C) is MORE similar functionality wise to the highlighted part on the top (A)?", and were asked to select either B or C. The user study had the same format and filters as the first study.

Study participants selected our result 93% of the time, and only on 5% queries did a plurality of respondents prefer the correspondences computed by Laga et. al (1.7% were a draw). Most of the outliers were on queries which compared elements on lamps with different attachment mechanisms (floor vs ceiling vs wall) . These results confirm that our metric is significantly better aligned with human perception of functional part compatibility. At the same time additional features may be useful to consider to address attachment diversity when processing hanging shapes.

4.7.5 Implementation and Runtimes

Our method is implemented in C++. Our method takes on average 6 min to synthesize a new model, with roughly 2 min out of this time spent pre-processing the models. The rest of the time is spent in the tabu search. Tabu search runtime depends on the complexity and number of operations, and ranges from 2 min for typical models to up to 10 min for the slowest ones. Learning the parameters of our compatibility measure requires about one hour for each pair of shape classes. This learning step is an offline process: once the compatibility measure is learned, evaluating the compatibility between all pairs of elements on two shapes takes only a few seconds. All running times are reported on an Intel E5-2697 v2 processor. Our source code is available on our project page: `https://people.cs.umass.edu/~zlun/papers/StyleTransfer`.

4.8 Discussion

We have described the first algorithm for synthesizing shapes by transferring style between man-made objects with different structure and functionality. As demonstrated by our results, given a single exemplar model, our method is able to successfully generate functional, plausible, similar-style models in a wide range of shape classes. Key to our success is a novel, learned metric designed to assess element compatibility across shapes with different structure and function.

CHAPTER 5

SHAPE RECONSTRUCTION FROM SKETCHES VIA MULTI-VIEW CONVOLUTIONAL NETWORKS

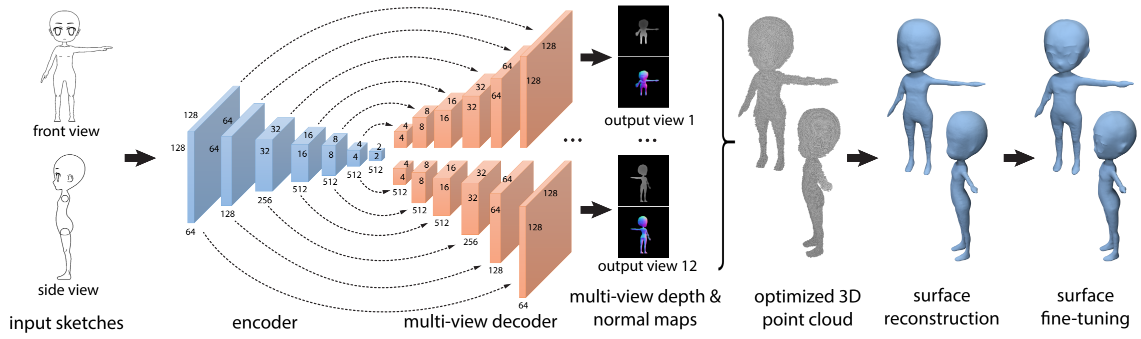


Figure 5.1. Our method takes line drawings as input and converts them into multi-view surface depth and normals maps from several output viewpoints via an encoder-multi-view-decoder architecture. The maps are fused into a coherent 3D point cloud, which is then converted into a surface mesh. Finally, the mesh can be fine-tuned to match the input drawings more precisely through geometric deformations.

In this chapter we propose a method for reconstructing 3D shapes from 2D sketches in the form of line drawings. Our method takes as input a single sketch, or multiple sketches depicting an underlying (unknown) 3D shape from different viewing angles, and outputs a dense point cloud representing a 3D reconstruction of the input sketch(es). The output point cloud is then converted into a polygon mesh representation, which is further fine-tuned to match the input sketch more precisely. At the heart of our method lies a deep, encoder-decoder network. The encoder converts the sketch into a compact representation

The work described in this chapter has been submitted as a full paper in *3DV 2017.*

which encodes shape information based on the input sketch(es). Then the decoder converts this representation into depth and normal maps that capture the underlying surface from several, densely sampled, output viewpoints. The multi-view maps are then consolidated into a coherent 3D point cloud by solving an optimization problem that fuses depth and normal information across all output viewpoints. Compared to other approaches, such as volumetric-based networks, our multi-view architecture offers several advantages, including more faithful reconstruction, higher output surface resolution, better preservation of surface detail and shape structure. We validated our results quantitatively through standard error measures, and qualitatively through a perceptual user study.

5.1 Overview

Given a single, or multiple, hand-drawn sketches in the form of line drawings, our method aims to reconstruct a 3D shape. Line drawings are made by humans to convey shape information [21, 19]. They typically contain external contours (silhouettes) and internal contours to underlie salient shape features. We designed a deep network to automatically translate line drawings into 2D images representing surface depth and normals across several output viewpoints (Figure 5.1). The depth and normal predictions are then fused into a 3D point cloud, which is in turn converted into a polygon mesh. Although surface normals could be inferred by depth alone, we found that best reconstructions are achieved when both depth and normal predictions are made by the network and coherently fused into the point cloud.

Our network is trained to reconstruct multi-view depth and normal maps from either a single sketch depicting the shape from a particular input view (*e.g.*, front, side, or top), or from multiple sketches depicting the shape from different views (*e.g.*, front and side). A single sketch may not be sufficient to reconstruct the shape accurately, *e.g.*, the front side of a car does not explicitly convey information about its back. Thus one would need

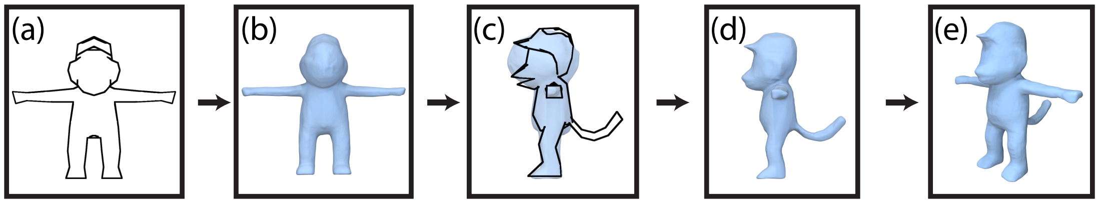


Figure 5.2. (a) The user can provide a front view sketch as input; (b) our network trained on a single input sketch generates an intermediate shape; (c) the user can further draw a sketch from the side view using the rendered shape as a guide; (d) & (e) our network trained on inputs from both views yields an updated 3D shape.

to entirely rely on the network to reconstruct the complete 3D shape based on that single sketch and the learned shape information during training. Yet, since there is a large, or even infinite number of shape reconstructions (*e.g.*, different car backs) from a single sketch , we also allow the user to provide multiple sketches depicting the shape from difference views as input at once, or provide them progressively while being guided by the intermediate shape reconstructions. In the latter case, illustrated in Figure 5.2, the user draws from one view, then our network, which is trained to reconstruct from that view, yields a 3D shape. The user can then draw a second sketch from another view, on top of the generated shape rendered semi-transparently from that view, similar to ShadowDraw [71]. Thus, given the previous and new line drawing as input, our network, trained to reconstruct from both views, yields an updated 3D shape. The process can continue until the user is satisfied with the result, at which point he/she may edit the mesh directly.

In what follows, we discuss our network architecture (Section 5.2), and its training (Section 5.3). Then, we discuss our optimization technique to fuse the muilti-view depth and normal maps into a single, coherent 3D point cloud, and finally the conversion to a polygon mesh (Section 5.4).

5.2 Network Architecture

Our ConvNet takes as input line drawings from particular views of an object and outputs depth and normal maps in several, uniformly sampled output viewpoints (Figure 5.1). Our implementation uses 12 output viewpoints located at the equidistant vertices of a regular icosahedron. A camera is placed at each icosahedron vertex looking towards the center of the object and oriented towards the upright axis. All our training shapes are normalized such that they fit inside the icosahedron and are consistently oriented.

5.2.1 Input

The input to our network are 256×256 intensity images representing the line drawings. When C input sketches are available, they are concatenated as channels resulting in $256 \times 256 \times C$ dimensional input. For each input view configuration, we train a different network *i.e.*, given a sketch representing the front of the object, we use the network trained to reconstruct the 3D shape from the front, or given two sketches representing the front and the top of the object, we use the network trained to reconstruct from the front and top (in this case, the two sketches are concatenated in this order). At first, this might seem restraining, yet we note that in traditional CAD systems, it is common for users to use canonical views [93], and better reconstruction results are achieved when the network is trained to reconstruct from specific rather than arbitrary views.

5.2.2 Encoder

The encoder network consists of a series of convolutional layers, all using kernel size of 4 and stride of 2. The filter size and number per layer is shown in Figure 5.1. All layers use batch normalization and leaky ReLUs (slope = 0.2) as activation functions. The output of the encoder is a $2 \times 2 \times 512$ representation, which encodes shape information based on the input sketch(es). We note that this representation can be used for sketch-based shape retrieval.

5.2.3 Decoder

The decoder consists of 12 branches, each containing a series of nearest-neighbor upsampling and convolutional layers. The branches have the same layer structure but do not share parameters. Each branch takes as input the encoder's representation and outputs a $256 \times 256 \times 5$ image for a corresponding output viewpoint. The 5-channel image includes a depth map (1 channel), a normal map (3 channels) and a foreground probability map for that viewpoint. All pixels that have probability more than 50% as foreground serve as a binary mask that indicates the projected surface area under that viewpoint. The output depth and normal maps are masked using this binary foreground map.

Following the U-net architecture [94], the input to each convolutional layer is formed by the concatenation of the previous layer output in the decoder, and a corresponding layer output in the encoder (see Figure 5.1). The upsampling layers of the decoder upsample their input with a factor of 2, while the convolutional layers use kernel size of 4 and stride of 1. Each convolutional layer is followed by batch normalization and leaky ReLU (slope = 0.2) as activation function. The first 3 layers in each decoder branch use dropout for regularization. The number and size of filters per layer in the decoder are shown in Figure 5.1. The output layer uses the $\tanh$ activation function since depths and normals lie in range $[-1, 1]$. Finally, the normal maps pass through a ℓ_2 normalization layer that ensures they are unit length.

5.3 Training

Given a training set of 3D shapes, the goal of our training procedure is to jointly learn the parameters of the encoder and the decoder such that our network can reliably map sketches to foreground, depth, and normal maps for our output multi-view configuration. To this end, we need to acquire training sketches according to our input view setting. One option would be to ask human subjects to provide us with line drawings depicting each training shape. However, gathering human line drawings is labor-intensive and time-consuming. In contrast, we generated synthetic line drawings that approximate human line drawings based on well-known principles. Below we discuss the procedure we followed for sketch generation, then we discuss the objective used for learning the network parameters.

5.3.1 Generating training sketches

Non-photorealistic rendering algorithms can be used to create synthetic line drawings of 3D shapes. First, contours, or silhouettes, can be estimated by finding and connecting the set of points on the surface whose normal vector is perpendicular to the viewing direction [21]. Second, suggestive contours are extensions of contours that can be used to draw internal feature curves in shapes. These are found from zero-crossings of the radial curvature (surface curvature along viewing directions) [21]. Other types of internal feature curves include ridges and valleys, which are formed by the minima or maxima of the surface principal curvature values [87], or view-dependent curvature (in this case, they are called "apparent" ridges) [55]. Another type of line drawings involves edge-preserving filtering [33] applied on rendered images of shapes under a simple shading scheme (e.g., Phong shading) [91]. All these feature curve definitions do not necessarily coincide each other [18]. We use a combination of these techniques to create several variants of line drawings per input shape. This also serves as a form of data augmentation. Specifically, for each shape and input views, we create 4 synthetic sketches by using: (i) silhouettes alone, (ii) silhouettes and suggestive contours, (iii) silhouettes, suggestive contours, ridges, valleys and apparent ridges, (iv) and edge-preserving filtering in rendered images of shapes. All training sketches and corresponding ground-truth depth and normal maps are rendered under orthographic projection (quite common for man-made objects) according to our output viewpoints. Rendering under perspective projection could also be an option, however, since depth has a relatively short range for our rendered objects, the differences in the resulting images would be small. It is also common for users, such as architects, to use orthographic projection in their drawings, especially for man-made objects, to preserve the relative size of parts.

