Exploring Design Space by Interpolating Between Multiple Sketches

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by

Rahul Arora
Roll No. 10327555

under the guidance of

Dr. Vinay P. Namboodiri, Department of CSE, IIT Kanpur
Dr. Adrien Bousseau, GraphDeco, INRIA Sophia-Antipolis

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
July, 2015

CERTIFICATE

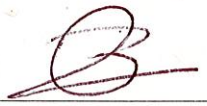
It is certified that the work contained in this thesis entitled "*Exploring Design Space by Interpolating Between Multiple Sketches*", by *Rahul Arora (Roll No. 10327555)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Vinay P. Namboodiri
Department of CSE, IIT Kanpur,
Kanpur, India - 208016

July, 2015

CERTIFICATE

It is certified that the work contained in this thesis entitled "*Exploring Design Space by Interpolating Between Multiple Sketches*", by *Rahul Arora (Roll No. 10327555)*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Dr. Adrien Bousseau
GraphDeco, INRIA Sophia-Antipolis,
Sophia-Antipolis, France - 06902

July, 2015

Abstract

Designers start product design process by drawing various quick and imperfect sketches. In this process, they represent the target concept via multiple shapes and geometric perspectives. We present a sketch-based rendering method and a tool built on top of it to help designers explore the design space induced by sketches of a single concept. Our tool allows a designer to visualize novel sketches by mixing and matching between the geometric and visual properties of multiple sketches.

The method relies on an iterative algorithm to match and warp between sketches using minimal user interaction. We modify known techniques for image matching, warping and morphing to adapt to the unique challenges posed by sketchy inputs, and combine them to allow quick navigation of the design space of sketches. Our tool tries to fill a gap in the initial stage of the product design pipeline, allowing designers and their patrons to make better informed choices before proceeding to the time-consuming and expensive parts of the pipeline involving CAD, 3D design and rendering and/or 3D printing.

Dedicated to my parents for constantly encouraging me; my brother, without whose guidance I wouldn't be here to write a thesis in the first place; and to my friends, who supported me along the journey.

Acknowledgement

I would like to express my sincere gratitude towards my thesis supervisors, Dr. Vinay Namboodiri and Dr. Adrien Bousseau, for their support and encouragement. I am grateful to them for their careful evaluation of my work and giving sound guidance while still giving me freedom to explore and discover my own solutions. Their diverse knowledge, expertise and understanding significantly enhanced my experience as a Masters student. I am also thankful to them, and to Dr. Shashank Mehta, for going the extra mile in administrative efforts to make this collaboration possible.

I thank the Department of Computer Science and Engineering, IIT Kanpur, and Inria Sophia-Antipolis, for providing the necessary infrastructure and a congenial environment for research work. I must also acknowledge the role of Sophie Honnorat and Dr. George Drettakis for making sure I was always welcome at Inria and ensuring that my visits were comfortable, and my academic needs were always taken care of.

A special thanks goes out to Dr. Raghunath Tewari who has been a teacher, research adviser, mentor, and academic guide for me for the past three years. It was under his mentorship that I took my first steps in research, and his guidance prepared me for this thesis.

I cannot appreciate enough the technical help I received from Rodrigo, Ayush, Ashudeep and Kenneth, whom I could always count on whenever I was stuck. A special thanks goes out to Jerome for solving my problems and patiently teaching me time and again whenever I mixed up compilers, target systems, or libraries. I'm obliged to Emmanuel and Raghav for helping me with Cintiq tablets.

A big thank you to all my friends in Kanpur, Delhi, Bangalore and Nice for making this past one year and a quarter fun and happening. Thanks to my friends in the department of CSE, IIT Kanpur, and in GraphDeco, Inria Sophia-Antipolis, for our philosophical debates, exchange of ideas, and venting of frustration over coffee/lunches/dinners during the course of this thesis.

I thank my family for always providing moral support and encouragement for my academic endeavors, and through my entire life.

Lastly, this research would not be possible without the financial assistance of Department of Science and Technology (Govt. of India), and Inria Sophia-Antipolis. I express my gratitude towards both of these institutions.

Rahul Arora

Contents

List of Figures

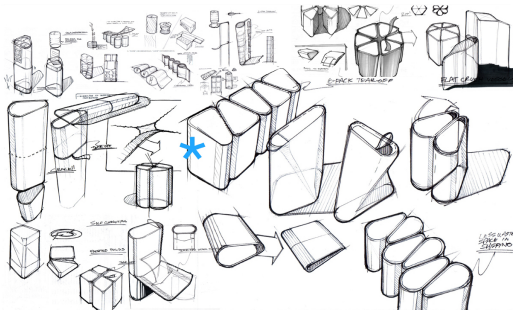
Chapter 1

Introduction

Sketches are swiftly executed freehand drawings used to make a record of an idea that an artist may plan to develop later, or to quickly demonstrate an idea, object or concept. In product design, sketches take a more specific function of representing the general design and functionality of the intended product. In the product design pipeline, various forms of sketches are utilized [Sok]: beginning from those demonstrating rough ideas to geometrically accurate sketches showing exact details of the product. Contemporarily, the latter are generally executed on a digital platform like a tablet computer or a desktop computer while the former may be drawn digitally or using the traditional pen-and-paper approach, depending on the designer's own preferences. The pipeline is a set of various 2D and 3D visualizations produced by the designer to communicate the product's visual and functional details to her patron, before the product is sent for manufacturing.

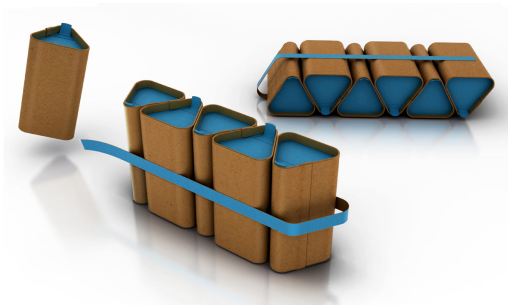
Product design is a dynamic field, and with constant innovation in designing, prototyping and manufacturing techniques, the pipeline keeps on adapting to technological changes. A typical design pipeline (figure 1.1) will, however, involve some of the following generic stages [ES11; Sok]:

- Creating a quick freehand sketch which gives a general idea of the intended product
- Drawing a detailed sketch showing the product from various viewpoints. Depending on the complexity of the product being designed, there may also be

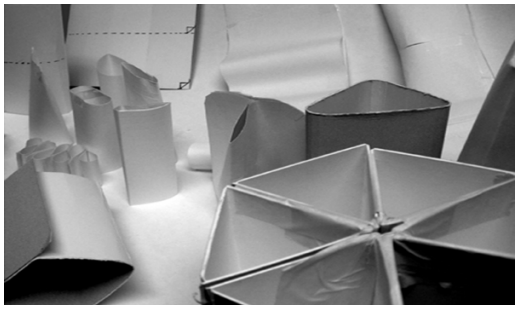


(a) Ideation sketches

(b) Refined sketches



(c) Shaded 3D models



(d) Prototype

Figure 1.1: Product design pipeline. Notice how the number of options is reduced after each stage. © Mike Serafin. `http://www.memikeserafin.com/`.

drawings focusing on the functional details of the product.

- Using a CAD software to produce a to-scale 3D model of the product.
- Prototyping the product using a 3D rendering and shading software or producing a tangible prototype (using traditional techniques such as clay modeling or wire meshing, or with the aid of upcoming technology such as 3D printing).
- Manufacturing the product.

The pipeline is not a linear step-by-step procedure, and involves significant back-and-forth movement depending on changing product needs, developments in understanding of the product, or dialogue between the designers and patrons. However, since steps in the advanced stage of the pipeline entail significantly more designer time and monetary costs, making sound choices at early stages of design can cut down these accruing costs.

This thesis deals with the very initial phase of the product design pipeline. When beginning the process of designing a product, a designer imagines multiple ideas and sketches them roughly. Such sketches are known as *ideation sketches* [BW05], and

provide the starting point for a designer to communicate a concept. The purpose of ideation sketches is to explore ideas quickly, the ideas usually being rough and incomplete and not following conventions which typically guide final drawings.

We present a method to interpolate between multiple sketches of the same functional concept. That is, we interpolate between sketches differing in shape and viewpoint but depicting objects belonging to the same functional class: a typical use case being a product designer exploring initial ideas. While our method is generic, and should work for different types of sketches, we focus on ideation sketches since interpolating and morphing ideation sketches is a problem that hasn't been studied before. Earlier methods [Sha+12; Xu+14; Nea+07; BBS08] have dealt with slightly advanced stages of the design pipeline: facilitating the movement from initial sketches to final drawings or CAD models. While these tools are useful in their own right, it still takes significant effort to generate proper inputs for these tools, and such tools don't work out of the box for inaccurate sketches which are common during the ideation phase. For instance, along with additional user interaction, such methods often require the designer to simplify and *vectorize*[1] their sketches to make the inputs computationally simpler and more approachable. Therefore, it makes sense to explore methods which allow designers to make more informed choices in the ideation phase itself, before taking the effort to vectorize the sketches. For completeness, we note that there exist several systems for automatic vectorization of sketches, but even state of the art systems such as [OK11; BC13] have significant constraints on the type of sketches or the sketching methods they can work with. For instance, the method presented in [OK11] makes use of temporal properties of the sketching process and records strokes while sketching digitally, thereby foregoing the traditional sketches preferred by many designers [BD03].

Our interpolation method allows exploration of design space of sketches by generating in-between shapes and viewpoints for the objects being created. See Figure 1.2 for a concise description of our pipeline, and an example of the final in-between

[1]Vectorization is the process of converting the raster graphics expressed as pixel intensities to vector graphics which are expressed using geometrical primitives such as lines, circles, et cetera.

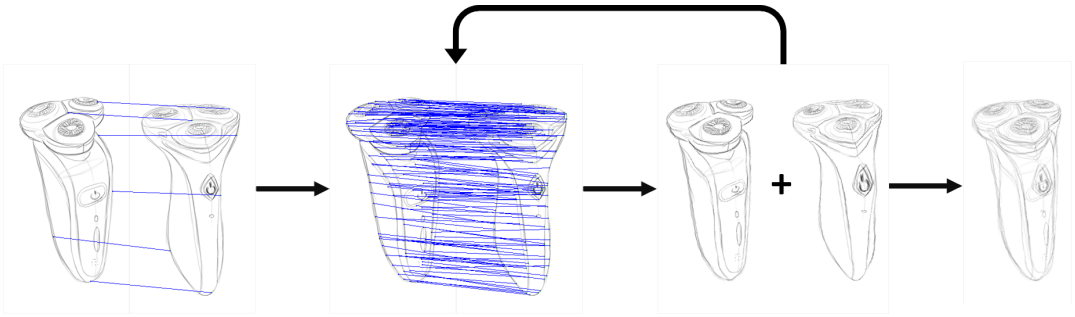


Figure 1.2: The proposed sketch morphing pipeline. (Left) Input: sketches along with very sparse (5-6) point correspondences. (Center-left) Matching: expand the set of correspondences. (Center-right) Warping: align the two sketches to generate dense (per-pixel) correspondences. (Right) Rendering: generate the final interpolated sketch using the two warps and blending. In our pipeline, matching and warping are performed as alternating steps in an iterative algorithm: point correspondences improve alignment and improved alignment gives better matches.

sketches we generate. This allows a designer to quickly explore more ideas at an early stage of design, thus enabling her to make more informed choices with the increased ideas. The tool we build on top of the interpolation algorithm also allows the designer to explore the design space as an animated visualization of a sequence of in-between sketches in the design space. The tool also allows designers to mix and match among the projected shapes and styles of several sketches.

Our interpolation pipeline is based on the standard image morphing pipeline (see Figure 1.2): estimate a sparse matching between images, warp the two images onto each other, and blend the warped images together. However, there are numerous options available for executing each step, and we describe a number of them in detail, along with rationales for our choices in chapter 3. We also motivate our choices based on experiments discussed in chapter 4. Moreover, multiple non-trivial adaptations are required to make the pipeline work for sketches, as most of the ingredients available are suited for natural images and do not work well out-of-the-box for sketches. This is because a sketchy input style presents unique challenges of its own. Unlike natural images, sketch data is very sparse: everything other than the contours is just white space. Freehand sketches also tend to be geometrically inaccurate, are often incomplete, and do not represent exact projections of the shape of the object being described. Another challenge specifically due to our use case is

that we aim at interpolating between different shapes as well, which means that we need to estimate a highly non-rigid transformation between the two sketches, which is not handled well by many popular methods, since they try approximating affine transformations. Finally, a large number of modern registration frameworks such as [TN10; TZY08; Shr+11] which work for natural images or simple shapes are data-driven but such approaches are difficult to implement successfully for ideation sketches since data is scarce. This motivates us to build an algorithmic technique.

The basic pipeline interpolates between two sketches, and interpolating between multiple sketches is a natural generalization. First, we use a small set of user correspondences to build a set of uniformly distributed sparse correspondences between the two sketches. Then, we overlay a triangle mesh each on both of the sketches. We warp the mesh in a *shape preserving* manner, using the information from the sparse matching. In our warp, the mesh functions as a partition of the sketch into a piecewise linear domain, that is, we just solve for the positions of the mesh vertices, and positions of all pixels inside the triangles are obtained via linear interpolation. This gives us a dense matching between the two sketches. We do these two steps iteratively to converge to a good solution (normally, we use a fixed number of iterations). Lastly, we use OpenGL to generate the individually warped sketches, or to blend both the sketches together to produce an in-between warp. Our tool provides the designer with a choice to explore either of these options. This pipeline lists the set of components which gives the best results in our experiments. However, we also explore various other options for the sparse matching and warping steps, which are described in detail in chapter 3. Specifically, it is possible to forgo user correspondences altogether and build a completely automatic solution. However, we normally keep the user correspondences since it gives better results and allows some level of user control on the process.

Thesis organization. The thesis is organized as follows. chapter 2 briefly discusses the literature closely related to the thesis. In chapter 3, we describe the pipeline in detail, along with various components we tried for each step. chapter 4

then lists the experiments we carried out and the results we obtained. This chapter also describes the tool we built for exploration of design space. We conclude with a summary of the thesis and a discussion of possible future work in chapter 5.

Chapter 2

Related Work

While there isn't any previous work that we are aware of which deals with the problem of morphing between freehand sketches, a lot of interesting work has been done in various related fields which we build upon. We broadly categorize the related work into three categories: 1. Sketch-based modeling and interpolation, 2. Image-based rendering and morphing, and 3. Image matching. We will briefly discuss each of them.

2.1 Sketch-based modeling and interpolation

Sketch based modeling has been extensively studied in literature. A number of methods reviewed by [Ols+09] require a user to draw a clean sketch from multiple viewpoints, and the system fits a 3D model to them. Lately, some methods have focused on building full or partial 3D models beginning with a single sketch as the starting input. CrossShade [Sha+12] infers surface normal fields using cross-sections marked by designers to show the principal surface directions, and uses these fields to facilitate 3D rendering. However, the method requires the user to provide vectorized sketches and clearly marked cross-sectional curves, which itself is a time consuming process. More recently, [Xu+14] proposed a method to build 3D models from vectorized drawings using certain 3D regularity cues inferred from sketches. Some methods such as [BBS08] allow users to directly sketch 3D models

using a custom interface. We see our work as fitting in the design pipeline before methods such as these so that the designers have a better idea of the shape they're going to design even before committing to computing or designing a 3D model. Our method also differs from the above in the sense that they use a single sketch/shape as the definitive input while our method is meant to explore the design space for the selection of one definitive sketch to take forward along the design pipeline.

Perhaps the closest relative to the problem being solved in this thesis is [RID10], which uses vectorized cartoon drawings drawn from various different angles to infer some 3D information, and render from novel viewpoints. The usage of 2D interpolations to approximate 3D rotations is similar to what we try to achieve. The major differences are that the method uses vectorized drawings, and the correspondences are provided implicitly by the user by drawing labeled strokes in each view. Finding correspondences is a difficult problem itself, which we try to solve. Also, their method is quite restrictive in terms of the key viewpoints required and the complexity of supported drawings.

[Sha+13] also try to interpolate between different concept sketches. However, the method focuses on transitioning between extreme positions of a moving part in an industrial object. Moreover, the moving part must be approximable using a cylindrical or cuboidal geometry (also known as the *proxy* geometry) and goes through a rigid motion . On the contrary, our method can be used for general classes of objects and motions between sketches. The proposed method gives a more general way to represent non-rigid interpolations.

2.2 Image-based rendering and morphing

Another closely related area which has undergone comprehensive scientific study is image-based rendering. Sitting at the intersection of the fields of computer graphics and vision, image-based rendering (IBR) algorithms attempt to render novel views by warping and/or morphing input images, unlike the traditional 3D computer graphics pipeline which requires complete geometric information. Image-based ren-

dering methods may ([CW93; SD96]) or may not ([LH96]) use *implicit* geometric information specified as correspondences between input images. Sometimes sparse or approximate 3D geometric information about the scene may also be specified as an input, for example in [MB95]. More discussion on IBR goes beyond the scope of this thesis. However, the interested reader may go through [SK00] for a brief review of IBR techniques. The reader is also advised that the review, however, excludes recent developments in the field.

Our method is most comparable to approaches which use implicit geometric information, that is, 2D correspondences between images. [SD96] introduced view morphing, which uses manually provided sparse point correspondences to estimate camera positions and mimic 3D rotations between the two viewpoints. However, as discussed in chapter 1, 2D correspondences in sketches do not directly encode correspondences in 3D space, and can therefore be more challenging to use. Moreover, correspondences were provided manually in the original paper, while our method generates them based on very few user indications. Recent methods have bettered on the classical view morphing technique introduced in [SD96] using various methodologies such as estimation of optical flow [Mah+09], or computation of half-way image for better alignment [Lia+14]. The latter works particularly well for images with varying shapes, colors as well as viewpoints and requires only very sparse correspondences from the user. However, the technique is only suited to a dense representation provided by natural images, and doesn't naturally generalize to sparse shape representations inherent in sketches.

2.3 Image matching

Image matching, often known as *image registration* in literature, aims at automatically or manually finding correspondences between images in order to align them. Most recent automatic image registration methods are *feature-based*, that is, they represent an image using features computed uniformly or on detected keypoints, and then match the feature representation of one image with another. SIFT [Low04] is

one of the most common features used in modern registration pipelines. Image registration methods often also include a warping method as the final step or as part of an iterative process to align the input images. A thorough compilation on recent feature-based registration techniques can be found in [Bar07].

A well-studied approach to feature-based correspondence is translating the problem into what is known as *graph matching*. This approach involves representing an image as a graph with its vertices representing low level features (say patch features such as SIFT and patch intensity) and edges representing the relations between the features. A graph matching algorithm then tries to lay the graph corresponding to one image over the other while minimizing the distortions of the two. Since general graph matching is NP-hard [GR96], various optimization methods, e.g., [TKR13; Lee12; SCL12] have been proposed to solve its various relaxations.

Our sketch interpolation pipeline is closely linked to image registration. Beginning with very sparse user correspondence between two sketches, we generate denser correspondences by iteratively matching shape features and warping one sketch towards the other. We also examine a state of the art graph matching algorithm in chapter 3, discussing its adaptation to a sketchy input style, and its performance vis-à-vis our approach.

Chapter 3

Interpolation between Sketches

In this chapter, we describe in detail our algorithm to interpolate between sketches. The goal is to generate novel shapes and viewpoints to visualize the concept the designer wants to explore. Our focus, for this chapter, largely remains on interpolating between two sketches, since the interpolation algorithm for multiple sketches is a straightforward generalization. The problem of interpolation between two sketches is similar to that of keyframing [D.85] in animation in which a designer specifies key events in the motion path, and an algorithm attempts to mimic the full motion path by interpolating between subsequent frames. The keyframes, however, tend to be much more similar than the pairs of sketches we target, and the transformations can normally be treated as rigid.

Here, we describe our method to perform the non-rigid transformation between two sketches. We also go through various different approaches we explored, before arriving at the current solution.

3.1 Overview

Consider two sketches $\mathcal{S}_0$ and $\mathcal{S}_1$, both represented in the form of grayscale images of size (h, w). That is, $\mathcal{S}_i \in [0,1]^h \times [0,1]^w,\ i \in \{0,1\}$. On a high level, our task is to generate all plausible sketches *between* them. That is, if we could map the two

sketches $\mathcal{S}_0$ and $\mathcal{S}_1$ to some 1D space, we need to produce

$$\mathcal{S}_\alpha = \alpha \times \mathcal{S}_0 + (1 - \alpha) \times \mathcal{S}_1 \tag{3.1}$$

Since there is no single mathematically correct way to transform images or sketches to such a one-dimensional space, there can be various methods to perform sketch interpolation. Many image morphing algorithms can be broken down into a three-step pipeline: 1. Matching, 2. Warping, and 3. Rendering.

We model our algorithm on this traditional pipeline, while adapting each step for our purpose. Matching refers to the generation of a sparse set of correspondences between the two given sketches. Since sketch matching is just a special case of image matching, our matching pipeline is similar to a typical registration algorithm for natural images, with certain adaptations to suit our inputs. Warping is the process of using these sparse correspondences, as well as inherent information present in the sketches, to warp both sketch onto each other. Rendering, the final step of the pipeline, is used to morph the warped versions of sketches together.

In the remaining parts of this chapter, we describe various methods we tried for the three steps, along with the final solution we arrived at: An iterative method involving alternating *shape context* matching and *shape preserving* warping steps augmenting each other, followed by an alpha blending based morphing step to render the final sketch(es). The organization is as follows. We first describe the matching strategies we tried, including shape contexts. Then we describe various approaches to warping, with the shape preserving warp discussed in detail. We then outline our iterative method which combines the two. Finally, we describe how we obtain the warped sketches, and blend two or more of them together to generate the final render.

3.2 Matching

The aim of the matching step in our algorithm is to generate a set of point correspondences between the two sketches. This set of correspondences is then used as an input for the next step in the algorithm, warping. The warping step is algorithmically independent of how the matches are generated; therefore, we describe various methods to get the sparse correspondence set, $M(\mathcal{S}_0, \mathcal{S}_1)$.

3.2.1 Manual matching

The simplest way to obtain matches is just to ask the user for them. However, providing correspondences is a tedious task, and any modern registration algorithm cannot rely on the user to provide too many correspondences. Moreover, the application we target (see Chapter 4) can work with a large number of concept sketches, and it can be a daunting task for a user to provide all the required correspondences. Therefore, while our method does rely on user correspondences, we limit the requirement to a very small number. Typically, our pipeline needs only 4-6 point correspondences per sketch pair from the user. The user can also choose to keep the same correspondences for multiple pairs, further reducing the effort required.

3.2.2 Shape context matching

Shape context [BMP02] is a descriptor which has been widely used to describe 2-D shapes. Shape contexts are computed by treating an object as a set of two-dimensional points. A drawing or sketch of an object, $\mathcal{S}$ can be represented by a set of discrete points by sampling along its contours. Practically, this is done by taking a sample of its edge pixels, as detected via a suitable edge detector $\texttt{Edge}(\mathcal{S})$, to get a set $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ of n points, where each $\mathbf{p}_i = (x_i, y_i)$ is a pixel of the sketch $\mathcal{S}$.

Consider a point $\mathbf{p}_i,\ 1 \leq i \leq n$. To compute the shape context descriptor of $\mathbf{p}_i$, imagine a set of $n-1$ vectors from $\mathbf{p}_i$ to all other points in the set $\mathcal{P}$. Shape context

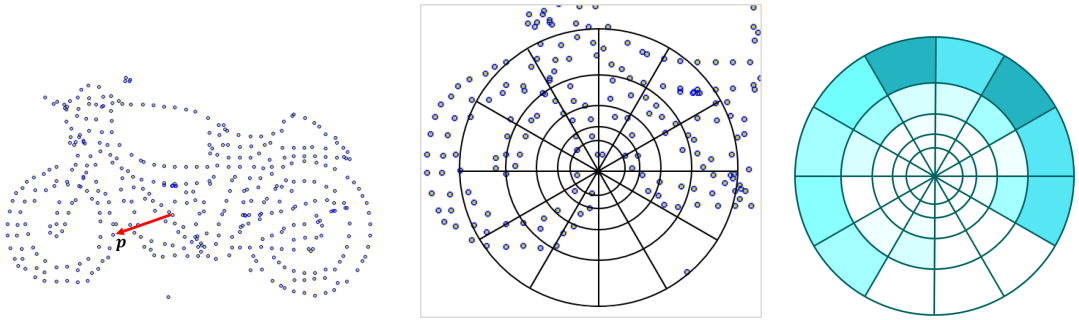


Figure 3.1: Shape context feature descriptor. a) Point set sampled from a sketch. b) Laying the shape context log-polar grid on the point $\mathbf{p}$. c) Shape context histogram: color darkness represents the value stored in the bin.

includes information about these $n-1$ vectors, expressing the configuration of the entire shape relative to $\mathbf{p}_i$. This makes shape context a rich and discriminative shape descriptor, unlike more local visual descriptors such as contour location or pixel intensity. However, to make the description compact and robust to small shape differences, the vectors are distributed into histograms by dividing the $\mathbb{R}^2$ space into K bins. The sequence of histograms on these bins is known as the *shape context* of $\mathbf{p}_i$.

$$h_i(k) = \#\{\mathbf{q} \neq \mathbf{p}_i | (\mathbf{q} - \mathbf{p}_i) \in \text{bin}_k\} \tag{3.2}$$

We use the histogram bin distribution as described in the original paper. The bins are uniform in log-polar space [AD96] with respect to $\mathbf{p}_i$. This means that the bins are demarcated by concentric circles centered at $\mathbf{p}_i$ whose logarithm of radii forms an arithmetic progression, and lines passing through $\mathbf{p}_i$, whose angles from an arbitrary reference direction form an arithmetic progression (see Figure 3.1).

Shape contexts are intrinsically invariant to translations. They are easily made scale invariant by normalizing all radial distances by the mean distance between the n^2 point pairs in the shape. Moreover, the coarse histogram representation makes shape contexts robust against small geometric perturbations, occlusion, and outliers.

Consider a point $\mathbf{p}_i$ sampled from $\mathcal{S}_0$ and a point $\mathbf{q}_j$ sampled from $\mathcal{S}_1$. Since, shape contexts are histogram based distributions, the cost of matching $\mathbf{p}_i$ to $\mathbf{q}_j$ is

naturally defined using the χ^2 test statistic:

$$C(\mathbf{p}_i, \mathbf{q}_j) = \frac{1}{2} \sum_{k=1}^{K} \frac{[h_i^0(k) - h_j^1(k)]^2}{h_i^0(k) + h_j^1(k)} \tag{3.3}$$

where $h_i^0(k)$ and $h_j^1(k)$ are the K-bin normalized histograms at $\mathbf{p}_i$ and $\mathbf{q}_j$, respectively. [BMP02] also suggests using an optional term based on local appearance similarity in terms of intensities at the patches around $\mathbf{p}_i$ and $\mathbf{q}_j$. However, the paper uses this term only when working with natural images. Since we deal with sketches varying in shape, and do not wish to rely on drawing style for matching as well, we do not use such a term.

Once we compute the shape context for all sampled points on both the images, we can use a bipartite matching algorithm such as the Hungarian algorithm [Mun57] to compute the best matching between the two point sets. This method works by solving for the permutation σ of size n such that the shape distance

$$H(\sigma) = \sum_{i=1}^{n} C(\mathbf{p}_i, \mathbf{q}_{\sigma(i)}) \tag{3.4}$$

is minimized. The algorithm works in $O(n^3)$ time.

However, the Hungarian algorithm only makes use of the correspondences, thus matching each point to the most similar one in the second image, independent of its neighbors. Such a technique can give a geometrically discontinuous matching (figure 3.2c), which is undesirable for warping applications. Introducing *regularization* over the image space enables generation of smooth warp functions, as described next.