5.3.2 Loss function

Given training sketches of shapes along with the corresponding foreground, depth and normal maps for our output viewpoints, we attempt to estimate the network parameters to minimize a loss function. Our loss function consists of four terms penalizing (a) differences between the training depth maps and predicted depth maps, (b) angle differences between the training normal maps and predicted normal maps, (c) disagreement between ground-truth and predicted foreground mask, (d) large-scale structural differences between the predicted maps and the training maps. Specifically, given T training sketches along with ground-truth foreground, depth and normal maps for our V output viewpoints, our loss function is a combination of the following terms described in the following paragraphs:

$$L = \sum_{t=1}^{T} (\lambda_1 L_{depth}(t) + \lambda_2 L_{normal}(t) + \lambda_3 L_{mask}(t) + \lambda_4 L_{adv}(t)) \qquad (5.1)$$

where $\lambda_1 = 1.0, \lambda_2 = 1.0, \lambda_3 = 1.0, \lambda_4 = 0.01$ are weights tuned in a hold-out validation set.

5.3.2.1 Per-pixel depth and normal loss

The first two terms consider per-pixel differences in the predicted depths and normals with respect to ground-truth. Specifically, we use ℓ_1 distance for depths and angle cosine differences for normal directions. The depth and normal differences are computed only for pixels marked as foreground in the ground-truth:

$$L_{depth}(t) = \sum_{p,v} \left(|d_{p,v}(\mathbf{S}_t) - \hat{d}_{p,v,t}| \right) \hat{f}_{p,v,t} \tag{5.2}$$

$$L_{normal}(t) = \sum_{p,v} \left(1 - \mathbf{n}_{p,v}(\mathbf{S}_t) \cdot \hat{\mathbf{n}}_{p,v,t}\right) \hat{f}_{p,v,t} \tag{5.3}$$

where $\mathbf{S}_t$ is a training sketch, $\hat{d}_{p,v,t}$ and $\hat{\mathbf{n}}_{p,v,t}$ are ground-truth depth and normal for the pixel p in viewpoint v. Each pixel has a ground-truth binary label $\hat{f}_{p,v,t}$, which is 1 for foreground, and 0 otherwise. The depth and normal predictions for the sketch $\mathbf{S}_t$ are denoted as $d_{p,v}(\mathbf{S}_t)$ and $\mathbf{n}_{p,v}(\mathbf{S}_t)$ respectively. We note that all training depths are normalized within the range $[-1, 1]$ while predicted depths are also clamped in this range. Thus both terms above have comparable scale (*i.e.*, both range between $[0, 2]$ per pixel). We also note that we tried ℓ_2 distance for penalizing depth differences but this tended to produce less sharp maps.

5.3.2.2 Mask loss

Penalizing disagreement between predicted and ground-truth foreground labeling can be performed via the cross-entropy function commonly used in classification:

$$L_{mask}(t) = -\sum_{p,v} (\hat{f}_{p,v,t} \log f_{p,v}(\mathbf{S}_t) + (1 - \hat{f}_{p,v,t}) \log(1 - f_{p,v}(\mathbf{S}_t)) \tag{5.4}$$

where $f_{p,v}(\mathbf{S}_t)$ is the probability for the pixel p in viewpoint v to be foreground, as predicted by the decoder.

5.3.2.3 Adversarial loss

We add an adversarial loss [35] to penalize structural differences in the output maps with respect to ground-truth. These have been shown as effective priors for various image-to-image transformation tasks [51]. The adversarial loss term takes as input a 5-channel image $\mathbf{I}_v(\mathbf{S}_t)$ that concatenates the depth channel, the 3 normal channels, and foreground map channel produced by the decoder per viewpoint, and outputs the probability for these maps to be "real":

$$L_{adv}(t) = -\sum_{v} \log P(\text{“real”}|\mathbf{I}_v(\mathbf{S}_t)) \tag{5.5}$$

The probability is estimated using an "adversarial" network trained to discriminate ground-truth ("real") maps $\hat{\mathbf{I}}_{v,t}$ from generated ("fake") maps $\mathbf{I}_v(\mathbf{S}_t)$. Both networks are trained alternatively using the technique of [35]. The adversarial network architecture is the same as the encoder except the last layer that maps the output to probabilities via a fully-connected layer followed by a sigmoid activation.

5.4 Point Cloud and Mesh Generation

Given multi-view depth and normal maps produced by our network at test time, our next goal is to consolidate them into a single, coherent 3D point cloud. The depth and normal predictions produced by the network are not guaranteed to be perfect or even consistent i.e., the derivatives of the predicted depth might not entirely agree with the predicted normals, or the predicted depths for common surface regions across different viewpoints might not yield exactly the same 3D points. Below we discuss an optimization approach to fuse all multi-view depth and normal map predictions into a coherent 3D point cloud, then we discuss mesh generation and post-processing to match the input sketches more precisely. Our optimization approach shares similarities with bundle adjustment and multi-view reconstruction [114, 31]. In our case, our output viewpoints are fixed and we use the normal maps in our energy minimization to promote consistency between depth derivatives and surface normals.

5.4.1 Multi-view depth and normal map fusion

The first step of the fusion process is to map all foreground pixels to 3D points. Each pixel is considered foreground if its predicted probability in the foreground map is above 50%. Given the depth $d_{p,v}$ of a foreground pixel p with image-space coordinates $\{p_x, p_y\}$ in the output map of a viewpoint v, a 3D point $\mathbf{q}_{p,v}$ can be generated according to the known extrinsic camera parameters (coordinate frame rotation $\mathbf{R}_v$ and translation $\mathbf{e}_v$ in object space). Under the assumed orthographic projection, the point position is computed as:

$$\mathbf{q}_{p,v} = \mathbf{R}_v \begin{bmatrix} \kappa p_x & \kappa p_y & d_{p,v} \end{bmatrix}^T + \mathbf{e}_v \tag{5.6}$$

where κ is a known scaling factor, representing the distance between two adjacent pixel centers when their centers are mapped to object space. Each point is also equipped with a normal $\mathbf{n}_{p,v}$ based on the predicted normal map. The result of this first step is a generated point set per view. In a second step, we run ICP [95] to rigidly align all-pairs of point sets.

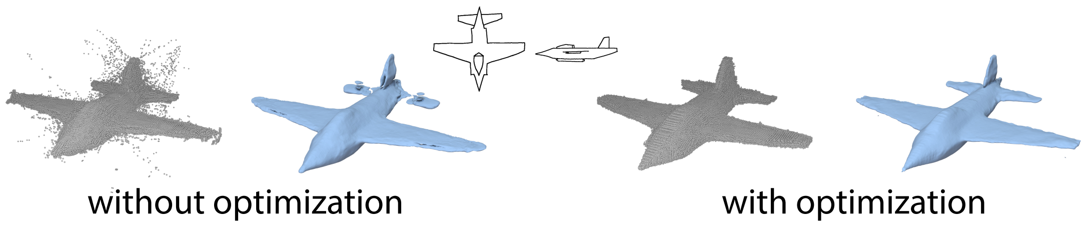


Figure 5.3. Without optimization the noisy point cloud will lead to misaligned regions in the reconstructed shape.

A naive reconstruction method would be to simply concatenate all aligned point sets from all output views into a single point cloud. However, such approach often results in a noisy point cloud with misaligned regions due to inconsistencies in the predicted depth maps. The effect of these inconsistencies tend to be amplified during mesh generation, since a smooth surface cannot pass through all the misaligned regions (Figure 5.3).

Our optimization procedure aims to deal with these inconsistencies. Specifically, we treat the depths of all pixels as variables we want to optimize for. The pixel depths are estimated such that (a) they are close to the approximate predicted depths produced by the network, (b) their first-order derivatives yield surface tangent vectors that are as-orthogonal-as-possible to the predicted normals, (c) they are consistent with depths of corresponding 3D points seen by different viewpoints. These three requirements can be expressed in a single energy over all pixel depths $\mathbf{D} = \{d_{p,v}\}$, with terms imposing the above three conditions:

$$E(\mathbf{D}) = E_{net}(\mathbf{D}) + E_{orth}(\mathbf{D}) + E_{cons}(\mathbf{D}) \tag{5.7}$$

We explain each term in detail in the following paragraphs, then discuss how the energy is minimized.

5.4.1.1 Network prediction term

This energy term penalizes deviation from the approximate depths $\tilde{d}_{p,v}(\mathbf{S}_t)$ produced from the network at each pixel p and viewpoint v:

$$E_{net}(\mathbf{D}) = w_1 \sum_{p,v} (d_{p,v} - \tilde{d}_{p,v}(\mathbf{S}_t))^2 \tag{5.8}$$

where w_1 weights this term (set to 1.0 through hold-out validation). We use ℓ_2 norm here so that the energy minimization yields a linear system that can be solved efficiently.

5.4.1.2 Orthogonality term

The term $E_{orth}(\mathbf{D})$ penalizes deviation from orthogonality between surface tangents, approximated by first-order depth derivatives, and predicted surface normals. Given a 3D point $\mathbf{q}_{p,v}$ generated for the pixel p and viewpoint v, we estimate two surface tangent directions based on the first-order depth derivatives [83], as follows:

$$\mathbf{t}^{(x)}_{p,v} = \begin{bmatrix} \kappa & 0 & \dfrac{\partial d_{p,v}}{\partial x} \end{bmatrix}^T, \quad \mathbf{t}^{(y)}_{p,v} = \begin{bmatrix} 0 & \kappa & \dfrac{\partial d_{p,v}}{\partial y} \end{bmatrix}^T \tag{5.9}$$

The first-order derivatives of the depth can be approximated with a horizontal and vertical gradient filter which is convolved with depths in the 3×3 neighborhood around p. The energy term is expressed as:

$$E_{orth}(\mathbf{D}) = w_2 \sum_{p,v} [(\mathbf{t}^{(x)}_{p,v} \cdot \tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t))^2 + (\mathbf{t}^{(y)}_{p,v} \cdot \tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t))^2] \tag{5.10}$$

where $\tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t)$ is the approximate normal vector produced from the network and w_2 weights this term (set to 1.0 through holdout validation). Since the derivatives are unreliable near the shape silhouette, we omit silhouette points for each view from this term.

5.4.1.3 View consistency term

Given a 3D point $\mathbf{q}_{p,v}$ generated from pixel p at viewpoint v, we can calculate its depth with respect to the image plane of another viewpoint v' as well as the pixel that it is projected onto as: $p' = \Pi_{v'}(\mathbf{q}_{p,v})$, where $\Pi_{v'}$ denotes orthographic projection based on the parameters of viewpoint v'. When the 3D point is not occluded and falls within the image formed at viewpoint v', the calculated depth $d_{v'}(\mathbf{q}_{p,v})$ of that point should be in agreement with the depth $d_{p',v'}$ stored in the corresponding pixel p' of the viewpoint v'. Similarly, the normal of that point $\mathbf{n}_{v'}(\mathbf{q}_{p,v})$ relative to the viewpoint v' should be as-orthogonal-as-possible to the surface tangent vector, approximated by the derivative of the depth stored in the corresponding pixel p'. The view consistency term penalizes: (a) squared differences between the depth at each pixel and the calculated depth of all 3D points projected onto that pixel,

(b) deviation from orthogonality between the surface tangent vector at each pixel and the normal of all 3D points projected onto that pixel. The term is expressed as follows:

$$E_{cons}(\mathbf{D}) = w_3 \sum_{\substack{p,v,p',v': \\ p'=\Pi_{v'}(\mathbf{q}_{p,v})}} (d_{p',v'} - d_{v'}(\mathbf{q}_{p,v}))^2 + w_4 \sum_{\substack{p,v,p',v': \\ p'=\Pi_{v'}(\mathbf{q}_{p,v})}} (\mathbf{t}^{(x)}_{p',v'} \cdot \mathbf{n}_{v'}(\mathbf{q}_{p,v}))^2 + (\mathbf{t}^{(y)}_{p',v'} \cdot \mathbf{n}_{v'}(\mathbf{q}_{p,v}))^2 \tag{5.11}$$

where w_3 and w_4 are weights both set to 0.3.