3.2.3 Regularization

In various computer vision and machine learning applications, additional constraints are generally introduced when solving a problem to prevent overfitting. Such terms are known as regularization terms. Intelligent regularization terms help produce smoother results by reducing the weights of bad matches. In computer vision applications, therefore, it is common to call the regularization terms as *smoothness*

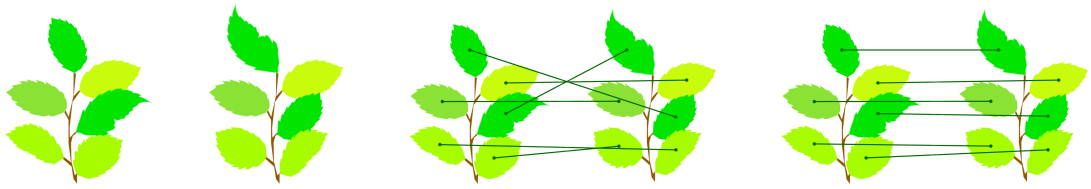


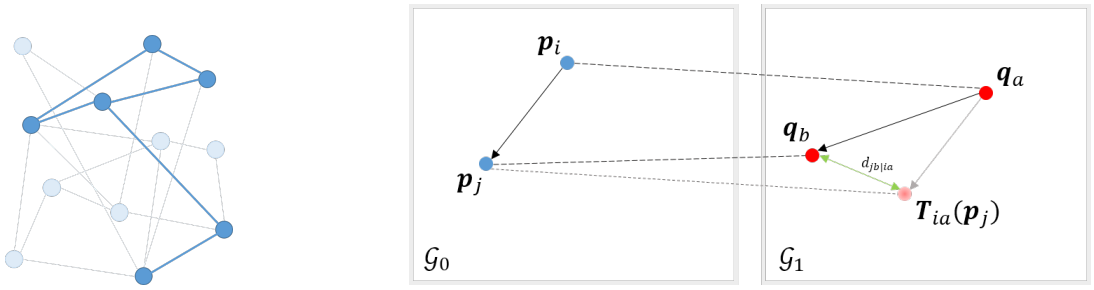
Figure 3.2: Consider matching using a scale and flip invariant feature computed on continuous regions of a single color. a) and b) Images to be matched. c) Matching in the absence of regularization: each leaf matches its most similar looking counterpart. d) Matching utilizing geometric regularization favoring matched leaves to have the same relative positions. In an application like warping, the second matching set is better, since it can produce a smooth warp.

terms, and the correspondences as the *data* terms.

See Figure 3.2 for an example of how regularization helps produce better matches by preventing local discontinuities. Smoothness terms can also be a way to densify the sparse matches in an intelligent fashion, and section 3.3 on warping explains the various smoothness terms which can be used to create a dense match from the sparse matches obtained via shape context matching, graph matching, or manual matching methods. We now describe graph matching.

3.2.4 Graph matching

For sketch matching using shape contexts, an image is represented using a set of two-dimensional points. Another representation widely used in literature [TKR13; Lee12; SCL12] is a graph. Let us try to represent a sketch S by a graph $\mathcal{G}(V, E)$. We can use an edge detection based strategy as described in section 3.2.2 to find a set of n points to represent the image, say $V \equiv \mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$. However, we also include relations between some pairs of vertices in the form of edges of the graph, E. Elements of both V and E may also have some attributes associated with them. Attributes associated with the the vertices of $\mathcal{G}$ are generally image features such as SIFT [Low04] or MSER [DB06], or shape features such as shape contexts. Edge attributes, and the decision to choose edges to be considered out of the possible n^2 vertex pairs depend on the exact graph matching algorithm being used.



(a) (b)

Figure 3.3: Progressive Graph Matching. a) Active graph (solid edges and vertices) out of the full graph (faded edges and vertices). b) Computation of distance $d_{jb|ia}$ to find geometric compatibility of the matches $(\mathbf{p}_i, \mathbf{q}_a)$ and $(\mathbf{p}_j, \mathbf{q}_b)$. $\mathbf{T}_{ia}$ is the translation from $\mathbf{p}_i$ to $\mathbf{q}_a$.

Here, we describe "Progressive Graph Matching", a state of the art graph matching algorithm. In [Lee12], the authors present a progressive framework for solving general graph matching problems, along with a strategy to construct the graph from natural images for registration applications. We adapt the strategy for registering sketches.

The progressive framework consists of two alternating steps executed iteratively: *graph progression* and *graph matching*. Evidently, *graph matching* refers to using any available graph matching algorithm for matching the *current* graphs, but the authors report their best results using the Reweighted Random Walks for graph Matching (RRWM) algorithm described in [CLL10], and we stick to the same algorithm for all our experiments as well. The *graph progression* step updates the current graphs using a Bayesian formulation. The formulation essentially uses a probabilistic voting to choose the sets of *active* vertices and edges for both graphs, with *active* meaning that only these vertices/edges are considered for matching during the next graph matching step (see Figure 3.3a for example). We just give a basic description of the algorithm here. For exact optimization methods, the reader can refer to the original papers.

Let us now come back to the problem of matching sketches $\mathcal{S}_0$ to $\mathcal{S}_1$. Let us call the full graph for sketch $\mathcal{S}_i$ $(i \in \{0, 1\})$ as $\mathcal{G}_i(V_i, E_i)$, while the active graph for the j^{th} iteration be $\mathcal{G}_i^j(V_i^j, E_i^j)$. Note that all these graphs are undirected. Firstly, the

full graph $\mathcal{G}_i$ is created by dividing the sketch $\mathcal{S}_i$ into a grid, and then searching from each cell a junction of two or more edges, and if one is not found, then searching for any point on an edge (see red points in figure 3.5c). The former is done since junctions mark regions of rapid shape or orientation change and thus form important visual cues in sketch understanding, while the latter is useful since edge detectors find contours, which are the basic building blocks of sketches. The edge sets are chosen to form complete graphs, that is, $E_i = V_i \times V_i$.

The vertex set of $\mathcal{G}_0^0$, V_0^0, is kept as V_0, while V_1^0 is selected as:

$$V_1^0 = \cup_{\mathbf{p} \in V_1} \{(\mathbf{p}, \mathbf{q}_j) | \mathbf{q}_j \in \texttt{kSC}(\mathbf{p}, V_j, n_0)\} \tag{3.5}$$

where $\texttt{kSC}(\mathbf{p}, V, n)$ gives the n nearest neighbors of $\mathbf{p}$ in the set V in terms of shape context distance, and n is a parameter. The edge set is computed as:

$$E_i^0 = \cup_{\mathbf{p} \in V_i^0} \{(\mathbf{p}, \mathbf{p}') | \mathbf{p}' \in \texttt{kNN}(\mathbf{p}, k_1)\} \tag{3.6}$$

where $\texttt{kNN}(\mathbf{p}, k)$ gives the set of k-nearest neighbors of p in the image space. For any t, the edge set is constructed in a similar fashion. In each iteration, the graph progression step constructs $\mathcal{G}_i^t$ from $\mathcal{G}_i^{t+1}$ using probabilistic voting from each set of matches in $\mathcal{G}_i^t$. More precisely, the probability of retaining/including a match $\mathbf{m} = (\mathbf{p}_i, \mathbf{q}_a)$ is dependent on two terms: the data term is computed independently for all matches, and is therefore known as the *unary affinity*, while the smoothness (regularization) term is computed by taking pairs of neighboring matches, and is called the *pairwise affinity*. The former is given by the shape context distance $C(\mathbf{p}_i, \mathbf{q}_a)$, and represents the strength of the match. The latter describes how well $\mathbf{m}$ fits with the other matches which are connected to it via an edge in E_0^t or E_1^t. The fitness between two matches $\mathbf{m}_1 = (\mathbf{p}_i, \mathbf{q}_a)$ and $\mathbf{m}_2 = (\mathbf{p}_j, \mathbf{q}_b)$ is computed as a monotonically decreasing function (for example, $f(x) = \alpha - x$, or $f(x) = e^{-x}$) of the geometric difference $d_{jb|ia}$ of the vectors from $\mathbf{p}_i$ to $\mathbf{p}_j$ and from $\mathbf{q}_a$ to $\mathbf{q}_b$, that is, $d_{jb|ia} = \|(\mathbf{p}_j - \mathbf{p}_i) - (\mathbf{q}_b - \mathbf{q}_a)\|$ (refer figure

3.3b). Note that, $d_{ia|jb} = d_{jb|ia}$. To make the problem tractable, and to consider only local effects, the geometric fitness is computed only for matches such that $\{\mathbf{p}_i, \mathbf{p}_j\} \in E_0^t \wedge (\mathbf{q}_b \in \texttt{kNN}(\mathbf{q}_a, k_2) \vee \mathbf{q}_a \in \texttt{kNN}(\mathbf{q}_b, k_2))$, where $k_2 (< k_1)$ is a parameter. Since the formulation computes probabilities, the geometric difference is normalized over all pairs of matches considered. The pairwise affinity helps solve the problem of overfitting by suppressing matches which are not compatible with their neighbors. This results in a smoother alignment of the two images and reduces local discontinuities.

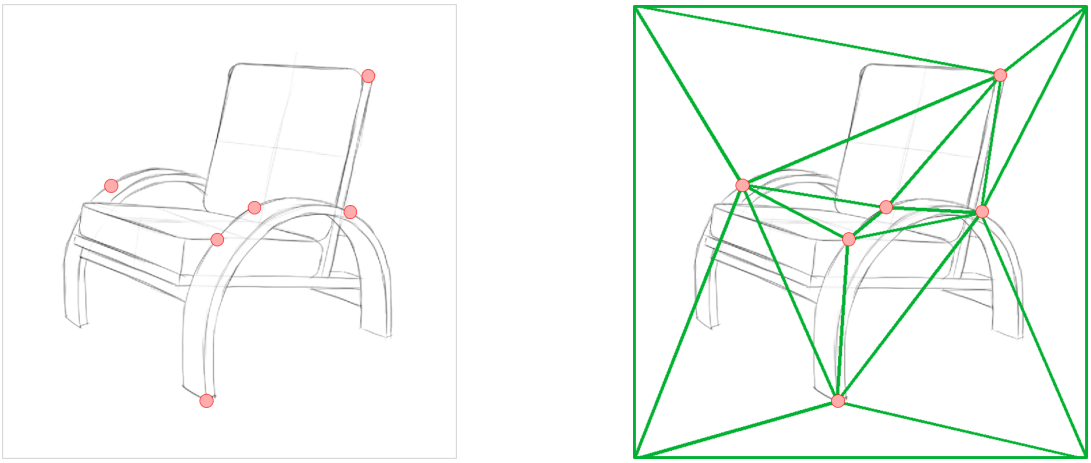
The graph progression then computes the optimal values of V_i^{t+1} $(i \in \{0, 1\})$ so that the sum of the two affinities for all valid matches and match pairs is maximized. This value is called as the *score* of the graph matching step. The iterations are continued till this score increases.

3.3 Warping

Image warping is the process of moving the pixels of an image. In our context, we use image warping techniques to transform a sketch (the source image) so that it aligns with another one (the destination image). Since, sketches are just a special class of images, in order to fully specify such a warp, we need to specify where each pixel of the source image should move to. This is equivalent to specifying the motion vector for each pixel. We use the term *motion field* to refer to the motion vectors of all the pixels being moved. Mathematically, the motion field $\mathcal{F}$ can be expressed as $\mathcal{F}_{\mathbf{p}} = T(\mathbf{p}),\ \forall \mathbf{p} \in$ pixels in the image, where $T(\mathbf{p}) \in \mathbb{R}^2$ gives the motion vector at a pixel $\mathbf{p}$. Now, we discuss various ways of solving for this motion field, given sparse point correspondences from one sketch to another.

3.3.1 Linear interpolation using triangulation

Consider $\mathcal{S}_0$ and $\mathcal{S}_1$, the source and destination sketches. Let the set of correspondences between them be $\mathbf{M} = \{\mathbf{m}_i = (\mathbf{p}_i, \mathbf{q}_i) \mid 1 \leq i \leq n\}$, where $\mathbf{p}_i$'s are pixels on



(a) **(b)**

Figure 3.4: Visualizing how triangulation-based linear interpolation works. a) An input sketch along with points which have their correspondences defined. b) A Delaunay triangulation having vertices as points which have their correspondences defined, along with the corners. For a general point, its barycentric coordinates are used for linear interpolation.

$\mathcal{S}_0$ and $\mathbf{q}_i$'s are pixels on $\mathcal{S}_1$. Now, to compute the motion field for the $(\mathcal{S}_0, \mathcal{S}_1)$ pair, we partition the sketch into a set of triangles generated using Delaunay triangulation on the set $\{\mathbf{p}_i \mid 1 \leq i \leq n\} \cup \{\text{corners of } \mathcal{S}_0\}$ (see figure 3.4). Let $\mathcal{F}_{xy}$ represent the motion field at a pixel (x, y) in a triangle with vertices (x_1, y_1), (x_2, y_2) and (x_3, y_3), with barycentric coordinates (α, β, γ). Assume that the corners of $\mathcal{S}_0$ are matched to the respective corners of $\mathcal{S}_1$. Now, the motion field at (x, y) can be computed as the weighted average of the three vertices as:

$$\mathcal{F}_{xy} = \alpha \mathcal{F}_{x_1 y_1} + \beta \mathcal{F}_{x_2 y_2} + \gamma \mathcal{F}_{x_3 y_3} \tag{3.7}$$

However, linear interpolation using Delaunay triangulation is too simple to produce good results for most registration applications. Moreover, it can only be used to densify correspondences, and cannot correct local mismatches by smoothing the motion field, as all matches must be treated as hard constraints. In Chapter 4, we show that linear interpolation is insufficient for sketch warping.

3.3.2 Thin plate splines

Another widely used model for interpolation is the Thin Plate Spline (TPS) model [Duc77]. In addition to interpolation, TPS also allows for local smoothing. The model is physically analogical to the bending of a thin metal sheet. Just as a rigid metal sheet resists bending, TPS also includes energy terms to impose a penalty involving the smoothness of the fit surface. To understand the thin plate spline model, consider the same set of correspondences and their associated motion vectors. Also, let $\mathcal{F}^x_{\mathbf{p}}$ give the x-component of the motion field at $\mathbf{p}$, and $\mathcal{F}^y_{\mathbf{p}}$ the y-component. Imagine the field $\mathcal{F}^x$ as a displacement in the z-direction, orthogonal to the plane of the sketch $\mathcal{S}_0$. Now, we need to fit a surface which best interpolates the displacement field $z = \mathcal{F}^x_{\mathbf{p}},\ \forall(\mathbf{p},\mathbf{q}) \in \mathbf{M}$. Similarly, we proceed for $\mathcal{F}^y$. We do not describe the exact equations governing the bending energy in the TPS model. Suffice to say that the TPS model works better than triangulation based linear interpolation since it provides for smoothing as well. However, the shape preserving model described in the next section works better than TPS, and we use the same for our experiments. See the results in chapter 4 for a comparison.