We note that if a 3D point is projected onto a pixel that is masked as background (thus, its depth is invalid), then we exclude that pixel from the above summation. If the 3D point is projected onto background pixels in the majority of views, then this means that the point is likely an outlier and we remove it from the point cloud. As a result, there are few (p, p') pixel pairs in the above equation: each foreground pixel often has 3-4 corresponding pixels in other views.

5.4.2 Energy minimization

The above energy is quadratic in the unknown pixel depths, thus we can minimize it by solving a linear system. We note that due to the orthogonality term, which involves a linear combination (filtering) of depths within a pixel neighborhood, the depth of each pixel cannot be solved independently of the rest of the pixels. The solution can be computed through a sparse linear system - we provide its solution in Appendix C.1.

When we estimate the pixel depths, the corresponding 3D point positions, generated by these pixels, are updated. Given new 3D point positions, the consistency term also needs updating since the points might now be projected onto different pixels. This gives rise to an iterative minimization scheme, where at each step we estimate pixel depths by solving the linear system, then update the 3D point positions. We observed that the depth estimates become increasingly consistent across different views at each iteration, and practically we observe convergence after 3-5 iterations. The resulting point cloud yields a much smoother reconstructed surface, as shown in Figure 5.3.

5.4.3 Mesh reconstruction and fine-tuning.

We apply the Screened Poisson Surface Reconstruction algorithm [59] to convert the resulting point cloud and normals to a surface mesh. After mesh generation, our method can optionally further "fine-tune" it so that it matches the input contours more precisely. Specifically, for each input line drawing, we first extract its external contours and discretize them into a dense set of 2D points. Then for each input view, we render the mesh under the same orthographic projection, and find nearest corresponding mesh points to each contour point under this projection. Then we smoothly deform the 3D mesh, such that the projected mesh points move towards the contour points under the constraint that the surface Laplacians [82], capturing any underlying surface details, are preserved. We also deform the mesh so that it better matches the internal contours of the sketch. To do this, we find nearest corresponding mesh points to each internal contour point and scale their Laplacian according to the scheme suggested in [82]. Mesh deformation is executed by solving a single sparse linear system involving all constraints from all internal and external contours across all input views. Figure 5.1 shows a reconstructed mesh before and after fine-tuning.

5.4.4 Implementation.

The network is implemented in Tensorflow [1]. Training takes about 2 days for 10K training meshes (40K training sketches) on a TitanX GPU. We use the Adam solver [62] (hyperparameters $\beta 1$ and $\beta 2$ are set to 0.9 and 0.999 respectively). At test time, processing input sketches through the network takes 1.5 sec on a TitanX GPU, fusing the depth and normal maps takes 3 sec, mesh reconstruction and fine-tuning takes about 4 sec (fusion and mesh reconstruction are implemented on the CPU - running times are reported here for a dual Xeon E5-2699v3). In total, it takes our method less than 10 seconds to present a reconstructed mesh to the user. Our data and source code is available on our project page: `https://people.cs.umass.edu/~zlun/papers/SketchModeling`.

5.5 Evaluation

We now discuss the experimental evaluation and analysis of our method.

5.5.1 Datasets

To train our network, we gathered three collections of 3D shapes along with their synthetic sketches. Each of the collections included shapes belonging to the same broad category. The categories were 3D computer characters, airplanes, and chairs. To create the 3D computer character collection, we downloaded freely available 3D models of characters from an online repository ("The Models Resource" [2]). The collection contained humanoid, alien, and other fictional 3D models of characters. The airplanes and chairs originated from the 3D ShapeNet [13]. We used these particular categories from ShapeNet because the shapes in these categories have large geometric and structural variation. Table 5.1 reports the number of training shapes and views used to generate the training sketches.

	#training shapes	view A	view B
Character	10000	front	side
Airplane	3667	top	side
Chair	9573	front	side

Table 5.1. Training dataset statistics.

5.5.2 Test dataset

To evaluate our method and compare it with alternatives, we created a test dataset of synthetic and human line drawings for each of the above categories. Each line drawing was created according to a reference test shape. The goal of the evaluation was to examine how well the reconstructed 3D shapes from these test line drawings matched the reference test shapes. To execute a proper evaluation, the reference test shapes should be sufficiently different from all training shapes. Otherwise, by overfitting a network to the training dataset or by simply using a nearest neighbor sketch-based retrieval approach, one could perfectly reproduce the reference shapes. To create the test dataset of reference shapes, one option would be to randomly split the above collections into a training and test part. However, a problem with this strategy is that several test shapes would be extremely similar to one or more training shapes because of duplicate 3D models that often exist in these collections (i.e., models that are identical up to an affine transformation, having tiny part differences, or only different mesh resolution). To create our test dataset, we found 120 shapes (40 per category) in our collections that we ensured to be sufficiently different from the shapes used for training by performing two checks. First, for each shape, we aligned it to each other shape in the collection through the best matching affine transformation and compute their Chamfer distance. The Chamfer distance is computed by measuring the distance of each of the points on one shape to the nearest surface point on the other shape, then the average of these distances is used (we sampled 10K points uniformly per shape). We verified that the Chamfer distance between each test shape and its nearest training shape is well above a threshold. Second, we rendered synthetic sketches for each shape based on the input views per category and extracted the representation from our encoder for these sketches. We then retrieved the nearest other shape based on Euclidean distance over the sketch representations. We verified that the distance is well above a threshold. We also visually confirmed that test and training shapes were different and the selected thresholds were appropriate.

For our 120 test shapes, we produced synthetic sketches for 90 of them (30 per category), and gathered human line drawings for the remaining 30 shapes (10 per category). Synthetic sketches were produced from the test shapes using the line rendering techniques described in Section 5.3 based on the input views A and B per category (Table 5.1). The human sketches were produced by asking two artists to provide us with hand-drawn line drawings of reference test shapes. The test shapes were presented to the artists on a computer display and were rendered using Phong shading. Their views were selected to approximately match the input views A and B per category. We asked the artists to create on paper line drawings depicting the presented shapes based on the selected views. We then scanned their line drawings, cropped and scaled them so that the scanned drawn area matches the drawing area of training sketches on average. We note that in contrast to synthetic sketches, human line drawings might not be consistent across different views.

5.5.3 Evaluation measures

Given the above test sketches as input, the goal of our evaluation is to measure how well the 3D shapes reconstructed by various methods, including ours, matched the reference test shapes used to produce these sketches. Our method and the alternatives, listed in the following paragraphs, were trained and tested separately on each shape category using the same splits. We used five evaluation measures to compare the reconstructed shapes to the reference ones: Chamfer point-based distance, Hausdorff point-based distance, surface normal distance, depth map error, volumetric Jaccard distance. The Chamfer distance is computed by measuring the distance of each of the points on the reconstructed shape to the nearest surface point on the reference shape, then computing the average of these distances; the Hausdorff distance computes the maximum instead of the average of these distances. The surface normal distance is computed by measuring the angle between the surface normal at each point on the reconstructed shape and the surface normal at the nearest surface point on the reference shape, then computing the average of the angles. The depth map error is computed by measuring the absolute differences between pixel depths in each of the output depth maps produced by our network and the corresponding depth maps of the reference shape, then computing the average depth differences. To compute the volumetric Jaccard distance, we voxelized the reconstructed and reference shapes in a $128 \times 128 \times 128$ binary grid and measured the number of voxels commonly filled in both shapes (intersection of their volume) divided by the number of filled voxels for the two shapes (union of their volumes). This is the Intersection over Union (IoU). We use $1 - IoU$ to get the volumetric Jaccard distance.

5.5.4 Comparisons with baselines

We tested the reconstructions produced by our method (called "ShapeMVD") versus the following methods: (a) a network based on the same encoder as ours but using a volumetric decoder baseline instead of our multi-view decoder, (b) a network based on the same encoder as ours but with the Tatarchenko *et al.*'s view-based decoder [110] instead of our multi-view decoder, (c) the convolutional 3D LSTM architecture (R2N2) provided by Choy *et al.*'s implementation [17], and (d) nearest sketch-based shape retrieval.

For the volumetric decoder baseline (a), we used a $128 \times 128 \times 128$ output binary grid (the maximum we could fit in 12GB GPU memory). To make sure that the comparison is fair, we set the number of parameters in the volumetric decoder such that it is comparable to the number of parameters in our decoder. The volumetric decoder consisted of five transpose 3D convolutions of stride 2 and kernel size $4 \times 4 \times 4$. The number of filters starts with 512 and is divided by 2 at each layer. Leaky ReLU functions and batch normalization were used after each layer. We note that we did not use skip-connections (U-net architecture) in the volumetric decoder because the size of the feature representations produced in the sketch image-based encoder is incompatible with the ones produced in the decoder. For Tatarchenko *et al.*'s method, the viewpoint is encoded into a continuous 64×1 representation passed as input to the view-based decoder described in [110] without separate branches. To ensure a fair comparison, we increased the number of filters per up-convolutional layer by a factor of 3 so that the number of parameters in their and our decoder is comparable. We also train it with the same loss function as ours. We additionally implemented a variant of Tatarchenko *et al.*'s decoder by adding U-net con-

	Man-made objects (synthetic)						Character models (synthetic)					
	Shape MVD	nearest retrieval	Tatarchenko et al.[110]	[110]+ U-net	volumetric decoder	R2N2 [17]	Shape MVD	nearest retrieval	Tatarchenko et al.[110]	[110]+ U-net	volumetric decoder	R2N2 [17]
Hausdorff distance	**0.092**	0.165	0.142	0.121	0.113	0.144	**0.089**	0.200	0.119	0.092	0.152	0.148
Chamfer distance	**0.015**	0.025	0.022	0.017	0.021	0.026	**0.015**	0.036	0.025	0.016	0.026	0.032
normal distance	**30.66**	42.57	35.58	32.32	49.40	48.78	**30.61**	44.93	34.98	31.00	53.84	53.13
depth map error	**0.026**	0.049	0.039	0.030	0.038	0.045	**0.018**	0.040	0.030	0.019	0.031	0.036
volumetric distance	**0.344**	0.501	0.442	0.374	0.432	0.512	**0.313**	0.541	0.428	0.329	0.437	0.493

	Man-made objects (human drawing)						Character models (human drawing)					
	Shape MVD	nearest retrieval	Tatarchenko et al.[110]	[110]+ U-net	volumetric decoder	R2N2 [17]	Shape MVD	nearest retrieval	Tatarchenko et al.[110]	[110]+ U-net	volumetric decoder	R2N2 [17]
Hausdorff distance	**0.116**	0.176	0.153	0.153	0.130	0.149	**0.117**	0.188	0.139	0.136	0.178	0.168
Chamfer distance	**0.017**	0.031	0.024	0.025	0.022	0.028	**0.021**	0.036	0.025	0.024	0.032	0.036
normal distance	**27.04**	40.96	32.40	30.45	48.32	48.12	**33.44**	43.81	36.11	34.74	54.91	54.29
depth map error	**0.021**	0.042	0.033	0.032	0.032	0.042	**0.026**	0.040	0.031	0.027	0.037	0.040
volumetric distance	**0.311**	0.544	0.405	0.403	0.405	0.500	**0.298**	0.458	0.342	0.307	0.420	0.436

Table 5.2. Comparisons of our method with baselines based on our evaluation measures (the lower the numbers, the better)

nections between the encoder and their decoder. We report the evaluation measures on this additional variation. For the nearest-neighbor baseline, we extract the representation of the input test sketches based on our encoder. This is used as a query representation to retrieve the training shape whose sketches have the nearest encoder representation based on Euclidean distance. All methods had access to the same training dataset per category and were evaluated on the same test sketches (two input sketches per test shape).

Table 5.2 reports the evaluation measures for all competing methods based on both synthetic and human line drawings. We include evaluation separately for organic shapes (3D character collection) and man-made shapes (measures are averaged over airplanes and chairs). Our method produces much more accurate reconstructions than the competing methods in all cases. With respect to Tatarchenko *et al.*'s method, we find that its enhancement with U-net connections improves its performance, but still performs worse than our method, especially for man-made objects. This implies that U-net is a significant enhancement. We finally observe that the R2N2 does not perform better than our volumetric decoder baseline.

Figure 5.4 shows representative input test sketches, and output meshes for competing methods. In general, the nearest neighbor results look plausible because retrieval returns human-modeled training shapes with fine details (e.g., facial features). Such details are not captured by any of the methods, including ours. On the other hand, as shown in the figure, and confirmed by numerical evaluation, compared to nearest neighbor retrieval and other methods, ours produces shapes that better match the input sketch. The main reason is that our method better preserves the shape structure, topology and coarse geometry depicted in the input sketch.

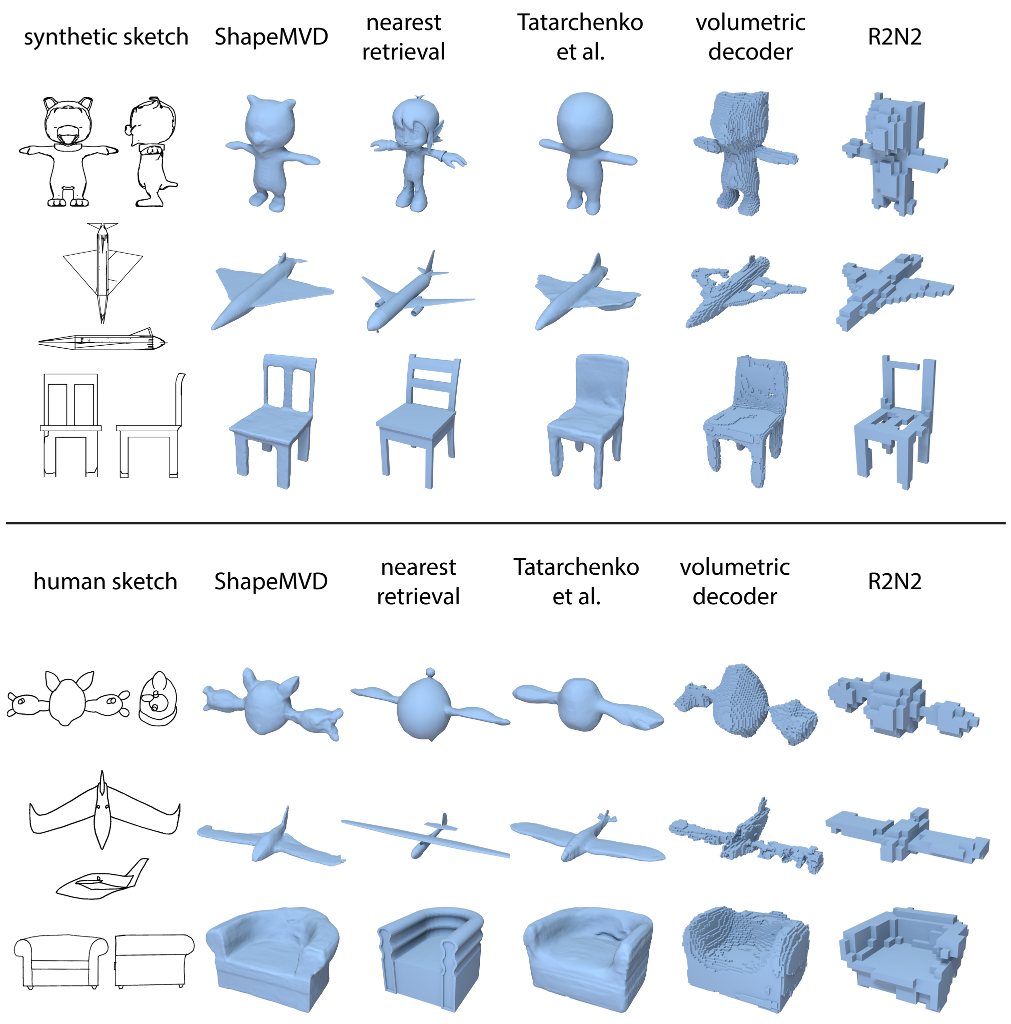


Figure 5.4. Comparisons of shape reconstructions from sketches for our method and baselines.

We note that mesh fine-tuning was not used here for any methods. The reason was to evaluate the methods by factoring out the post-processing effects of fine-tuning. Fine-tuning is optional and does not significantly affect the errors. It is used only to add details ("stylize") the produced meshes based on the input contours when these are precisely drawn, and if users desire so. "Fine-tuning" can be applied not only to the reconstructed meshes of our method but also to the resulting meshes of the other competing methods. Thus, we also experimented when fine-tuning is applied to the results of all methods. We found that the effect on evaluation measures tends not to be significant and our method has still much smaller errors than the others also in this case. The reason is that the mesh deformation applied during fine-tuning works well only if the produced shape matches the drawn shape in terms of structure and topology (e.g., layout and number of parts). While this is mostly true for our method, it is often not the case for shapes produced by volumetric decoders and nearest retrieval. For example, given the line drawing of a chair with a vertical middle bar on its back (Figure 5.4, top), the chair returned by nearest retrieval has a horizontal bar instead. Fine-tuning cannot add or remove parts, but instead deforms irrelevant surface points on the retrieved chair back towards the silhouette points of the vertical bar, yielding a largely implausible shape. Due to such mismatches, fine-tuning the retrieved shapes can slightly amplify errors with respect to ground-truth shapes. For example, for human line drawings, Hausdorff distance is further increased by 10% for nearest retrieval when fine-tuning is applied to the retrieved shapes. In contrast, for our method after fine-tuning, the error drops by a tiny amount ($< 1\%$) i.e., deformation adds small details, like the alien's eyes of Figure 5.1, without causing implausible deformations.

5.5.5 Comparisons with variants of our method

We also evaluated the reconstructions produced by our full method against degraded variants of it. Table 5.3 reports the results. Specifically, we tested the following variants: (a) we do not use the optimization procedure of Section 5.4 ('no fusion' column), (b) we set

	Man-made objects					Character models				
	full method	no fusion	no normal	no GAN	single input	full method	no fusion	no normal	no GAN	single input
Hausdorff distance	**0.092**	0.102	0.108	0.107	0.134	0.089	0.090	**0.088**	0.098	0.113
Chamfer distance	**0.015**	**0.015**	0.017	0.016	0.020	**0.015**	**0.015**	0.016	0.016	0.021
normal distance	**30.66**	30.78	31.22	30.89	34.49	**30.61**	30.84	30.85	30.72	34.15
depth map error	**0.026**	0.027	0.029	0.028	0.035	**0.018**	0.019	0.020	0.019	0.026
volumetric distance	**0.344**	0.356	0.354	0.347	0.428	**0.313**	0.318	0.323	0.320	0.396

Table 5.3. Comparisons with variants of our method based on our evaluation measures (the lower the numbers, the better).

the output of our network to depth alone ('no normal' column). Since Poisson reconstruction requires both points and normals as input, we produce normals by least-squares plane fitting for each generated 3D point, (c) we skip the adversarial loss term during training ('no GAN' column). For all these variants, the network uses two input sketches based on views A and B of Table 5.1. We also tested the reconstructions produced by our method when it uses a single sketch as input (view A, 'single input' column in Table 5.3) versus two sketches as input (views A and B). We note that mesh fine-tuning was not used for any of these variants. Based on the resulting numbers, our full method tends to produce lower errors than its degraded variants, especially for man-made objects that often have more structural and geometric variability than character models. We also observe that using two sketches significantly improves the reconstructed shapes. This is not surprising since two input sketches contain more shape information than one.

5.5.6 Perceptual user study

In addition to the above numerical evaluation measures, we also performed a perceptual user study to compare our method with the volumetric decoder, view-based decoder based on Tatarchenko *et al.* [110] and the nearest neighbor sketch-based retrieval. The user study was executed through the Amazon Mechanical Turk (MTurk) service. Each questionnaire included 30 queries. Each query showed: (a) a pair of synthetic or human line drawings depicting a test shape from two different views, (b) a rendered image of the 3D surface mesh

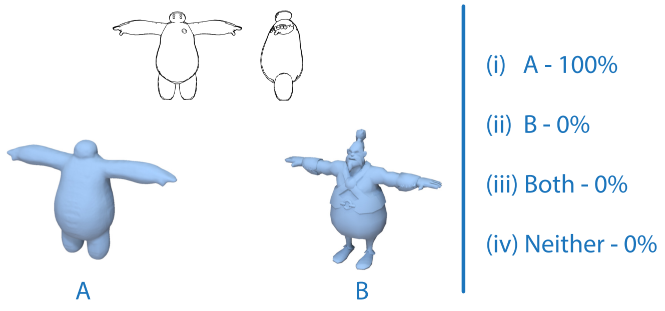


Figure 5.5. Query layout shown to participants of our user study.

reconstructed using our method given these two input line drawings, (c) another rendered image of the 3D surface mesh reconstructed using one of the alternative methods. The images were laid out as shown in Figure 5.5. Queries were shown at a random order, while each page was repeated twice (i.e., 15 unique queries), with the two rendered mesh images randomly flipped, to detect unreliable users giving inconsistent answers. Each query included the following question: "Which of the two 3D models on the bottom (A or B) is MORE similar to the object depicted by the line drawings on the top? ". Participants were asked to pick one of the following answers: "(i) A, (ii) B, (iii) can't tell - Both A and B look equally similar to the line drawings, (iv) can't tell - Neither A nor B looks similar to the line drawings". To avoid any individual bias, we allowed each participant to complete only one questionnaire per category. Participants were rewarded $\$1$ for each questionnaire completion. Each query was answered by 5 different, reliable MTurk participants. We filtered out unrealiable MTurk participants who gave two inconsistent answers to more than 7 out of the 15 unique queries in the questionnaire, or took less than 2 minutes to complete it. Participants agreed with each other 89.0% of the times, indicating a high degree of consistency across participants.

In total, we gathered 1800 consistent responses from reliable users: 600 responses comparing the reconstructions of our method with the ones from the volumetric decoder, 600 responses comparing our method with sketch-based retrieval, and 600 responses compar-

	plurality					raw votes			
	A	B	both	neither	draw	A	B	both	neither
ours (A) vs Tatarchenko et al. (B)	99.2%	0.8%	0.0%	0.0%	0.0%	94.7%	2.5%	1.5%	1.3%
ours (A) vs volumetric decoder (B)	96.7%	1.7%	0.0%	0.0%	1.7%	92.8%	2.0%	3.0%	2.2%
ours (A) vs nearest retrieval (B)	87.5%	12.5%	0.0%	0.0%	0.0%	81.2%	14.7%	1.0%	3.2%

Table 5.4. Perceptual user study results comparing our method with baseline methods: per-query plurality responses (left) and raw vote percentages (right).

ing our method with the alternative view-based decoder based on [110]. The 600 query responses were gathered for all 120 human and synthetic test sketches in all our 3 categories (as explained above, each test sketch pair and resulting reconstructions was examined by 5 different, reliable MTurk participants).

Table 5.4 reports the results of the user study. We report the percentage of plurality responses per-query (plurality is formed by the 5 reliable users per query). Our method was found to produce shapes that look much more similar to the depicted shapes in the line drawings.

5.5.7 More results

Figure 5.6 shows reconstructed shapes produced by our method for various input synthetic and human sketches. Fine-tuning was used for the meshes of this figure.