3.3.3 Shape preserving warp

The idea of a shape preserving warp is to maintain the local shape of the image being warped as much as possible, that is, minimizing the distortion caused by the warp. Various strategies for shape preserving warps have been described in [Zha+09; Liu+09; CSD11; CSC14]. While none of these strategies is directly applicable for sketch warping, the overall strategy we employ is adapted from [CSD11], and the optimization problem we solve is very similar to theirs.

We set up an energy minimization problem by combining the positional constraints due to the correspondences $\mathbf{M}$, and the shape preserving behavior of the warp. We then solve this optimization problem using variational optimization.

Setup. Before solving for the shape-preserving warp, we need to overlay a triangle mesh on the given sketch $\mathcal{S}_0$. The triangulation we require is a con-

strained Delaunay triangulation [Che87] in which the contours of the sketches are necessarily included as segments in the triangulation. The contours are first detected using a suitable edge detection method. Then, regular samples are taken along the edges to build the set of vertices $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ for the triangulation. Then, the edges detected by the edge detection method are discretized as segments $\mathcal{E} = \{\mathbf{e} = \overline{\mathbf{v}_i\mathbf{v}_j} \mid \mathbf{e} \text{ lies on an edge}\}$, where $\overline{\mathbf{v}_i\mathbf{v}_j}$ represents a line segment between $\mathbf{v}_i$ and $\mathbf{v}_j$. Since large edges can essentially be covered using small consecutive edges, we perform this discretization step by searching in a small neighborhood around each vertex only. We also add additional vertices and segments to $\mathcal{V}$ and $\mathcal{E}$, respectively, by taking a mask around the sketched object and discretizing its boundary (see Figure 3.5c). Also, in order to keep the size of the triangles approximately the same, we avoid large triangles by adding vertices to $\mathcal{V}$ by uniformly sampling the empty (white) regions inside the mask (see Figure 3.5c). Lastly, the constrained Delaunay triangulation may give us triangles which are outside the mask, and can lead to undesirable constraints by connecting unrelated parts of the sketched object. Therefore, we prune the triangulation by removing these *bad* triangles (see Figure 3.5d). Let us call the triangle mesh obtained as $\mathcal{M}$, and the set of its triangles as $\mathcal{T}$. Further, let us denote the warp function as W, and for any vertex $\mathbf{v}_i$, its warped position be given by $\mathbf{v}_i' = W(\mathbf{v}_i)$. The warp function minimizes the energy function we describe next. Since, W is linear within each mesh triangle, we just need to compute the values of $\mathbf{v}_i'$ for all $i \in [N]$.

Correspondence constraints. Keeping the same notations as before, let $\mathcal{P}$ denote the set of pixels on $\mathcal{S}_0$ which have a correspondence in $\mathcal{S}_1$, and for each $\mathbf{p} \in \mathcal{P}$, we have a corresponding point $\mathbf{q}$ in $\mathcal{S}_1$. The warp should, therefore, satisfy

$$W(\mathbf{p}) = \mathbf{q} \tag{3.8}$$

Let the triangle in which $\mathbf{p}$ is contained be $(j, k, l) \in \mathcal{T}$ and let $\alpha(\mathbf{p}), \beta(\mathbf{p}), \gamma(\mathbf{p})$ be its barycentric coordinates w.r.t the triangle. The least-squares energy term for the

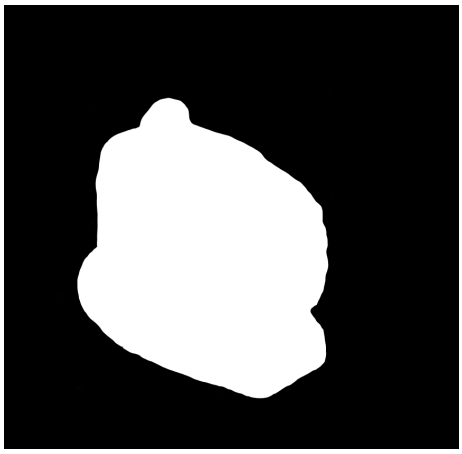
warp constraint is therefore:

$$E_p(W) = \sum_{\mathbf{p}\in\mathcal{P}} \|(\alpha(\mathbf{p})\mathbf{v}'_j, \beta(\mathbf{p})\mathbf{v}'_k, \gamma(\mathbf{p})\mathbf{v}'_l) - \mathbf{q}\|^2 \tag{3.9}$$

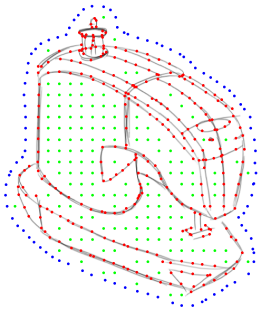
If we also have access to a confidence score for each correspondence $c(\mathbf{p}) \in (0, 1]$, then we can use this as a way to produce a weighted sum of energies for each match

$$E_p(W) = \sum_{\mathbf{p}\in\mathcal{P}} c(\mathbf{p})\|(\alpha(\mathbf{p})\mathbf{v}'_j, \beta(\mathbf{p})\mathbf{v}'_k, \gamma(\mathbf{p})\mathbf{v}'_l) - \mathbf{q}\|^2 \tag{3.10}$$

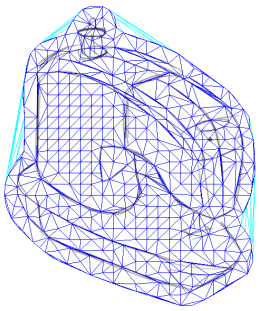
(a) Sketch



(b) Mask



(c) Sampling



(d) Triangulation

Figure 3.5: Generating a constrained Delaunay triangulation for shape preserving warp. a) The input sketch, b) input mask, c) sampled points: junctions and contours (red), mask border (blue), and uniform internal samples (green), and d) triangulation generated with good triangles (blue) and bad triangles (cyan). Observe how sketch contours are traced by triangulation edges.

Note that the matching schemes we discussed, namely, shape context matching and progressive graph matching, provide a confidence score for each match. For manual matching, each match is assigned a constant score of 1.

Triangle shape constraints. Consider a mesh triangle $t = (j, k, l) \in \mathcal{T}$ and attach a local orthogonal frame to it: $\{\mathbf{v}_k - \mathbf{v}_j, R_{90}(\mathbf{v}_k - \mathbf{v}_j)\}$, where R_{90} is a counterclockwise rotation by 90 degrees. Assume that $\mathbf{v}_j$ is the origin of the frame. Now, in the frame, $\mathbf{v}_k$ is simply $(1, 0)$ and let $\mathbf{v}_i = (a, b)$. To preserve the shape of this triangle, we need to ensure that the transformation it goes through is as close as possible to a similarity transformation. Thus, we try to ensure that the local frame remains orthogonal and the coordinates of the vertices remain the same. The energy to express this constraint is

$$E_s(W) = \sum_{t \in \mathcal{T}} \left\| \mathbf{v}'_l - \left(\frac{a}{\|\mathbf{v}_k - \mathbf{v}_j\|}(\mathbf{v}'_k - \mathbf{v}'_j) + \frac{b}{\|\mathbf{v}_k - \mathbf{v}_j\|} \left(\mathbf{v}'_k - \mathbf{v}'_j\right)^{\perp} \right) \right\|^2 \tag{3.11}$$

where

$$a = (\mathbf{v}_l - \mathbf{v}_j)^T (\mathbf{v}_k - \mathbf{v}_j) / \|\mathbf{v}_k - \mathbf{v}_j\| \tag{3.12}$$

$$b = (\mathbf{v}_l - \mathbf{v}_j)^T R_{90} (\mathbf{v}_k - \mathbf{v}_j) / \|\mathbf{v}_k - \mathbf{v}_j\| \tag{3.13}$$

Contour shape constraints. Recall that the segments $\mathcal{E}$ represent the contours (and the mask border). To preserve the shape of the contours, we add an additional constraint requiring the contours to undergo a locally shape preserving transformation. This is another smoothness term, and is similar in spirit to E_s, but is defined on a contour, instead of a continuous 2D domain. Consider the set of pairs of segments which share a single vertex, $\{(E_i, E_j) \in \mathcal{E} \times \mathcal{E} \mid |E_i \cup E_j| = 3\}$. If $E_i \cup E_j = \{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\}$ and $E_i \cap E_j = \mathbf{v}_1$, let us denote $(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$ as e. Borrowing terminology from [CSD11], we call such a sequence representing two consecutive edge segments as an *edgelet*. A similarity (shape preserving) transformation of an edgelet e would preserve the angle θ between the two edges, as well as the length

ratio $\|\mathbf{v}_0 - \mathbf{v}_1\|/\|\mathbf{v}_2 - \mathbf{v}_1\|$. We can write this constraint as an energy term as

$$E_b(W) = \sum_{e \in \tilde{\mathcal{E}}} \left\| (\mathbf{v}'_0 - \mathbf{v}'_1) - \frac{\|\mathbf{v}_0 - \mathbf{v}_1\|}{\|\mathbf{v}_2 - \mathbf{v}_1\|} R_\theta (\mathbf{v}'_2 - \mathbf{v}'_1) \right\|^2 \tag{3.14}$$

where $\tilde{\mathcal{E}}$ is the set of edgelets, and R_θ is a counterclockwise rotation by θ.

Energy minimization. The total warp energy is taken to be the total sum of all the three constraint energies.

$$E(W) = w_p E_p + w_s E_s + w_b E_b \tag{3.15}$$

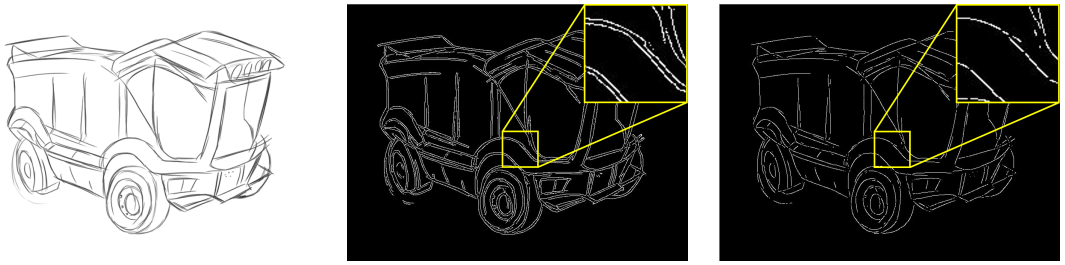
The values of these coefficients are given later, when we describe the experiments in chapter 4. Since, all the energy terms are in terms of squared residuals, and the system of equations is overdetermined (that is, there are more equations than unknowns), this is a standard least-squares optimization problem. We use the QR decomposition to solve the system. In practice, we just use the MATLAB programming language's '\' operator [The], and it automatically chooses the QR decomposition method for solving the system.

In-between warps and extrapolation. If, instead of warping $\mathcal{S}_0$ completely towards $\mathcal{S}_1$, we just need to warp it halfway, or in general, warp it $\alpha \in [0, 1]$ way towards $\mathcal{S}_1$, we just take α factor of the motion vectors for all $\mathbf{v} \in \mathcal{V}$. That is,

$$W_\alpha(\mathcal{S}_0) = \alpha W(\mathcal{S}_0) \tag{3.16}$$

Observe that $W(\cdot) \equiv W_1(\cdot)$. Notice that this assumes that the points follow a linear path while being warped. The other method will be to first move the correspondences to their positions at time $t = \alpha$ (say, $\mathcal{S}_0$ is at time $t = 0$, and $\mathcal{S}_1$ is at $t = 1$), and then compute the warp. But, this approach is more costly, and doesn't ensure that the warp is temporally coherent.

In a similar fashion, we can have $\alpha < 0$ or $\alpha > 1$, and keep $w_\alpha(\mathcal{S}_0) = \alpha W(\mathcal{S}_0)$, and allow for extrapolation.



(a) Original image

(b) Structured forests edge detector [DZ13]

(c) Proposed detector

Figure 3.6: Edge detection: comparison of the proposed edge detection method with state of the art natural image edge detection method. Inset: notice the double edge detected in (b), in contrast to the single edges in (c).

3.3.4 Edge detection

Figure 3.6 gives a comparison of two edge detectors for sketches. The first one is a state-of-the-art edge detector [DZ13] used for natural images. Notice the double edges. The second one is an edge detector we designed based on the popular Canny edge detector [Can86]. Though not as good as modern detectors like [DZ13], the Canny edge detector is an efficient and useful edge detector. It uses the following steps for edge detection

1. Remove noise by applying a small Gaussian filter on the image.
2. Find intensity gradients (magnitude and direction) of the image.
3. Apply non-maximum suppression on the intensity gradient magnitude image [1] in the principal gradient directions to thin edges, and to remove false edges.
4. Apply double threshold on edge response to find certain edges (called strong edges) and potential edges (called weak edges).
5. Track edges by hysteresis: All weak edges connected to a strong edge are kept, while the rest are removed.

Note that the intensity gradient of a natural image gives its contours, which is precisely the information already contained in a sketch, albeit inverted in terms of color. Therefore, we replace the usage of intensity gradient magnitude image in

[1] The intensity gradient magnitude image G_I of a given image I is just an image having the same size as I such that for all pixels $\mathbf{p} \in I$, $G_I(\mathbf{p}) = |\mathbf{G}(I, \mathbf{p})|$, where $\mathbf{G}(I, \mathbf{p})$ gives the intensity gradient of I at pixel $\mathbf{p}$.

step 3 with an inverted color image of the sketch. We keep the direction the same as regular canny detector. The intuition is that if we consider a sketch S to be equivalent to the intensity gradient magnitude image of a natural image $\mathcal{S}_{nat}$, then the intensity gradient directions at a given pixel $\mathbf{p}$ in the Gaussian smoothed version of $\mathcal{S}$ are the same as the directions of the Laplacian of Gaussian of $\mathcal{S}_{nat}$ at p. The Laplacian of Gaussian itself is used for edge detection and sharpening. Therefore, it makes sense to not modify the intensity direction computation in the original Canny detector. Let us call our edge detector as the *modified Canny* detector. For all our experiments in this thesis, we use this edge detector only, unless stated otherwise.