5.6 Conclusion

We presented an approach for 3D shape reconstruction from sketches. Our method employs a ConvNet to predict depth and normals from a set of viewpoints, and the resulting information is consolidated into a 3D point cloud via energy minimization. We evaluated our method and variants on two qualitatively different categories (characters and man-made objects). Our results indicate that view-based reconstruction of a 3D shape is significantly more accurate than voxel-based reconstruction. We also showed that our method can gen-

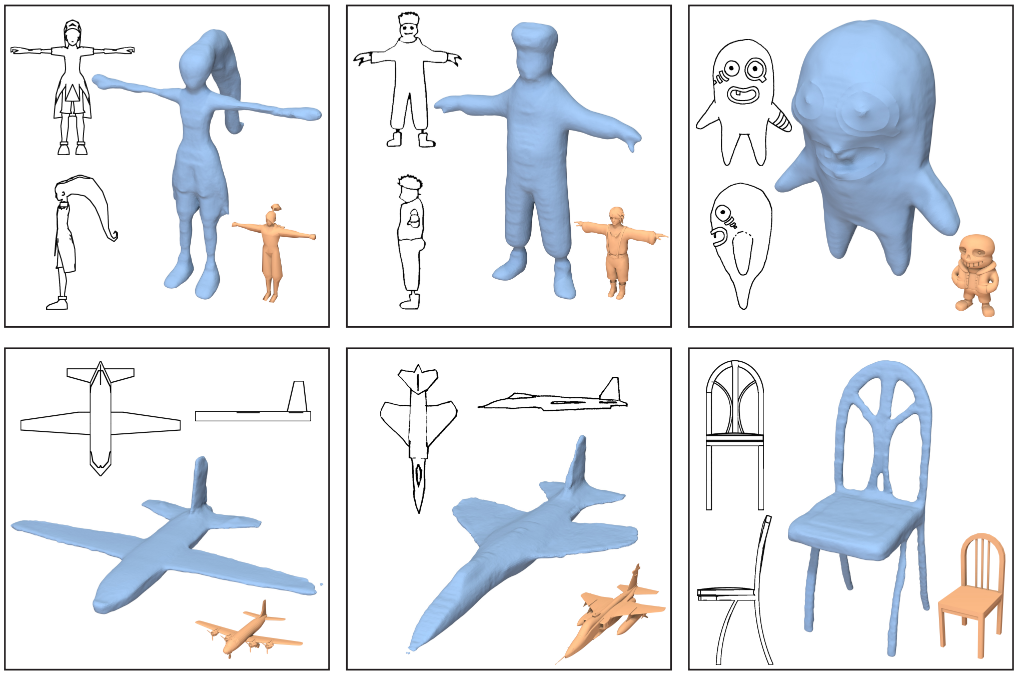


Figure 5.6. Gallery of results. Blue shapes represent reconstructions produced by our method from the input sketches. Orange shapes are the nearest shapes in the training datasets retrieved via sketch-based retrieval.

eralize to human-drawn sketches. We believe that there is significant room for improving our method in the future. For example, it would be interesting to explore the possibility of incorporating the fusion process in the network, and modifying its architecture such that reconstruction is done from arbitrary viewpoints. Our reconstructed shapes often lack fine details that users would prefer to see in production-quality 3D models. We believe that these shapes can serve as starting "proxies" for artists to improve upon through modeling interfaces. From this aspect, it would be useful to integrate interactive modeling techniques into our method.

CHAPTER 6

DISCUSSION AND FUTURE WORK

In this dissertation we have discussed algorithms to analyze stylistic properties in 3D shapes and algorithms to automatically synthesize shapes given style specifications in the form of another shape in a particular style or human line drawing inputs.

We have introduced the first method for evaluating stylistic similarity between structurally and functionally different objects. We quantified style similarity criteria into geometric properties following observations from art history literature. We utilized crowdsourced data to relate geometric features to actual human perception of style and employed relative comparison to learn our style similarity measure. We experimentally showed that our learned style similarity measure is well aligned with human perception of style.

We have also described a new algorithm for transferring style between structurally and functionally different man-made objects. We introduced a new compatibility measure for preserving functionality without explicitly identifying functional parts. We validated our style transfer framework via extensive user studies and showed that our method is able to generate functionally plausible and stylistically similar models in a wide range of shape classes.

We have also presented a system for reconstructing 3D shapes from 2D line drawings. We proposed the first approach to use a learned, view-based representation for generating shapes from sketches. We trained the framework using mass amount of synthetic data without the supervision from human line drawings data. We validated our algorithm quantitatively using standard evaluation metrics between our results and those from baseline methods and variants of our method. We also validated our outputs qualitatively via a perceptual user study and showed that our method can generate plausible 3D shapes from 2D sketch inputs.

6.1 Future Work

There are many exciting directions for future work. While we put significant effort into exploring geometric features and elementary distances relevant for visual motif and consequently style analysis, it remains an open question if the features discussed in Chapter 3 are sufficient to compare the style of shapes. In particular, for large structures, such as buildings, the overall arrangement of parts and elements is likely to play some role in style parsing. Instead of designing features and distances from scratch, it could be interesting to explore if these can be learned directly from raw shape data. Deep learning architectures could be used for this purpose, as well as for learning more advanced models of style similarity [75]. Our work focuses on similarity within broad object categories, such as between pieces of furniture, or buildings, where stylistic commonalities are most obvious; it may be interesting to consider cross-category style evaluation between objects, e.g. evaluating style similarity between buildings and furniture. The first step for such a task would be to evaluate how consistent humans are at this task. In [78] they introduced a method to evaluate style compatibility for furniture based on co-segmented shapes. We speculate that combining feature-based joint segmentation [47] or template fitting methods [61] with our alignment-based element matching technique could further improve our style similarity measure for some classes.

Our style transfer algorithm requires as input an exemplar 3D model that represents the desired style to be transferred to other objects. It would be interesting to explore other input modalities that describe style. Our work on reconstructing 3D shapes from 2D sketches provide an idea to encode style in line drawings. There are also other possible representations for style specification such as natural language input. Furthermore, our style transfer algorithm leverages the structure of a target shape, either specified manually or retrieved automatically, to synthesize new shapes. Instead of relying on a pre-existing target shape structure, it would be interesting to employ generative models that are capable of generating plausible shape structure and accurate surface geometry automatically. Such models could also avoid the need of slot-based part alignment that may fail when slots largely differ in number, size and orientation. Another interesting direction would be to combine our structure-based functional compatibility metric with functionality models [60, 42] that consider part interactions with agents and other objects in a scene to improve correspondences especially for parts where such interactions are meaningful.

Our framework for reconstructing 3D shapes from 2D sketches has an optimization step to fuse shape information from network outputs. This step requires a significant amount of parameter tunings (point cloud filtering thresholds, surface reconstruction algorithm parameters, curve-based deformation parameters, etc.). It would be better to incorporate this step into the network and learn all parameters in a unified optimization framework. Currently following our workflow, the framework should have pre-trained networks for all possible permutation of canonical input views, which is cumbersome in a usable application. A more advanced model such as a recurrent module which is invariant with the order of user input views may bypass the need for specifying the input views and can be trained as one single network. Furthermore, it would be even better not to put any assumptions on the input views. Recent work on view estimation [106] may give an insight on eliminating the assumption that the input view is known to the network.

APPENDIX A

LEARNING PERCEPTUAL SHAPE STYLE SIMILARITY

A.1 Shape and part features

In this section we describe the features used in the style similarity measure formulation in Section 3.2.

A.1.1 Elementary distance

We describe here the elementary distances we used for measuring geometric similarity between elements (Section 3.2.1). In total, we used 77 elementary distances. To compute them, first we uniformly sample the surfaces of the shapes with $20K$ points, so that the distances are invariant to mesh artifacts. Then we compute the elements' surface distance, by aligning them with ICP and including the average closest point-to-point distance and average distance between their normals as our 2 first elementary distances. Then we compute the curvature tensors for each point on the element surface and extract 13 feature values (min/max curvature by value, min/max curvature by magnitude, mean curvature, Gaussian curvature, the absolute value of the aforementioned six features, as well as the mean magnitude of the two principal curvatures). We compute histograms of those 13 curvature features with $16, 32, 64, 128$ bins for each element. We also compute histograms of the elements' shape diameter [102] with $16, 32, 64, 128$ bins, and the D2 shape distribution histograms [89] with $16, 32, 64, 128$ bins. On each of those curvature, shape diameter and D2 histograms, we measure the Earth Mover's Distances (4×15 elementary distances). We then extract the following feature curves on the elements: boundaries, ridges and valleys lines. Using the same element alignment we got from ICP, we compute the average closest curve point-to-point and normal distances for each of the three types of feature curves separately and for the whole set of curves (4×2 elementary distances). We extract the silhouette of the aligned shapes under different viewpoints [15], compute their Zernike moments, Fourier coefficients, eccentricity, circularity and estimate their euclidean distance for each of them (4 elementary distances). Finally, we use the axis-aligned bounding box scales of the aligned shape features and we measure their absolute differences along three axes (3 elementary distances). Finally, all distances are scaled to have unit variance across all training shape pairs.

A.1.2 Features for elementary saliency

We describe here the geometric features that were used for measuring element saliency (Section 3.2.3, see Equation 3.9) and the prevalence of matching elements (see Equation 3.11). In total, we gathered 20 geometric features in our element saliency measure. All geometric features are computed on the sample points of the elements' surface. First we used the height of the sample point and its horizontal distance to shape center. The metrics of height and horizontal distance are relative to the bounding box size of the shape (2 saliency features). We also compute the geodesic distance from each point to all other points and use the average geodesic distance as feature (1 saliency feature). We also compute the ambient occlusion for each point by shooting rays towards the hemisphere along its normal direction and counting the percentage of rays which do not intersect with the shape (1 saliency feature). Similarly to the curvature-related elementary distances, we include the absolute values of min/max curvature by value, the absolute values of min/max curvature by magnitude, the absolute value of the mean curvature and Gaussian curvature, as well as the mean magnitude of two principal curvatures (7 saliency features). Following the distinctness idea in [103], we compute histograms of various features and use the dissimilarity of the histograms between neighboring points as saliency features. Besides the

furniture (100 triplets)	MTurk	Expert	Casual
number of users	20	5	32
% consistency	88.2%	98.2%	94.9%
% consistency (i) vs (ii)	98.9%	99.5%	98.9%
building (100 triplets)	MTurk	Expert	Casual
number of users	20	5	32
% consistency	87.0%	92.1%	92.9%
% consistency (i) vs (ii)	97.5%	99.2%	99.4%
cutlery (50 triplets)	MTurk	Expert	Casual
number of users	15	5	32
% consistency	92.7%	95.7%	92.0%
% consistency (i) vs (ii)	99.6%	100.0%	100.0%

Table A.1. Pilot study statistics.

original Simplified Point Feature Histogram whose bins count relative angular directions of the normals, we also compute spin images [53] and 3D shape contexts histograms based on [57]. To measure distinctness among different range of contextual shape information, we use 3 levels of neighbor ranges (3×3 saliency features). Note that all of the saliency features above are calculated on points and the saliency of an element or a region is a sum of the point saliency which implicitly accounts for the area of the element or the region. Finally, all saliency features are shifted and scaled to have zero mean and unit variance across all training shapes.

A.2 Extra Study Statistics

Table A.1 summarizes the results of the pilot study per participant category.

APPENDIX B

FUNCTIONALITY PRESERVING SHAPE STYLE TRANSFER

B.1 Descriptors

In this section we describe the descriptors used in the element compatibility formulation in Section 4.3.