3.4 Iterative match-warp

Till now, we considered matching and warping as independent processes. However, it can be noted that both have the same goal of finding a dense matching between the two sketches. While the matching step gives a sparse correspondence, warping densifies it. We give an iterative framework of alternating match-warp steps to produce a final warp. Firstly, we use edge detection and sampling, as described in section 3.2.2, to generate a set of N points $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ representing the shape in $\mathcal{S}_0$. Given, the sketch $\mathcal{S}_0$, and a small set of correspondences manually specified by the user, we first use the manual correspondences as an input to the shape preserving warp algorithm. Then, we iteratively perform these two operations for a constant number of iterations:

- Use shape context based matching on the set of warped vertices $W(\mathbf{p})$, $\forall \mathbf{p} \in \mathcal{P}$ without any regularization, as described in Section 3.2.2, to get a set of sparse correspondences along with their confidence scores, and
- Feed these correspondences and their confidence scores, along with the manual correspondences, to the shape preserving warp.

A more formal description is given in Algorithm 1. All but one function used in Algorithm 1 have been described earlier in the text. We describe the one remaining function, WarpedPositions($\mathcal{P}, \mathcal{M}, \mathcal{M}'$). For all points $\mathbf{p} \in \mathcal{P}$, the function

returns their coordinates in the warped mesh $\mathcal{M}'$ having the same connectivity list as $\mathcal{M}$. This is done by simply taking the position of the point $\mathbf{p}$ in $\mathcal{M}$; given by a pair (t, B), where $t \in \mathcal{T}$, and $B = (\alpha, \beta, \gamma)$ is the barycentric coordinate of $\mathbf{p}$ in t; and finding the coordinate corresponding to the same position in $\mathcal{M}'$.

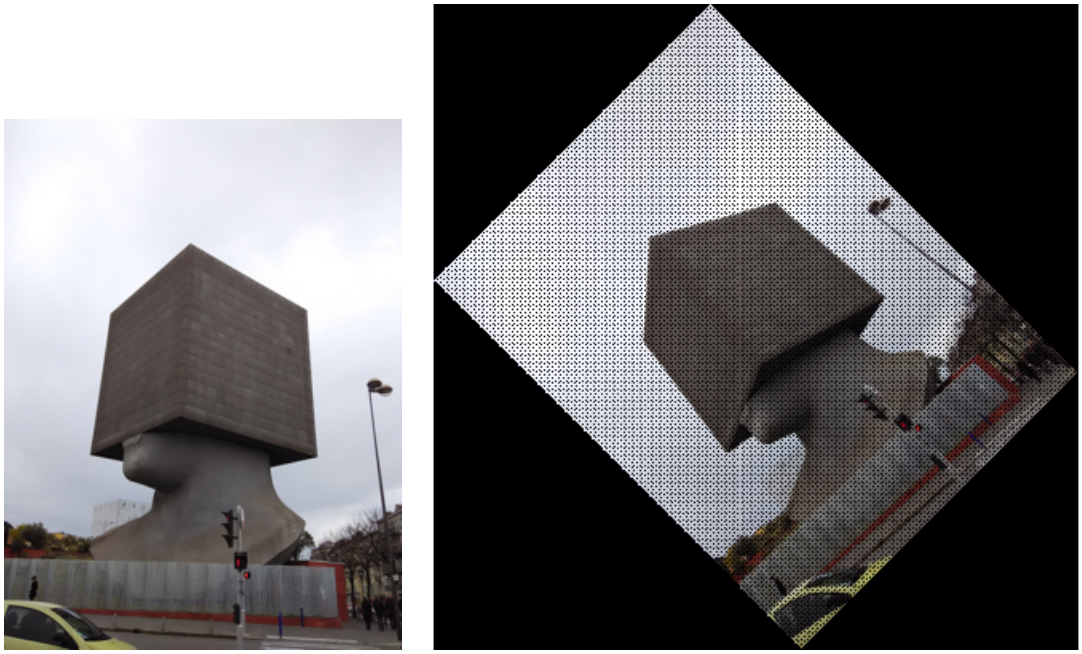
In our experiments, **this strategy gives the best results**. Detailed comparisons are outlined in Chapter 4. Note that it is also easy to specify occlusions or missing parts in this algorithm. Suppose that some region of $\mathcal{S}_0$ is occluded in $\mathcal{S}_1$, or is a part of the shape depicted by $\mathcal{S}_0$ which is absent in $\mathcal{S}_1$. Then, the user can just specify an occlusion mask to specify all such regions. While sampling for $\mathcal{P}$, we only sample points which lie outside the occlusion mask. Note that while sampling for $\mathcal{V}$, we do not care about the mask, as we would like even the occluded regions to warp in a shape preserving manner.

Algorithm 1 IterativeMatchWarp

function IterWarp($\mathcal{S}$, $\mathbf{M}^0$, $\mathcal{P}$, $\mathcal{Q}$) ▷ $\mathbf{M}^0$ is the set of manual matches
  $\mathcal{S}_E \leftarrow$ DetectEdges($\mathcal{S}$) ▷ $\mathcal{S}_E$, a binary image, is the edgemap of $\mathcal{S}$
  $\mathcal{V} \leftarrow$ SampleEdgePoints($\mathcal{S}_E$)
  $\mathcal{E} \leftarrow$ EdgeSegments($\mathcal{S}_E, \mathcal{V}$)
  $\mathcal{M} \leftarrow$ CDT($\mathcal{V}, \mathcal{E}$) ▷ Constrained Delaunay Triangulation
  $\tilde{\mathcal{E}} \leftarrow$ Edgelets($\mathcal{V}, \mathcal{E}$)
  $\mathcal{M}^0 \leftarrow$ ShapePreservingWarp($(\mathbf{M}^0, (1, 1, \ldots |\mathbf{M}^0| \text{ times})), \mathcal{M}, \tilde{\mathcal{E}}$)
  $\mathcal{P}^0 \leftarrow$ WarpedPositions($\mathcal{P}, \mathcal{M}, \mathcal{M}^0$)
  $i \leftarrow 1$
  while $i < numIter$ **do**
    $(\mathbf{M}^i, C) \leftarrow$ ShapeContextMatching($\mathcal{P}^{i-1}, \mathcal{Q}$)
    $\mathbf{M}'^i \leftarrow (\mathbf{M}^i, C) \cup (\mathbf{M}^0, (1, 1, \ldots |\mathbf{M}^0| \text{ times}))$
    $\mathcal{M}^i \leftarrow$ ShapePreservingWarp($\mathbf{M}'^i, \mathcal{M}, \tilde{\mathcal{E}}$)
    $\mathcal{P}^i \leftarrow$ WarpedPositions($\mathcal{P}, \mathcal{M}, \mathcal{M}^i$)
    $i \leftarrow i + 1$

3.5 Rendering

Notice that we have obtained a warp from $\mathcal{S}_0$ to $\mathcal{S}_1$ in the form of a warping function W which gives the warped position for every pixel of $\mathcal{S}_0$. However, in order to create the warped image $\mathcal{S}_{01}$, we need a mapping from every pixel of $\mathcal{S}_{01}$ to some pixel in $\mathcal{S}_0$. In technical terms, what we have obtained is known as a *forward mapping* but



(a) **(b)**

Figure 3.7: The forward mapping problem. Image (b) is generated by rotating (a)[2] by 45° and the colors of all pixels in (a) are forward mapped to pixels in (b). Notice the holes present in the rotated image. This problem is solved using backward mapping.

what we need is a *backward mapping*. Just solving for the forward mapping gives us an image with holes in it. Figure 3.7 explains the problem. In this section, we describe a method to solve for the backward mapping.

To reiterate, the problem we are trying to solve is: Given any pixel q of $\mathcal{S}_{01}$, what is its color? One way to obtain this is via the warped triangulation. Since, the warping does not effect the triangulation (connectivity list) but only the positions of the mesh vertices, we can find the triangle[3] t containing $\mathbf{q}$ along with the barycentric coordinates of $\mathbf{q}$ w.r.t this triangle, and locate the color of the pixel $\mathbf{p}$ having the same barycentric coordinates in t. This is precisely what we do in the function WarpedPositions() in Algorithm 1. But this is a costly operation, and is not feasible to perform for more than a few hundred pixels (while a typical image is 10^5-10^6 pixels or more). Fortunately, there is a very simple and practical way to

[2]Photograph of La Tête au Carré (The Square Head), Promenade des Arts, Nice, France. © Rahul Arora.

[3]Actually, the point $\mathbf{q}$ can be in multiple triangles since there is no explicit constraint to prevent triangles from intersecting. However, we ignore this issue and take the first triangle (in any arbitrary ordering) which contains $\mathbf{q}$.

avoid this problem: We use OpenGL to render the warped image. The image S_0 is used as a texture for rendering the warped mesh, while using the corresponding source mesh as the texture coordinates. Thus, for any vertex $\mathbf{v}_i = (x_i, y_i) \in \mathcal{V}$, the texture coordinate of its warped position $\mathbf{v}'_i$ is (x_i, y_i), and it therefore picks up the color at (x_i, y_i) in $\mathcal{S}_0$. OpenGL interpolates the texture coordinates inside the triangles to find the color of each pixel, thereby completing the backward mapping.

Now that we know how to get warped images, we just need to know how to morph them together. For this purpose, we use simple alpha blending. Consider the problem of blending together the warped versions of $\mathcal{S}_0$ and $\mathcal{S}_1$. Let the warp function warping $\mathcal{S}_0$ towards $\mathcal{S}_1$ completely be W^{01}, and that warping $\mathcal{S}_1$ onto $\mathcal{S}_0$ be W^{10}. Recall that we want to produce the in-between images for $\alpha \in [0, 1]$. If we know the in-between warps $W^{01}_\alpha(\mathcal{S}_0)$ and $W^{10}_{1-\alpha}(\mathcal{S}_1)$, we can compute the in-between image $\mathcal{S}_\alpha$ as

$$\mathcal{S}_\alpha = (1-\alpha)W^{01}_\alpha(\mathcal{S}_0) + \alpha W^{10}_{1-\alpha}(\mathcal{S}_1) \tag{3.17}$$

In general, given n images $\mathcal{S}_1, \ldots, \mathcal{S}_n$, the warp functions between them, W^{ij} $\forall 1 \leq i, j \leq n$[4], and a n-vector of positive real numbers $\Delta = (\alpha_1, \ldots, \alpha_n)$, we can generate an in-between image S_Δ as

$$\overline{W}^i_\Delta(p) = \sum_{j=1}^{n} \alpha_j W^{ij}(\mathbf{p}) \tag{3.18}$$

$$\overline{\mathcal{S}}_{i,\Delta} = \overline{W}^i_\Delta(\mathcal{S}_i) \tag{3.19}$$

$$\mathcal{S}_\Delta = \sum_{i=1}^{n} \alpha_i \overline{\mathcal{S}}_{i,\Delta} \tag{3.20}$$

If we set $n = 2$ and $\Delta = (1-\alpha, \alpha)$, then the above set of equations reduces to equation 3.17.

Note that we must be able to add various W^{ij} to get W^i. This means that all the warp functions defined on any one image should be compatible with each other. Since our warps W^{ij} are defined in terms of vertices of the triangulation

[4]For any i, the warp function W^{ii} just maps the image to itself, that is, $\forall \mathbf{p},\ W^{ii}(\mathbf{p}) = (\mathbf{p})$, the identity warp

on S_i, we need to make sure that the triangle mesh remains the same when we compute the warps $W^{ij},\ 1 \leq j \leq n$, which can easily be done by pre-computing the triangulation once and using it for computing all the warps. For the implementation of the blending function, we write a shader using the OpenGL Shading Language (GLSL). See Figure 4.3 for an example using three sketches.

Chapter 4

Experiments, Results and Applications

We now present results we obtained using our interpolation pipeline. In order to support our choice of components, we provide comparisons with various other choices, and describe how each option affects the final outcome. We also go through our choice of parameters, described in chapter 3, and motivate the choice of their values on the basis of interpolation results. Then, we compare our method with state of the art image matching algorithm. Finally, we present an application of our method for exploration of design space of ideation sketches.

All the experiments have been carried out on a standard machine with a 2.4 GHz clock Intel Core-i5 CPU including an integrated Intel HD Graphics 3000 graphics processor, 6 GB of memory, and an nVidia Geforce GT 525M GPU. The dataset for the experiments has been created by redrawing sketches picked up from various sources on the web. The redrawing was done in order to remove the shadows and hatching normally present in sketches on the web. However, all the parts of the sketches were drawn, and no contours or details were added or deleted. See Appendix A for details.

4.1 Results

Figure 4.1 presents some of the results of our experiments on morphing between two images. Figure 4.2 presents the same results while showing only the first sketch being warped, without blending with the second. For all of these pairs of sketches, we begin with 5 or 6 correspondences provided by the user, and then run the iterative match-warp algorithm. Recall that the iterated match-warp approach uses shape context as the data constraint for sparse matching, and shape-preserving warp energy as the smoothness constraint (Algorithm 1). Note how our method attempts to preserves the shape of the objects being warped as much as possible. Except in regions of very sharp distortions in the motion field due to the correspondences, our method preserves contour shape well, resulting in smooth curves in the transitional sketches. Figure 4.3 shows the generalization of our algorithm to multiple images. Although the warped images are not perfectly aligned, the in-between sketches look plausible, and give a good idea of how a sketch of an in-between shape/viewpoint should look like. Our algorithm also allows for extrapolation. We give some extrapolation results in figure 4.4.

We also provide an example (figure 4.5) using an occlusion mask along with a comparison of results achieved when no occlusion information is provided. An occlusion mask over a sketch identifies parts which are not present in the other sketch, and should therefore be ignored by the matching algorithm. Notice how the occlusion mask results in the dual benefits of maintaining the shape of the occluded regions better, while also resulting in better alignment of the overall sketches due to better matching produced as a result of decreased confusion for the matching algorithm. The occlusion mask is quick and easy to create using common image manipulation programs such as GIMP or Adobe® Photoshop, and can be used to produce higher quality results when the user desires it.

Figure 4.6 shows the effect of changing the number of iterations of the iterative max-warp algorithm on the quality of the warp produced. Notice how the alignment changes dramatically from the first matching (using manual matches only) to 2 it-

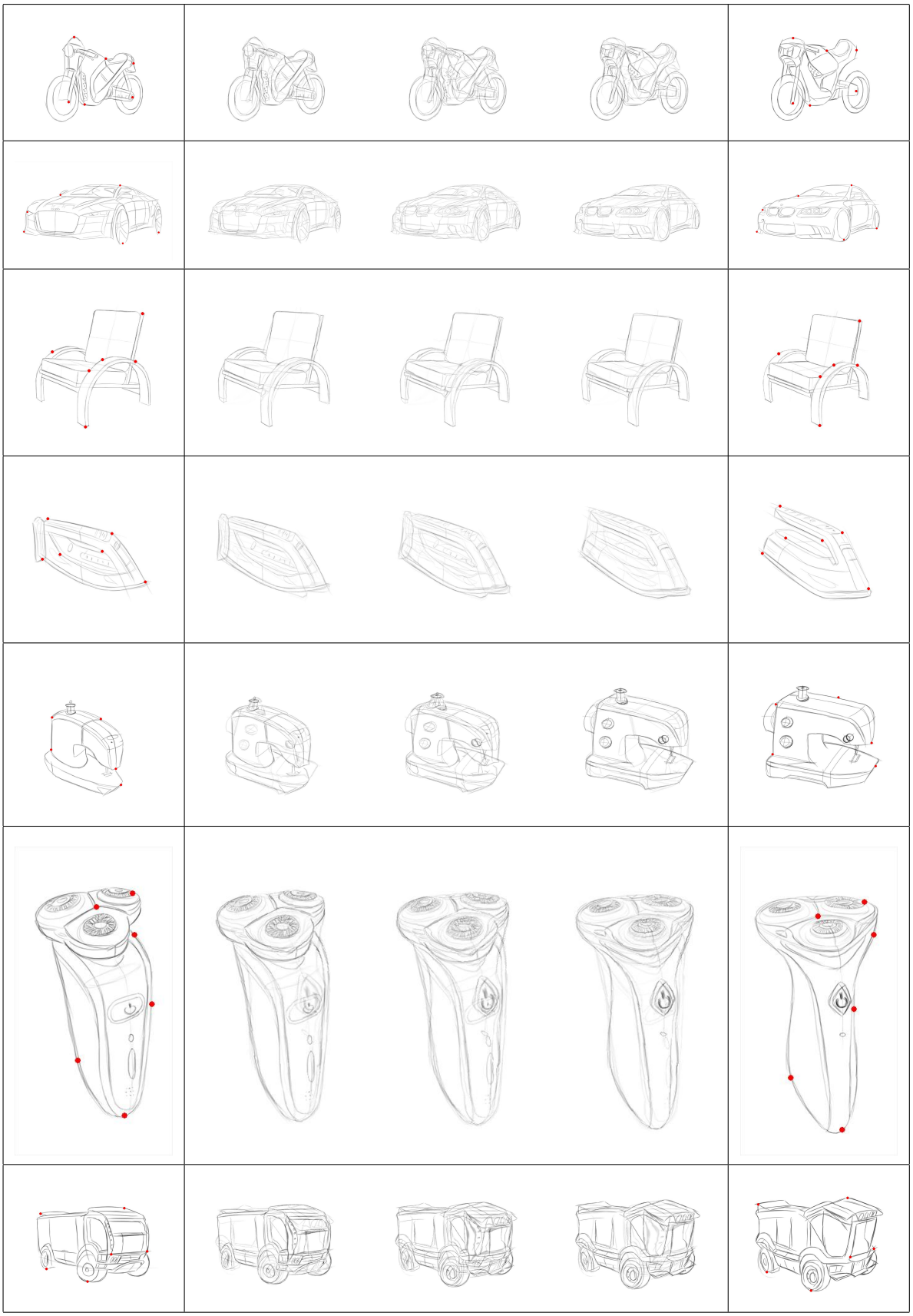


Figure 4.1: Morphing images warped using iterated match-warp algorithm. From column 1 to 5: $\alpha = 0$ (source sketch), $0.25, 0.5, 0.75$, and 1 (target sketch). Red dots on the source and target sketches indicate correspondences provided by the user.

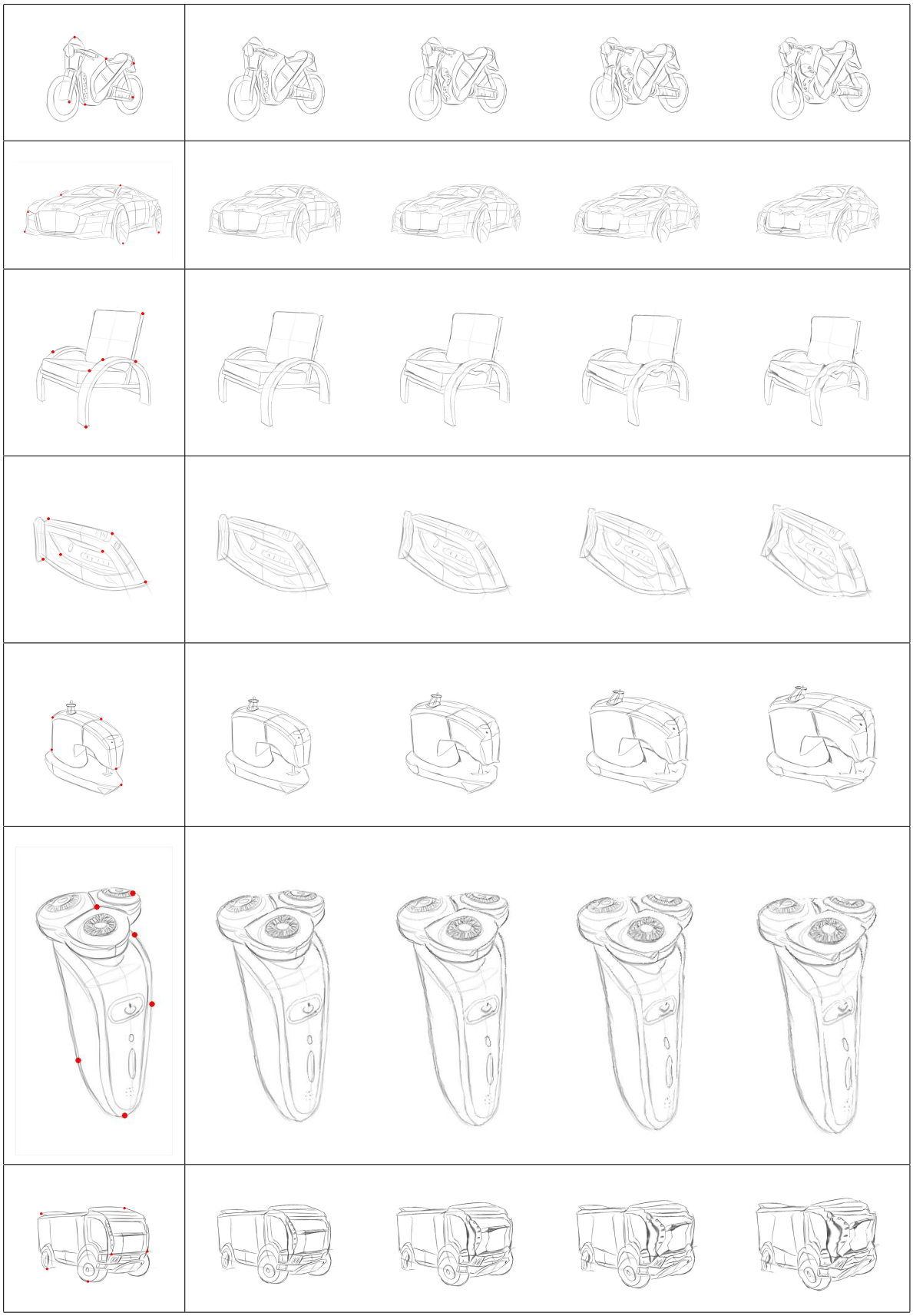


Figure 4.2: Warping using iterated match-warp algorithm. Column 1: Original sketch ($\alpha = 0$). Columns 2 to 5: $\alpha = 0.25, 0.5, 0.75,$ and 1 (target sketch). Notice how the contour shape is preserved in most of the regions. Red dots shows points matched to the target sketch by the user.

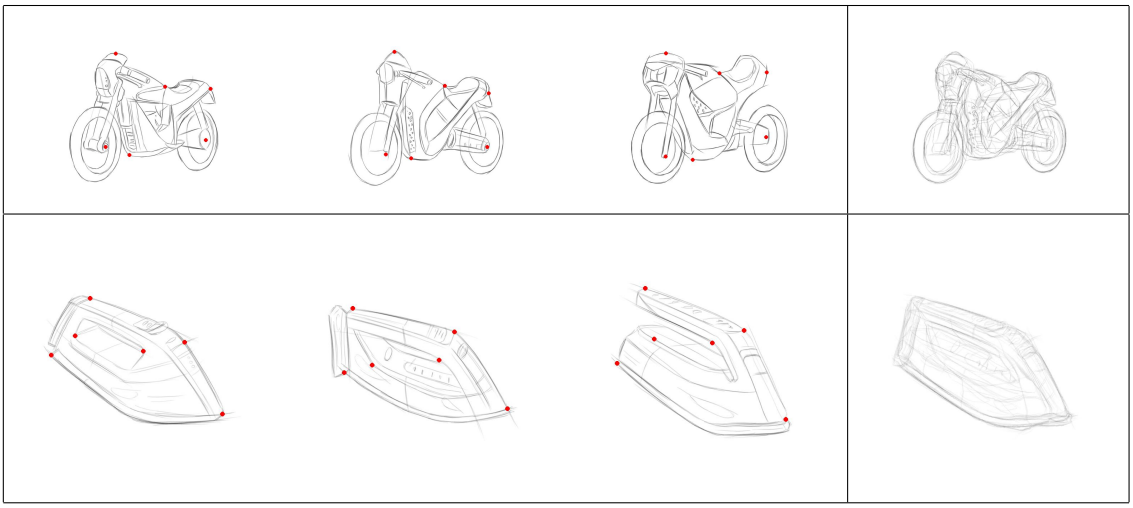


Figure 4.3: Morphing multiple images. Columns 1 to 3: Input images. Column 4: Morphed image at $\Delta = (1/3, 1/3, 1/3)$.

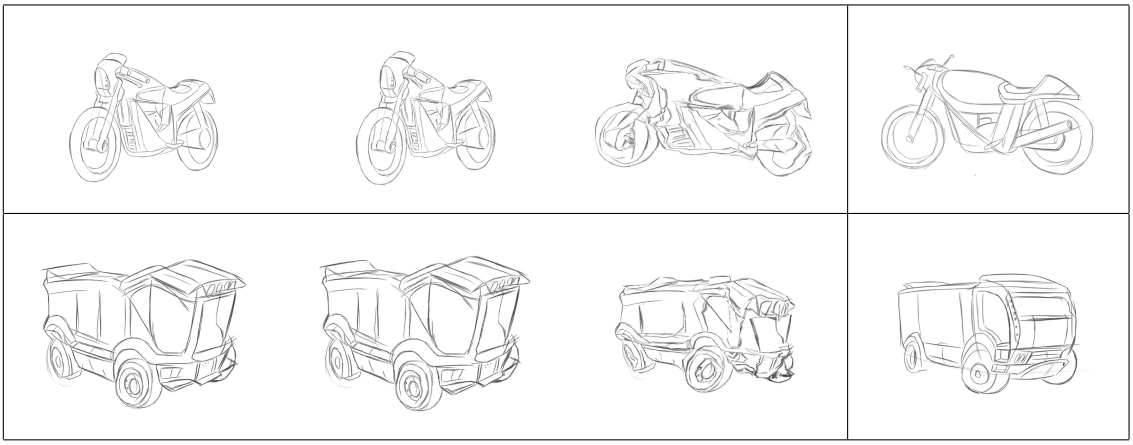


Figure 4.4: Extrapolating sketch warps. Columns 1 to 3: source sketch at $\alpha = 0$, $\alpha = -0.25$, and $\alpha = 1.25$. Column 4: target sketch.

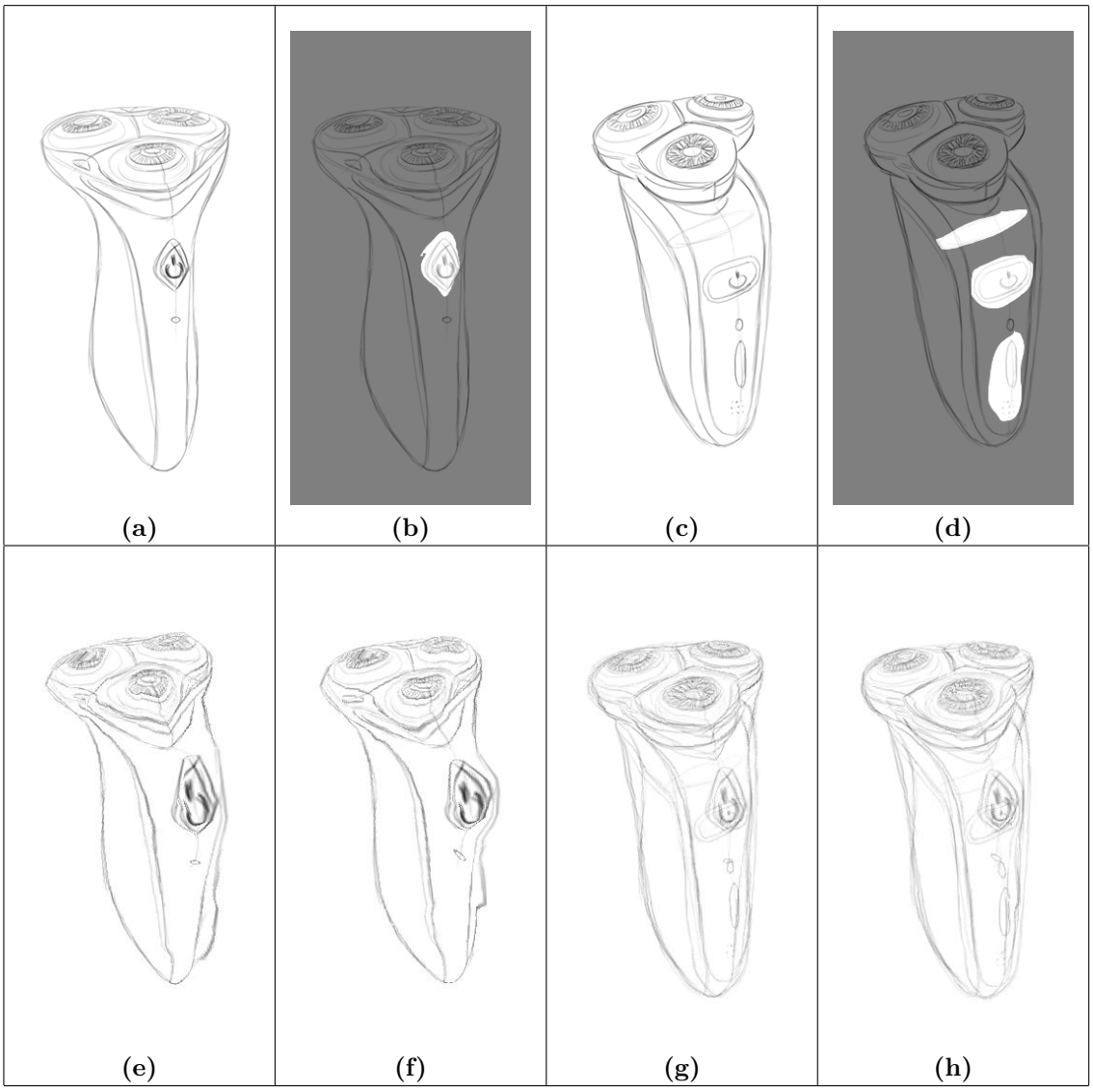


Figure 4.5: Effect of using an occlusion mask. Clockwise from top-left: a) Source sketch and b) its occlusion mask, c) target image and d) its occlusion mask, warped target sketch at $\alpha = 1$ e) when occlusion masks are used vs. f) when not, and, morph between the two sketches at $\alpha = 0.5$ g) with and h) without using occlusion masks. Notice the improved alignment and shape preservation when occlusion masks are taken into consideration by the algorithm.