B.1.1 Per-Element descriptors

In total, we have 13 element descriptors yielding 13 distance measures. The first set of descriptors capture the relative location of the markers on an element, including its centroid, its centroid projected on the ground plane, then its highest point, lowest point, and its centroid projected onto the upright axis (i.e., representing height from the ground plane). When comparing nodes in our graph, element locations are expressed with respect to the object's coordinate system. When comparing edges in our graph, element locations are expressed with respect to the local corresponding system of its neighboring element in the graph (the local coordinate system is formed by the neighboring element's corresponding marker locations and object's axes). Each of the five relative locations yields a Euclidean distance when comparing two elements. The next three descriptors store the proportions of the element's axis-aligned bounding box, relative to the object's bounding box proportions when comparing nodes, and relative to the neighboring element's bounding box proportions when comparing edges. These proportions (one per each axis) yield three more distances. The next three descriptors are similar to the previous three, but instead of the bounding box proportions, we use the variance of the element point positions along the object's axes. The next descriptor stores the major orientation of the element estimated via PCA. When comparing nodes, we measure the angle difference between the major orientations of the two corresponding elements. When comparing edges, we measure the relative angle difference between the major orientations of the two corresponding elements with respect to their neighboring elements major axes. The last descriptor is a histogram that approximately captures the distribution of point samples in an element. We build a $4 \times 4 \times 4$ grid and compute a histogram by counting how many sample points on the element surface are inside each bin. When comparing nodes, we compute the euclidean distance between the histograms for the corresponding elements. When comparing edges, we compute a histogram for each edge measuring the absolute difference of bin values between the corresponding histograms of neighboring elements, then measure the euclidean distance between the resulting histograms.

B.1.2 Curve descriptors

In total, we get 3 distance measures between curve descriptors. The first one represents distance between centroids of the two curves and the second one represents differences between their arc lengths. The last one represents the average point-to-point distance after aligning the two input curves via ICP.

B.2 Gradient for learning

Learning the compatibility metric requires computing the analytic gradient of our objective function (Equation 4.13) with respect to our parameters. The loss function evaluates the compatibility metric, which is defined through the recursive formula of Equation 4.1. Interestingly, it turns out that the gradient also follows a similar recursive definition, which makes it possible to compute it efficiently. For clarity, we provide here the formulas that evaluate the partial derivatives of our objective function with respect to the node kernel parameters $\{w_k\}_{k=1 \ldots K}$ and RBF variances $\{\sigma_k\}_{k=1 \ldots K}$. The partial derivatives for the rest of the parameters follow a similar recursive computation. We begin by computing the gradient of the loss function with respect to the node kernel parameters:

$$\frac{\partial L(\mathbf{w}, \boldsymbol{\sigma})}{\partial w_k} = -\lambda \cdot \text{sign}(w_k) + \sum_{t=1}^{T} \frac{\partial \ln P(p_q[t] \triangleright p_r[t])}{\partial w_k} \tag{B.1}$$

The gradient of the log likelihood per training example t can be expressed as:

$$\frac{\partial \ln P(p_q[t] \triangleright p_r[t])}{\partial w_k} = \Big(1 - \sigma\big(D_{func}(p,r) - D_{func}(p,q)\big)\Big) \cdot \Big(\frac{\partial D_{func}(p,r)}{\partial w_k} - \frac{\partial D_{func}(p,q)}{\partial w_k}\Big) \tag{B.2}$$

The partial derivatives of the distance function $D_{func}(p,r)$, and similarly for $D_{func}(p,q)$, are in turn computed as:

$$\frac{\partial D_{func}(p,q)}{\partial w_k} = \frac{\big(\frac{\partial K^{(n)}(p,p)}{\partial w_k} - 2\frac{\partial K^{(n)}(p,q)}{\partial w_k} + \frac{\partial K^{(n)}(q,q)}{\partial w_k}\big)}{2 D_{func}(p,q)} \tag{B.3}$$

The above formula requires computing partial derivatives of our graph-based compatibility function. The derivatives also follow a recursive definition :

$$\begin{aligned} \frac{\partial K^{(n)}(p,q)}{\partial w_k} = {} & \frac{\partial K_{node}(p,q)}{\partial w_k} \cdot \sum_{\substack{p' \in \mathcal{N}(p) \\ q' \in \mathcal{N}(q)}} K_{edge}(e_{pp'}, e_{qq'}) \cdot K^{(n-1)}(p', q') \\ & + K_{node}(p,q) \cdot \sum_{\substack{p' \in \mathcal{N}(p) \\ q' \in \mathcal{N}(q)}} K_{edge}(e_{pp'}, e_{qq'}) \cdot \frac{\partial K^{(n-1)}(p', q')}{\partial w_k} \end{aligned} \tag{B.4}$$

To evaluate the above formula, the partial derivatives of the node similarity functions with respect to the kernel node parameters are required. These are computed as follows:

$$\frac{\partial K_{node}(p,q)}{\partial w_k} = \exp\left\{ -\frac{D_k^2(p,q)}{2\sigma_k^2} \right\} \tag{B.5}$$

Computing the partial derivatives of our objective function with respect to the RBF variances follow the same procedure as above with only two differences: the sign term in Equation B.1 is omitted (no L^1-norm regularization is used for variances since sparsity is not required for them) while the partial derivatives of the kernel node functions are instead expressed as follows:

$$\frac{\partial K_{node}(p,q)}{\partial \sigma_k} = \frac{w_k D_k^2(p,q)}{\sigma_k^3} \exp\left\{ -\frac{D_k^2(p,q)}{2\sigma_k^2} \right\} \tag{B.6}$$

APPENDIX C

SHAPE RECONSTRUCTION FROM SKETCHES

C.1 Solution to the linear system for point cloud optimization

To minimize the energy $E(\mathbf{D})$ formulated in Section 5.4, we set its derivatives with respect to the unknown pixel depths $\mathbf{D}$ to zero, which in turn leads to a sparse linear system in the form of $\mathbf{Ax} = \mathbf{b}$. Here the unknown vector $\mathbf{x}$ consists of all pixel depths $d_{p,v}$ we wish to solve for. The system is solved using the conjugate gradient method in least-squares sense. The following Equation C.1 shows the linear system along with the sparse matrix $\mathbf{A}$ and the constant vector $\mathbf{b}$. In the following paragraphs, we explain how to derive the system based on the linear constraints originating from each of the energy terms explained in Section 5.4.1.

$$\begin{bmatrix} w_1 I \\ \ldots \\ \left(w_2 \cdot \tilde{\mathbf{n}}^{(z)}_{p,v}(\mathbf{S}_t)\right) \mathbf{L}^{(x)} \\ \ldots \\ \left(w_2 \cdot \tilde{\mathbf{n}}^{(z)}_{p,v}(\mathbf{S}_t)\right) \mathbf{L}^{(y)} \\ \ldots \\ w_1 I \\ \ldots \\ \left(w_2 \cdot \mathbf{n}^{(z)}_{v'}(\mathbf{q}_{p,v})\right) \mathbf{L}^{(x)} \\ \ldots \\ \left(w_2 \cdot \mathbf{n}^{(z)}_{v'}(\mathbf{q}_{p,v})\right) \mathbf{L}^{(y)} \\ \ldots \end{bmatrix} [\mathbf{D}] = \begin{bmatrix} w_1 \cdot \tilde{d}_{p,v}(\mathbf{S}_t) \\ \ldots \\ -w_2 \cdot \kappa \cdot \tilde{\mathbf{n}}^{(x)}_{p,v}(\mathbf{S}_t) \\ \ldots \\ -w_2 \cdot \kappa \cdot \tilde{\mathbf{n}}^{(y)}_{p,v}(\mathbf{S}_t) \\ \ldots \\ w_1 \cdot d_{v'}(\mathbf{q}_{p,v}) \\ \ldots \\ -w_2 \cdot \kappa \cdot \mathbf{n}^{(x)}_{v'}(\mathbf{q}_{p,v}) \\ \ldots \\ -w_2 \cdot \kappa \cdot \mathbf{n}^{(y)}_{v'}(\mathbf{q}_{p,v}) \\ \ldots \end{bmatrix} \tag{C.1}$$

C.1.1 Network prediction term

It is easy to see that this term leads to constraints $d_{p,v} = \tilde{d}_{p,v}(\mathbf{S}_t)$ weighted by the parameter w_1. Therefore we can fill the matrix $\mathbf{A}$ with w_1's and the vector $\mathbf{b}$ with $w_1 \cdot \tilde{d}_{p,v}(\mathbf{S}_t)$, as shown in Equation C.1 above.

C.1.2 Orthogonality term

Considering the two orthogonality terms separately, we have two linear constraints weighted by the parameter w_2:

$$\tilde{\mathbf{n}}^{(z)}_{p,v}(\mathbf{S}_t) \cdot \frac{\partial d_{p,v}}{\partial x} = -\kappa \cdot \tilde{\mathbf{n}}^{(x)}_{p,v}(\mathbf{S}_t), \qquad \tilde{\mathbf{n}}^{(z)}_{p,v}(\mathbf{S}_t) \cdot \frac{\partial d_{p,v}}{\partial y} = -\kappa \cdot \tilde{\mathbf{n}}^{(y)}_{p,v}(\mathbf{S}_t) \tag{C.2}$$

Here the superscripts (x), (y) and (z) of the normal $\mathbf{n}_{p,v}$ indicate its x, y, or z component respectively. The first-order derivatives of the depth are approximated with a gradient filter [83], which is convolved with depths in the 3×3 neighborhood per pixel:

$$\frac{\partial \mathbf{D}}{\partial x} \approx \mathbf{L}^{(x)}\mathbf{D} = \mathbf{D} * \frac{1}{12}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -4 & 0 & 4 \\ \hline -1 & 0 & 1 \\ \hline \end{array}, \qquad \frac{\partial \mathbf{D}}{\partial y} \approx \mathbf{L}^{(y)}\mathbf{D} = \mathbf{D} * \frac{1}{12}\begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -4 & -1 \\ \hline \end{array} \tag{C.3}$$

where $\mathbf{L}^{(x)}$ and $\mathbf{L}^{(y)}$ are matrices which implement the above convolution. Therefore for each pixel we can fill the corresponding columns in the sparse matrix $\mathbf{A}$ and entries in $\mathbf{b}$, as shown in the linear system of Equation C.1 above.

C.1.3 View consistency term

The view consistency terms yield similar linear constraints as above. The only difference is that the use the projected depths $d_{v'}(\mathbf{q}_{p,v})$ and transformed normals $\mathbf{n}_{v'}(\mathbf{q}_{p,v})$ (instead of the depths $\tilde{d}_{p,v}(\mathbf{S}_t)$ and normals $\tilde{\mathbf{n}}_{p,v}(\mathbf{S}_t)$ produced from the network).

By combining all linear constraints, weighted by their corresponding weights, we form the overconstrained, sparse linear system of Equation C.1.

BIBLIOGRAPHY

[1] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 86

[2] The models resource, 2017. https://www.models-resource.com/. 87

[3] Asafi, Shmuel, Goren, Avi, and Cohen-Or, Daniel. Weak convex decomposition by lines-of-sight. In *Proc. SGP* (2013). 25, 51

[4] Aucouturier, J., and Pachet, F. Music similarity measures: Whats the use. In *SMIR* (2002). 11

[5] Ballard, D. H. Readings in computer vision: Issues, problems, principles, and paradigms. 1987, ch. Generalizing the Hough Transform to Detect Arbitrary Shapes. 25

[6] Belhumeur, Peter N, Kriegman, David J, and Yuille, Alan L. The bas-relief ambiguity. *International journal of computer vision 35*, 1 (1999), 33–44. 15

[7] Bell, Robert M., and Koren, Yehuda. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl. 9*, 2 (2007). 11

[8] Besl, Paul J., and McKay, Neil D. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (1992). 25

[9] Blumenson, John J. G. *Identifying American Architecture: A Pictorial Guide to Styles and Terms, 1600-1945*. 1995. 4, 8, 22

[10] Bokeloh, Martin, Wand, Michael, and Seidel, Hans-Peter. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graphics 29*, 4 (2010). 13

[11] Bonneel, Nicolas, Sunkavalli, Kalyan, Paris, Sylvain, and Pfister, Hanspeter. Example-based video color grading. *ACM Trans. on Graph. 32*, 4 (2013). 11

[12] Burges, Chris, Shaked, Tal, Renshaw, Erin, Lazier, Ari, Deeds, Matt, Hamilton, Nicole, and Hullender, Greg. Learning to rank using gradient descent. In *Proc. ICML* (2005). 30, 59

[13] Chang, Angel X., Funkhouser, Thomas A., Guibas, Leonidas J., Hanrahan, Pat, Huang, Qi-Xing, Li, Zimo, Savarese, Silvio, Savva, Manolis, Song, Shuran, Su, Hao, Xiao, Jianxiong, Yi, Li, and Yu, Fisher. Shapenet: An information-rich 3d model repository. *CoRR abs/1512.03012* (2015). 87

[14] Chaudhuri, Siddhartha, Kalogerakis, Evangelos, Guibas, Leonidas, and Koltun, Vladlen. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph. 30*, 4 (2011). 13

[15] Chen, Ding-Yun, Tian, Xiao-Pei, Shen, Yu-Te, and Ouhyoung, Ming. On visual similarity based 3D model retrieval. *Computer Graphics Forum 22*, 3 (2003). 38, 102

[16] Chen, Xiaobai, Saparov, Abulhair, Pang, Bill, and Funkhouser, Thomas. Schelling points on 3d surface meshes. *ACM Trans. Graph. 31*, 4 (2012). 28

[17] Choy, Christopher B, Xu, Danfei, Gwak, JunYoung, Chen, Kevin, and Savarese, Silvio. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision* (2016), Springer, pp. 628–644. 16, 90, 91

[18] Cole, Forrester, Golovinskiy, Aleksey, Limpaecher, Alex, Barros, Heather Stoddart, Finkelstein, Adam, Funkhouser, Thomas, and Rusinkiewicz, Szymon. Where do people draw lines? In *ACM Transactions on Graphics (TOG)* (2008), vol. 27, ACM, p. 88. 78

[19] Cole, Forrester, Sanik, Kevin, DeCarlo, Doug, Finkelstein, Adam, Funkhouser, Thomas, Rusinkiewicz, Szymon, and Singh, Manish. How well do line drawings depict shape? In *ACM Transactions on Graphics (ToG)* (2009), vol. 28, ACM, p. 28. 15, 74

[20] Connected Lines. Period furniture style guide, 2014. http://www.connectedlines.com/styleguide/index.htm. 22

[21] DeCarlo, Doug, Finkelstein, Adam, Rusinkiewicz, Szymon, and Santella, Anthony. Suggestive contours for conveying shape. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 848–855. 74, 78

[22] Ding, Chao, and Liu, Ligang. A survey of sketch based modeling systems. *Frontiers of Computer Science* (2016), 1–15. 14

[23] Doersch, Carl, Singh, Saurabh, Gupta, Abhinav, Sivic, Josef, and Efros, Alexei A. What makes Paris look like Paris? *ACM Trans. Graph. 31*, 4 (2012). 11

[24] Dosovitskiy, Alexey, Tobias Springenberg, Jost, and Brox, Thomas. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1538–1546. 16

[25] Eigen, David, and Fergus, Rob. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 2650–2658. 16

[26] Eitz, Mathias, Richter, Ronald, Boubekeur, Tamy, Hildebrand, Kristian, and Alexa, Marc. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4 (2012), 31:1–31:10. 16

[27] Frey, Brendan J., and Dueck, Delbert. Clustering by passing messages between data points. *Science 315* (2007). 66

[28] Fu, Hongbo, Cohen-Or, Daniel, Dror, Gideon, and Sheffer, Alla. Upright orientation of man-made objects. *ACM Trans. Graph. 27*, 3 (2008). 21

[29] Funkhouser, Thomas, Kazhdan, Michael, Shilane, Philip, Min, Patrick, Kiefer, William, Tal, Ayellet, Rusinkiewicz, Szymon, and Dobkin, David. Modeling by example. *ACM Trans. Graphics 23*, 3 (2004). 13

[30] Funkhouser, Thomas, Min, Patrick, Kazhdan, Michael, Chen, Joyce, Halderman, Alex, Dobkin, David, and Jacobs, David. A search engine for 3d models. *ACM Trans. Graph. 22*, 1 (2003). 16

[31] Galliani, S., and Schindler, K. Just look at the image: Viewpoint-specific surface normal prediction for improved multi-view reconstruction. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 5479–5487. 81

[32] Garces, Elena, Agarwala, Aseem, Gutierrez, Diego, and Hertzmann, Aaron. A similarity measure for illustration style. *ACM Trans. Graph. 33*, 4 (2014). 11

[33] Gastal, Eduardo S. L., and Oliveira, Manuel M. Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30*, 4 (2011), 69:1–69:12. 78

[34] Glover, Fred, and Laguna, Manuel. *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, USA, 1997. 46

[35] Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. 2014, pp. 2672–2680. 80, 81

[36] Greig, D. M., Porteous, B. T., and Seheult, A. H. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society 51*, 2 (1989). 27

[37] Guo, Xuekun, Lin, Juncong, Xu, Kai, Chaudhuri, Siddhartha, and Jin, Xiaogang. Customcut: On-demand extraction of customized 3d parts with 2d sketches. *Comput. Graph. Forum 35*, 5 (2016), 89–100. 16

[38] Hertzmann, Aaron, Jacobs, Charles E., Oliver, Nuria, Curless, Brian, and Salesin, David H. Image analogies. In *SIGGRAPH* (2001). 11, 13, 14

[39] Hertzmann, Aaron, Oliver, Nuria, Curless, Brian, and Seitz, Steven M. Curve analogies. In *Proc. Eurographics workshop on Rendering* (2002). 13, 14

[40] Hoiem, Derek, Efros, Alexei A, and Hebert, Martial. Geometric context from a single image. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 1, IEEE, pp. 654–661. 15

[41] Hou, Suyu, and Ramani, Karthik. Sketch-based 3d engineering part class browsing and retrieval. In *Proc. SBIM* (2006). 16

[42] Hu, Ruizhen, van Kaick, Oliver, Wu, Bojian, Huang, Hui, Shamir, Ariel, and Zhang, Hao. Learning how objects function via co-analysis of interactions. *ACM Trans. Graph., to appear* (2016). 12, 100

[43] Hu, Ruizhen, Zhu, Chenyang, van Kaick, Oliver, Liu, Ligang, Shamir, Ariel, and Zhang, Hao. Interaction context (icon): Towards a geometric functionality descriptor. *ACM Trans. Graph. 34*, 4 (2015). 12

[44] Huang, Haibin, Kalogerakis, Evangelos, and Marlin, Benjamin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Computer Graphics Forum 34*, 5 (2015). 13

[45] Huang, Haibin, Kalogerakis, Evangelos, Yumer, Ersin, and Mech, Radomir. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE transactions on visualization and computer graphics* (2017). 16

[46] Huang, Qi-Xing, Su, Hao, and Guibas, Leonidas. Fine-grained semi-supervised labeling of large shape collections. *ACM Trans. Graph. 32*, 6 (2013). 9, 51

[47] Huang, Qixing, Koltun, Vladlen, and Guibas, Leonidas. Joint shape segmentation with linear programming. *ACM Trans. Graph. 30*, 6 (2011). 99

[48] Huang, Qixing, Wang, Hai, and Koltun, Vladlen. Single-view reconstruction via joint analysis of image and shape collections. *ACM Trans. Graph. 34*, 4 (2015). 13

[49] Hurtut, Thomas, Gousseau, Yann, Cheriet, Farida, and Schmitt, Francis. Artistic line-drawings retrieval based on the pictorial content. *J. Comput. Cult. Herit. 4*, 1 (2011). 11

[50] Igarashi, Takeo, Matsuoka, Satoshi, and Tanaka, Hidehiko. Teddy: A sketching interface for 3d freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), pp. 409–416. 15

[51] Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. *CoRR abs/1611.07004* (2016). 16, 80

[52] Jiang, Yun, Koppula, Hema, and Saxena, Ashutosh. Hallucinated humans as the hidden context for labeling 3d scenes. In *Proc. CVPR* (2013). 12

[53] Johnson, Andrew E., and Hebert, Martial. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Trans. Pattern Anal. Mach. Intell. 21*, 5 (1999). 103

[54] Johnson, Justin, Alahi, Alexandre, and Fei-Fei, Li. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision* (2016), Springer, pp. 694–711. 16

[55] Judd, Tilke, Durand, Frédo, and Adelson, Edward. Apparent ridges for line drawing. In *ACM transactions on graphics (TOG)* (2007), vol. 26, ACM, p. 19. 78

[56] Kalogerakis, Evangelos, Chaudhuri, Siddhartha, Koller, Daphne, and Koltun, Vladlen. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. 31*, 4 (2012). 9, 13, 49, 61

[57] Kalogerakis, Evangelos, Hertzmann, Aaron, and Singh, Karan. Learning 3d mesh segmentation and labeling. *ACM Trans. Graph. 29*, 4 (2010). 103

[58] Kalogerakis, Evangelos, Nowrouzezahrai, Derek, Simari, Patricio, McCrae, James, Hertzmann, Aaron, and Singh, Karan. Data-driven curvature for real-time line drawing of dynamic scene. *ACM Trans. Graph. 28*, 1 (2009). 52

[59] Kazhdan, Michael, and Hoppe, Hugues. Screened poisson surface reconstruction. *ACM Trans. Graph. 32*, 3 (2013), 29:1–29:13. 86

[60] Kim, Vladimir G., Chaudhuri, Siddhartha, Guibas, Leonidas, and Funkhouser, Thomas. Shape2pose: Human-centric shape analysis. *ACM Trans. Graph. 33*, 4 (2014). 12, 100

[61] Kim, Vladimir G., Li, Wilmot, Mitra, Niloy J., Chaudhuri, Siddhartha, DiVerdi, Stephen, and Funkhouser, Thomas. Learning part-based templates from large collections of 3d shapes. *ACM Trans. Graph. 32*, 4 (2013). 9, 99

[62] Kingma, Diederik P., and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014). 86

[63] Koenderink, Jan J. What does the occluding contour tell us about solid shape? *Perception 13*, 3 (1984), 321–330. 3

[64] Koenderink, Jan J, Van Doorn, Andrea J, and Kappers, Astrid ML. Surface perception in pictures. *Attention, Perception, & Psychophysics 52*, 5 (1992), 487–496. 15

[65] Kraevoy, Vladislav, Sheffer, Alla, Shamir, Ariel, and Cohen-Or, Daniel. Non-homogeneous resizing of complex models. *ACM Trans. Graphics 27*, 5 (2008). 62

[66] Kreavoy, V., Julius, D., and Sheffer, A. Model composition from interchangeable components. In *Proc. Pacific Graphics* (2007), pp. 129–138. 13

[67] Laga, Hamid, Mortara, Michela, and Spagnuolo, Michela. Geometry and context for semantic correspondences and functionality recognition in man-made 3d shapes. *ACM Trans. Graph. 32*, 5 (2013). 12, 53, 67, 69, 70

[68] Lanckriet, Gert R. G., Cristianini, Nello, Bartlett, Peter, Ghaoui, Laurent El, and Jordan, Michael I. Learning the kernel matrix with semidefinite programming. *J. Machine Learning Research 5* (2004). 56

[69] Larsson, Gustav, Maire, Michael, and Shakhnarovich, Gregory. Learning representations for automatic colorization. In *European Conference on Computer Vision* (2016), Springer, pp. 577–593. 16

[70] Lee, Jeehyung, and Funkhouser, Thomas. Sketch-based search and composition of 3d models. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling* (2008), pp. 97–104. 16

[71] Lee, Yong Jae, Zitnick, C. Lawrence, and Cohen, Michael F. Shadowdraw: Real-time user guidance for freehand drawing. *ACM Trans. Graph. 30*, 4 (2011). 75

[72] Leifman, George. Surface regions of interest for viewpoint selection. In *CVPR* (2012). 28

[73] Lewis, Miles. *Architectura: elements of architectural style*. Barrons Educational Series, 2008. 8, 10

[74] Li, H., Zhang, H., Wang, Y., Cao, J., Shamir, A., and Cohen-Or, D. Curve style analysis in a set of shapes. *Computer Graphics Forum 32*, 6 (2013). 10, 13

[75] Lim, Isaak, Gehre, Anne, and Kobbelt, Leif. Identifying Style of 3D Shapes using Deep Metric Learning. *Computer Graphics Forum* (2016). 10, 99

[76] Lipson, H., and Shpitalni, M. Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design 28* (1996), 651–663. 15