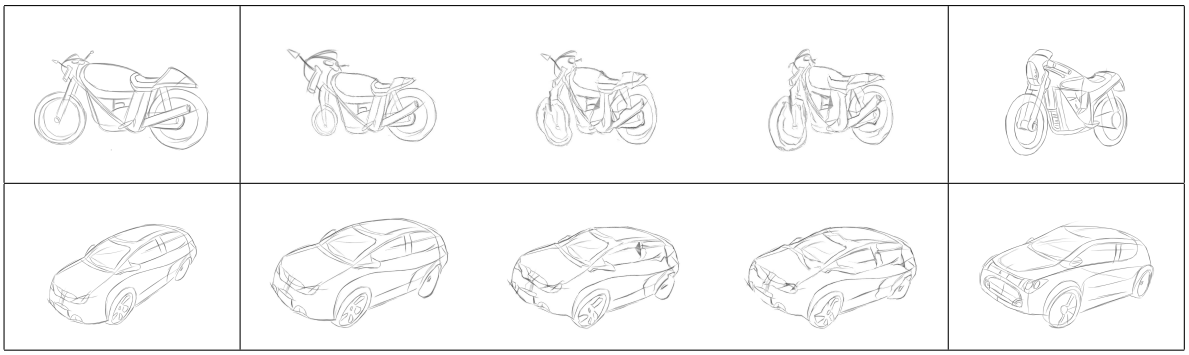


Figure 4.6: Results using various values of $numIter$. Column 1: Source mesh, column 5: target mesh, columns 2 to 4: warped mesh at $\alpha = 1$ using 0 iterations, 2 iterations, and 5 iterations.

erations. In general, we observed that most of the sketch alignment is done within the first two iterations, and further iterations provide diminishing returns. Therefore, we set the number of iterations to a slightly conservative value of 4, although changing it to any value between $2 - 6$ does not have any significant effect on the results.

We set the other parameters of the shape preserving warp, the energy coefficients w_p, w_s and w_b to 5, 2 and 10, respectively. Figure 4.7 shows the results when w_p is set to 1, 5 or 10. We can note that while a small value of $w_p = 1$ results in poor alignment, since the correspondences do not count for much, keeping too high a value of $w_p = 10$ doesn't allow shape to be preserved and results in local discontinuities in the motion field. We do not show results for varying the value of w_s vis-à-vis w_b, but the value of w_b is kept much higher as compared to w_s because we do not, in general, want the triangle shape to be preserved too much to allow for easy expansion or contraction in regions where there are no contours, and focus the shape preserving energy on the contours.

4.1.1 Choice of components

Now, we present some results using the components we discussed in chapter 3 but didn't employ as the final option, and contrast the results obtained using those components with our final choice, i.e., iterative match-warp. Figure 4.8 lists some results using these approaches:

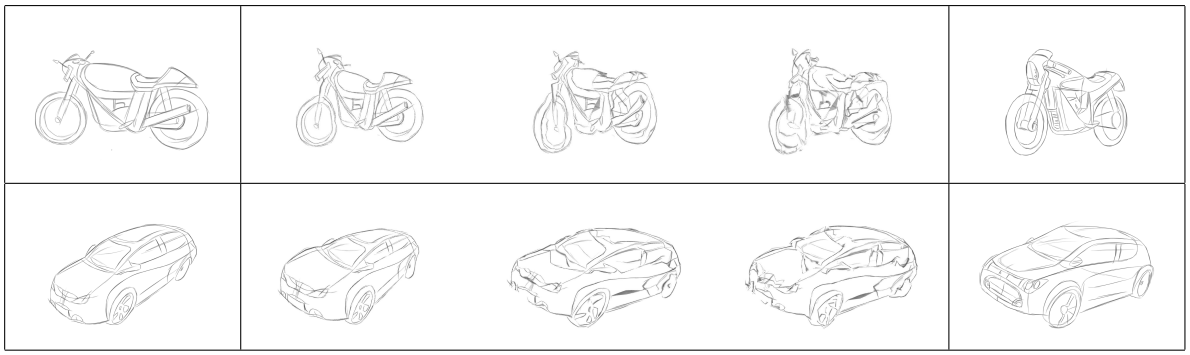


Figure 4.7: Results using various values of w_p. Column 1: Source mesh, column 5: target mesh, columns 2 to 4: warped mesh at $\alpha = 1$ using $w_p = 1$, $w_p = 5$, and $w_p = 10$. Notice how the middle column shows good alignment without too much shape distortion. For all these results, $w_s = 2$ and $w_b = 10$.

- When the warp is performed with a naive approach using only 5-6 sparse manual correspondences, and using linear interpolation to create a dense backward mapping. Notice how the alignment with the target sketch is really poor, albeit the shape is preserved since no significant warping happens.
- Iterative algorithm using shape contexts for matching and thin plate splines for regularization as described in [BMP02]. This approach preserves shape, but is unable to align the sketches well in regions of large movement.
- Progressive graph matching to extend the initial list of 5-6 manual correspondences followed by a single iteration of shape preserving warp[1]. The major disadvantage of this approach is the heavy local distortion caused by bad matches.

4.1.2 Comparison with other work

We compare our results with state of the art image morphing algorithm presented in [Lia+14] in figure 4.9. The figure also describes results using an approach which uses a large number of user correspondences (16-25) followed by uses shape preserving warp. Due to the large number of user correspondences provided, we treat these results as the reference warp/morph. Comparison with this reference shows that the warps generated by our approach are better aligned to the target sketches, and

[1]Since the shape preserving warp runs only once in this approach, to get proper alignment, we set $(w_p, w_s, w_b) = (10, 1, 10)$.

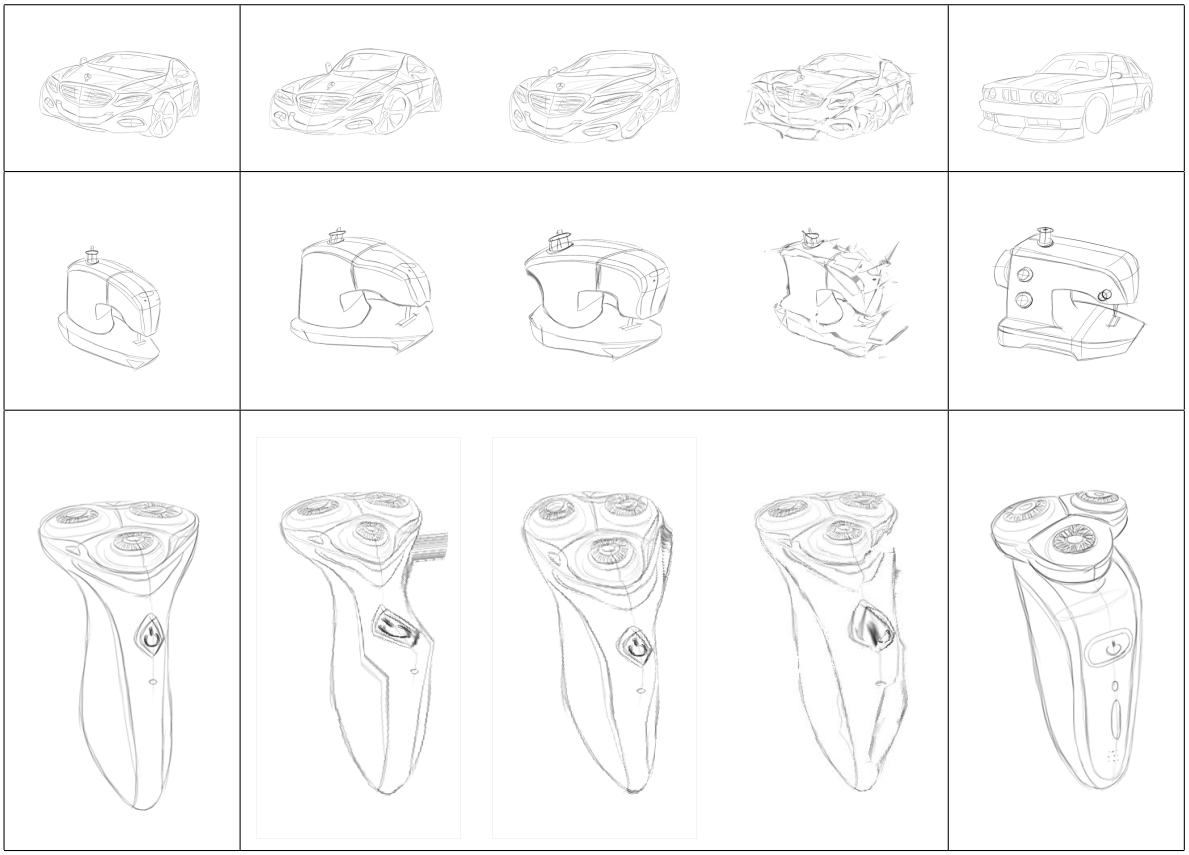


Figure 4.8: Results using component choices other than iterative match-warp. From left to right: source sketch; warped sketch at $\alpha = 1$ using very sparse manual matches and linear interpolation only, shape contexts with TPS, and, progressive graph matching; and target sketch.

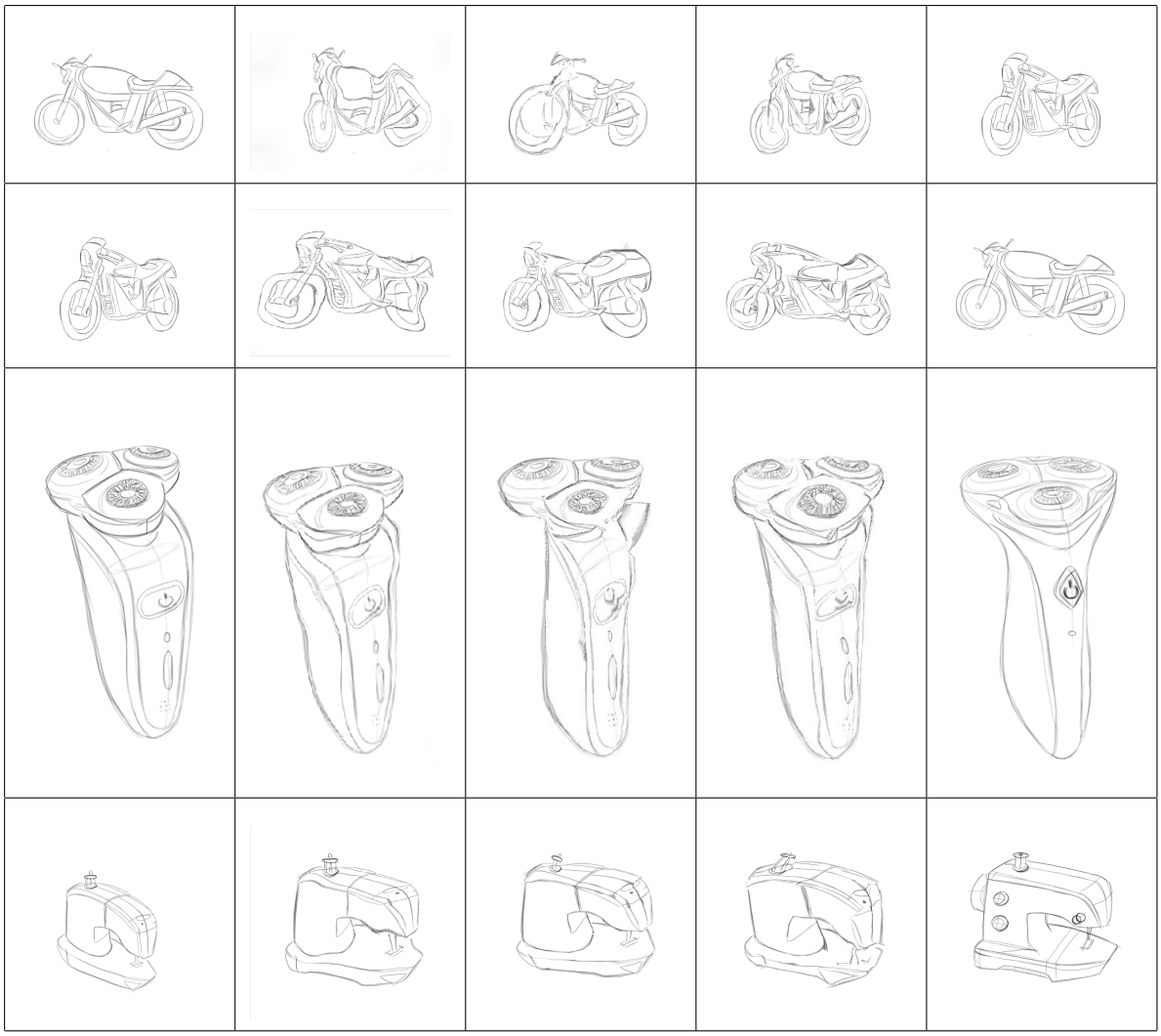


Figure 4.9: Comparison between proposed method and state of the art solution. Columns 1 to 5: source sketch, warped sketch at $\alpha = 1$ using [Lia+14], reference warps, using our approach, and target sketch.

preserve shape better. Also, the existing method produces wavy contours. This is because our approach is better suited to the sketchy domain, extracting and using important sketch information of contours, and preserving their shapes. Moreover, our approach runs much faster.