[77] Liu, Han, Vimont, Ulysse, Wand, Michael, Cani, Marie-Paule, Hahmann, Stefanie, Rohmer, Damien, and Mitra, Niloy J. Replaceable substructures for efficient part-based modeling. *Comp. Graph. Forum 34*, 2 (2015). 11

[78] Liu, Tianqiang, Hertzmann, Aaron, Li, Wilmot, and Funkhouser, Thomas. Style compatibility for 3d furniture models. *ACM Trans. Graphics 34*, 4 (2015). 6, 10, 53, 99

[79] Ma, Chongyang, Huang, Haibin, Sheffer, Alla, Kalogerakis, Evangelos, and Wang, Rui. Analogy-driven 3D style transfer. *Computer Graphics Forum 33*, 2 (2014). 13, 14

[80] Malik, Jitendra. Interpreting line drawings of curved objects. *International Journal of Computer Vision 1*, 1 (1987), 73–103. 3, 15

[81] Mitra, Niloy J., Guibas, Leonidas J., and Pauly, Mark. Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3 (2006). 25

[82] Nealen, Andrew, Sorkine, Olga, Alexa, Marc, and Cohen-Or, Daniel. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3 (2005), 1142–1147. 86

[83] Nehab, Diego, Rusinkiewicz, Szymon, Davis, James, and Ramamoorthi, Ravi. Efficiently combining positions and normals for precise 3d geometry. *ACM Trans. Graph. 24*, 3 (2005), 536–543. 84, 109

[84] Nishida, Gen, Garcia-Dorado, Ignacio, Aliaga, Daniel G., Benes, Bedrich, and Bousseau, Adrien. Interactive sketching of urban procedural models. *ACM Trans. Graph. 35*, 4 (2016), 130:1–130:11. 16

[85] Norman, Donald. *The Design of Everyday Things*. Basic Books, 1988. 6, 53

[86] Nutting, Wallace. *Furniture Treasury*. 1928. 4, 8, 10, 22

[87] Ohtake, Yutaka, Belyaev, Alexander, and Seidel, Hans-Peter. Ridge-valley lines on meshes via implicit surface fitting. In *Proc. Siggraph* (2004). 52, 78

[88] Olsen, Luke, Samavati, Faramarz F., Sousa, Mario Costa, and Jorge, Joaquim A. Sketch-based modeling: A survey. *Computers & Graphics 33*, 1 (2009), 85 – 103. 14, 15

[89] Osada, Robert, Funkhouser, Thomas, Chazelle, Bernard, and Dobkin, David. Shape distributions. *ACM Trans. Graph. 21*, 4 (2002). 101

[90] Pan, Hao, Liu, Yang, Sheffer, Alla, Vining, Nicholas, Li, Chang-Jian, and Wang, Wenping. Flow aligned surfacing of curve networks. *ACM Trans. Graph. 34*, 4 (2015). 15

[91] Phong, Bui Tuong. Illumination for computer generated pictures. *Commun. ACM 18*, 6 (1975), 311–317. 78

[92] Pu, Jiantao, Lou, Kuiyang, and Ramani, Karthik. A 2d sketch-based user interface for 3d cad model retrieval. *Computer-Aided Design and Applications 2*, 6 (2005). 16

[93] Rivers, Alec, Durand, Frédo, and Igarashi, Takeo. 3d modeling with silhouettes. *ACM Trans. Graph. 29*, 4 (2010), 109:1–109:8. 76

[94] Ronneberger, O., P.Fischer, and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (2015), vol. 9351, pp. 234–241. 77

[95] Rusinkiewicz, Szymon, and Levoy, Marc. Efficient variants of the icp algorithm. In *3DIM* (2001). 82

[96] Savva, Manolis, Chang, Angel X., Hanrahan, Pat, Fisher, Matthew, and Niessner, Matthias. Scenegrok: Inferring action maps in 3d environments. *ACM Trans. Graph. 33*, 6 (2014). 12

[97] Savva, Manolis, Chang, Angel X., Hanrahan, Pat, Fisher, Matthew, and Niessner, Matthias. PiGraphs: Learning Interaction Snapshots from Observations. *ACM Trans. Graph., to appear* (2016). 12

[98] Saxena, Ashutosh, Sun, Min, and Ng, Andrew Y. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence 31*, 5 (2009), 824–840. 15

[99] Schmidt, Ryan, Khan, Azam, Singh, Karan, and Kurtenbach, Gord. Analytic drawing of 3d scaffolds. *ACM Trans. Graph. 28*, 5 (2009). 15

[100] Schneider, Rosália G., and Tuytelaars, Tinne. Sketch classification and classification-driven analysis using fisher vectors. *ACM Trans. Graph. 33*, 6 (2014). 16

[101] Schölkopf, Bernhard. The kernel trick for distances. In *Proc. NIPS* (2001). 56

[102] Shapira, Lior, Shamir, Ariel, and Cohen-Or, Daniel. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer 24*, 4 (2008). 23, 101

[103] Shtrom, Elizabeth, Leifman, George, and Tal, Ayellet. Saliency detection in large point sets. In *Proc. ICCV* (2013). xvi, 28, 39, 102

[104] Sorkine, Olga, and Alexa, Marc. As-rigid-as-possible surface modeling. In *Proc. SGP* (2007). 64

[105] Su, Hang, Maji, Subhransu, Kalogerakis, Evangelos, and Learned-Miller, Erik. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 945–953. 16

[106] Su, Hao, Qi, Charles R., Li, Yangyan, and Guibas, Leonidas J. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *The IEEE International Conference on Computer Vision (ICCV)* (December 2015). 100

[107] Talton, Jerry, Yang, Lingfeng, Kumar, Ranjitha, Lim, Maxine, Goodman, Noah D., and Měch, Radomír. Learning design patterns with bayesian grammar induction. In *Proc. UIST* (2012), pp. 63–74. 13

[108] Tamuz, Omer, Liu, Ce, Belongie, Serge, Shamir, Ohad, and Kalai, Adam. Adaptively learning the crowd kernel. In *Proc. ICML* (2011). 59

[109] Tatarchenko, Maxim, Dosovitskiy, Alexey, and Brox, Thomas. Single-view to multi-view: Reconstructing unseen views with a convolutional network. *CoRR abs/1511.06702* (2015). 16

[110] Tatarchenko, Maxim, Dosovitskiy, Alexey, and Brox, Thomas. Multi-view 3d models from single images with a convolutional network. In *European Conference on Computer Vision* (2016), Springer, pp. 322–337. 16, 90, 91, 94, 96

[111] Tenenbaum, J.B., Silva, V.De, and Langford, J.C. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000). 41

[112] Tenenbaum, Joshua B., and Freeman, William T. Separating style and content with bilinear models. *Neural Comput. 12*, 6 (2000). 10

[113] Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society 58* (1996). 59

[114] Triggs, Bill, McLauchlan, Philip F., Hartley, Richard I., and Fitzgibbon, Andrew W. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice* (2000), ICCV '99, pp. 298–372. 81

[115] Ulyanov, Dmitry, Lebedev, Vadim, Vedaldi, Andrea, and Lempitsky, Victor. Texture networks: Feed-forward synthesis of textures and stylized images. In *Int. Conf. on Machine Learning (ICML)* (2016). 16

[116] van Kaick, Oliver, Xu, Kai, Zhang, Hao, Wang, Yanzhen, Sun, Shuyang, Shamir, Ariel, and Cohen-Or, Daniel. Co-hierarchical analysis of shape structures. *ACM Trans. on Graphics 32*, 4 (2013). 9

[117] van Kaick, Oliver, Zhang, Hao, Hamarneh, Ghassan, and Cohen-Or, Daniel. A survey on shape correspondence. *Computer Graphics Forum 30*, 6 (2011), 1681–1707. 11

[118] Waltz, David. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision* (1975). 3, 15

[119] Wang, Fang, Kang, Le, and Li, Yi. Sketch-based 3d shape retrieval using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1875–1883. 16

[120] Wang, Xiaolong, Fouhey, David, and Gupta, Abhinav. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 539–547. 16

[121] Willats, John, and Durand, Frdo. Defining pictorial style: lessons from linguistics and computer graphics. *Axiomathes 15* (2005). 11

[122] Wu, Jiajun, Zhang, Chengkai, Xue, Tianfan, Freeman, Bill, and Tenenbaum, Josh. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems* (2016), pp. 82–90. 16

[123] Xie, Xiaohua, Xu, Kai, Mitra, Niloy J., Cohen-Or, Daniel, Gong, Wenyong, Su, Qi, and Chen, Baoquan. Sketch-to-design: Context-based part assembly. *Computer Graphics Forum 32*, 8 (2013), 233–245. 16

[124] Xu, Baoxuan, Chang, William, Sheffer, Alla, Bousseau, Adrien, McCrae, James, and Singh, Karan. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. Graph. 33*, 4 (2014). 15

[125] Xu, Kai, Kim, Vladimir G., Huang, Qixing, and Kalogerakis, Evangelos. Data-driven shape analysis and processing. *Computer Graphics Forum, to appear* (2016). 11

[126] Xu, Kai, Li, Honghua, Zhang, Hao, Cohen-Or, Daniel, Xiong, Yueshan, and Cheng, Zhi-Quan. Style-content separation by anisotropic part scales. *ACM Trans. Graph. 29*, 6 (2010). 9, 13, 14

[127] Xu, Kai, Zhang, Hao, Cohen-Or, Daniel, and Chen, Baoquan. Fit and diverse: Set evolution for inspiring 3d shape galleries. *ACM Trans. Graph. 31*, 4 (2012). 13

[128] Xu, Kun, Chen, Kang, Fu, Hongbo, Sun, Wei-Lun, and Hu, Shi-Min. Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models. *ACM Trans. Graph. 32*, 4 (2013), 123:1–123:15. 16

[129] Yan, Xinchen, Yang, Jimei, Yumer, Ersin, Guo, Yijie, and Lee, Honglak. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems* (2016), pp. 1696–1704. 16

[130] Yang, Jimei, Reed, Scott E, Yang, Ming-Hsuan, and Lee, Honglak. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *Advances in Neural Information Processing Systems* (2015), pp. 1099–1107. 16

[131] Yumer, M.E., and Kara, L.B. Co-abstraction of shape collections. *ACM Trans. Graphics 31*, 6 (2012), 166:1–166:11. 52

[132] Yumer, M.E., and Kara, L.B. Co-constrained handles for deformation in shape collections. *ACM Trans. Graph. 32*, 6 (2014). 9, 13

[133] Yumer, Mehmet Ersin, Chaudhuri, Siddhartha, Hodgins, Jessica K., and Kara, Levent Burak. Semantic shape editing using deformation handles. *ACM Trans. Graph. 34*, 4 (2015). 13

[134] Zeleznik, Robert C., Herndon, Kenneth P., and Hughes, John F. Sketch: An interface for sketching 3d scenes. In *Proc. SIGGRAPH* (1996), pp. 163–170. 15

[135] Zhang, Richard, Isola, Phillip, and Efros, Alexei A. Colorful image colorization. In *European Conference on Computer Vision* (2016), Springer, pp. 649–666. 16

[136] Zheng, Youyi, Cohen-Or, Daniel, and Mitra, Niloy J. Smart variations: Functional substructures for part compatibility. *Comp. Graph. Forum 32*, 2 (2013). 11

[137] Zhou, Kun, Huang, Jin, Snyder, John, Liu, Xinguo, Bao, Hujun, Guo, Baining, and Shum, Heung-Yeung. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3 (2005). 64

[138] Zhou, Tinghui, Tulsiani, Shubham, Sun, Weilun, Malik, Jitendra, and Efros, Alexei A. View synthesis by appearance flow. In *European Conference on Computer Vision* (2016), Springer, pp. 286–301. 16

[139] Zhu, Ciyou, Byrd, Richard H., Lu, Peihuang, and Nocedal, Jorge. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw. 23*, 4 (1997). 60

[140] Zhu, Yuke, Fathi, Alireza, and Fei-Fei, Li. Reasoning about object affordances in a knowledge base representation. In *Proc. ECCV* (2014). 12