4.1.3 Analysis and failure cases

Our results for ideation sketches of varied sketch styles and object domains shows that the method serves its intended purpose well. The comparison with state of the art image morphing algorithm also proves that the current natural image morphing and interpolation methods are not directly applicable for sketches, and the adap-

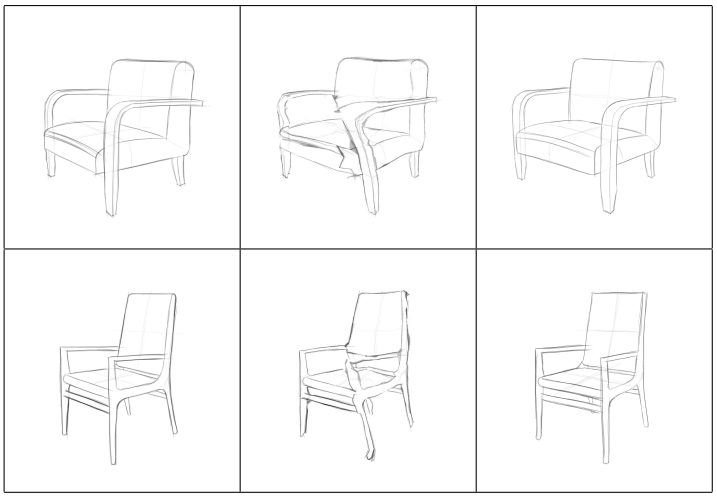


Figure 4.10: Failure due to self-intersecting projections. From left to right: source sketch, source sketch warped to $\alpha = 1$, and target sketch.

tations we make improve the results. A strength of our method is that we do not require the three dimensional shape of the object being depicted, due to the inherent projection inaccuracies in ideation sketches. Another key advantage is that we need little user effort, which is an important constraint at the early stage of design we target.

However, due to the fully two-dimensional nature and little user interaction, the proposed solution fails for certain cases. This may occur because the two-dimensional motion field cannot possibly capture the correct three dimensional motion-field due to large self intersections in the sketched projection, as in figure 4.10, or because the shape difference between the objects is too high, as exemplified by figure 4.11.

We now move on to the application we developed for designers to explore the design space of ideation sketches, before returning to the failure cases and discussing possible solutions in chapter 5.

4.2 Application: Design space exploration

We describe a tool which works on top of the proposed sketch interpolation algorithm to allow designers to explore the design space induced by their ideation sketches.

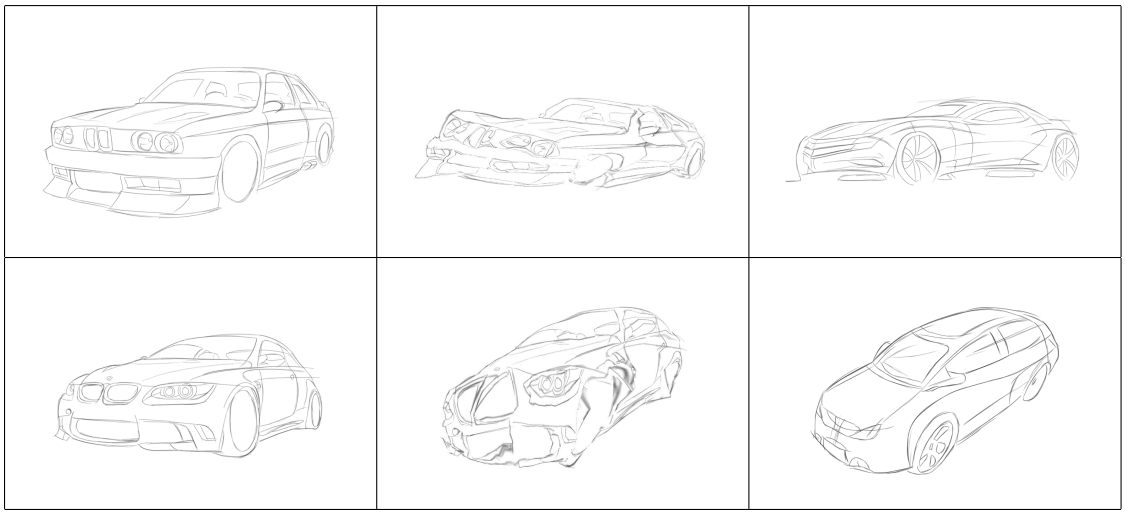
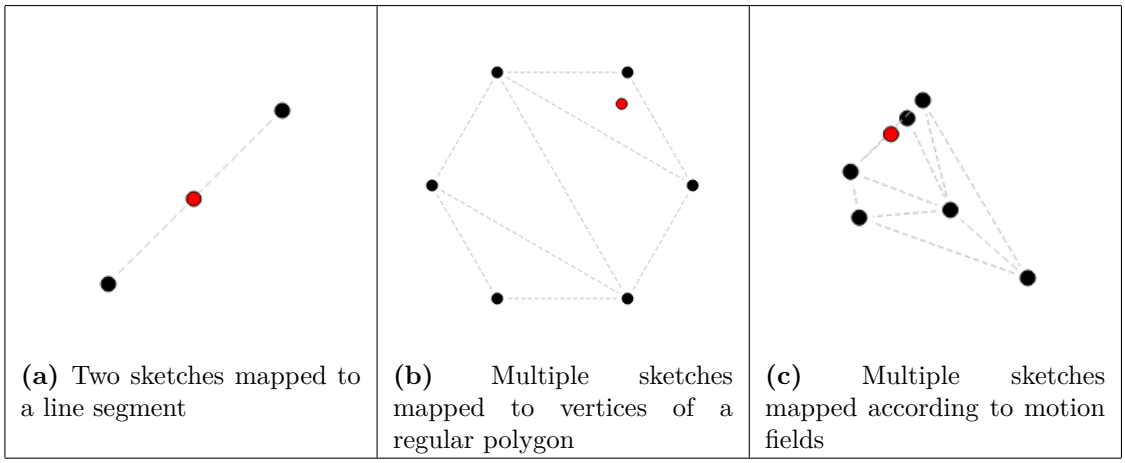


Figure 4.11: Failure due to large shape difference. From left to right: source sketch, source sketch warped to $\alpha = 1$, and target sketch.

The tool can help designers understand possible variations which can be generated from the existing sketches by generating animations visualizing inbetween sketches and generating the inbetween sketches themselves, which the designer can choose to later improve or modify.

Figure 4.12 gives a description of the user interaction with the tool. For two sketches, the tool provides the designer with an option to interpolate (or extrapolate) between them by choosing a point on a line representing the alpha values with one of the input sketches at $\alpha = 0$ and the other at $\alpha = 1$. For multiple (more than two) sketches, the sketches are mapped to vertices of a planar Delaunay mesh. The plane in which the triangle mesh lies is our idea of the *design space* induced by the input sketches. The user can choose any location on the plane of this mesh (may or may not be inside the mesh boundaries) and can look at the sketch generated at this point. The user can also choose any two points on the plane, and the tool generates an animation by iterating through the sketches represented by the segment joining these two points. In case of two sketches, this animation can be generated between two points on the line drawn through points representing the input sketches.

The user can also make a choice on which of the input sketches should be morphed in order to produce the interpolated sketch. Note that by choosing only one sketch, this reduces to just looking at the warped version of that sketch. Another option



(a) Two sketches mapped to a line segment

(b) Multiple sketches mapped to vertices of a regular polygon

(c) Multiple sketches mapped according to motion fields

Figure 4.12: Mapping sketches into the plane representing the design space. The black dots are input sketches, and the red dot is a point in the design space currently being explored by the user.

provided to the user is with respect to how the sketches are arranged (see figure 4.12). One way is to arrange them as a regular polygon: given n sketches, they are mapped to vertices of a n-vertex regular polygon in the design space. This is useful when the sketches differ just in terms of shape and not viewpoint, since there is no physical prior on how they should be arranged in a plane. The other method is to place the sketches in the design space based on the motion fields between them. This is useful for sketches differing in the projection viewpoint (and, possibly, in shape). Arranging sketches based on their motion fields gives a rough idea of the relative three dimensional rotation between the sketches, and can guide the user in generating new viewpoints (see figure 4.13).

For two sketches, both the arrangements are the same. Therefore, we focus on $n > 3$. We present a least-square optimization to solve for the optimum positions of the sketches in the design space, given their relative motion fields.

Given $n > 2$ sketches $S_1, \ldots, S_n$, and their relative motion fields $\mathcal{F}_{ij}$, $\forall\ 1 \leq i, j \leq n$, we first compute the mean motion field between all sketch pairs. For any given pair (i, j) (where $1 \leq i, j \leq n$), the mean motion field $\bar{\mathcal{F}}_{ij}$ is given by taking the mean of motion vectors of all the vertices in the triangulation used to warp sketch S_i to S_j. Let us also denote the component of the mean motion field in the x direction as $\bar{\mathcal{F}}^x_{ij}$, and that in the y direction as $\bar{\mathcal{F}}^y_{ij}$. Now, for all pairs (i, j) such

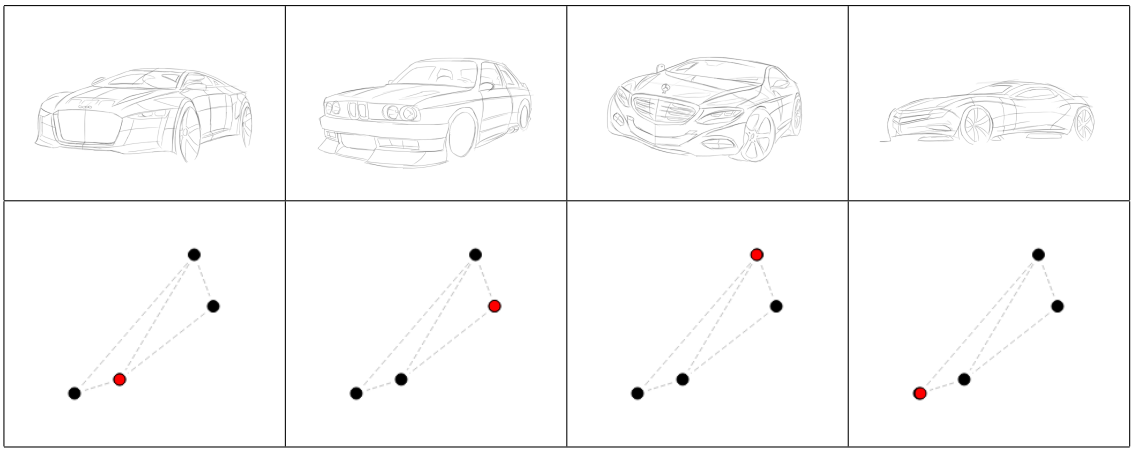


Figure 4.13: Mapping sketches to design space according to relative motion fields. The top row shows the input sketches, while the red dot in bottom row shows the position of each sketch in the plane. The average motion field direction can be inferred by looking at the parts of the sketches. For example, while moving from column 2 to 3, a visual inspection hints that the motion field is towards the top-right, which is what the mapping suggests as well. Other pairs can be similarly observed.

that $1 \leq i < j \leq n$, define

$$f^{dir}(i,j) = \frac{1}{2}\left(\mathcal{F}_{ij}^{dir} + (-\mathcal{F}_{ji}^{dir})\right) \;\; \forall dir \in \{x, y\}$$

Now, let (x_i, y_i) be the position of S_i in the design space. Assume that the point (x_1, y_1) is the origin, that is, $x_1 = y_1 = 0$. We solve for (x_i, y_i) $\forall$ $1 \leq i \leq n$ by setting up a least squares optimization over the following equations

$$x_j = f^x(1,j) \qquad \forall\ 2 \leq j \leq n \tag{4.1}$$

$$y_j = f^y(1,j) \qquad \forall\ 2 \leq j \leq n \tag{4.2}$$

$$x_j - x_i = f^x(i,j) \qquad \forall\ 2 \leq i < j \leq n \tag{4.3}$$

$$y_j - y_i = f^y(i,j) \qquad \forall\ 2 \leq i < j \leq n \tag{4.4}$$

Since we have $n(n-1)$ equations and $2(n-1)$ unknowns, we can find the globally best solution using the least squares method for all values of $n \geq 2$. After finding this solution, we normalize the values of (x_i, y_i) to restrict to the unit square between $(0,0)$ and $(1,1)$. See figure 4.13 for an example of how the solver arranges the sketches in the design space.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we have presented a novel algorithm for interpolation between ideation sketches depicting a functional class of objects using different shapes and projection viewpoints. The algorithm closely follows a typical image morphing pipeline, but all the components are adapted to suit the input domain of rough sketches. The main visual criteria we used to judge the quality of a warp or morph is the alignment between sketches being warped, and the preservation of shape of the warped images, which makes the interpolated contours look plausible. The proposed approach performs better than all other approaches studied in the paper, as well as the state of the art image morphing method. We also described extensions of the main algorithm to handle extrapolation, and occlusions/disocclusions.

Further, we built a tool to help designers explore the various possible variations generated from a small set of ideation sketches of a single concept. The tool allows the designer to mix and match between the visual and geometric properties of the input sketches. The main user interface aspect of the tool is an embedding of the sketches as points on a plane. The tool also allows the embedding to be based on the warp motion field between the sketches, and relative positions of the sketches in the plane identify with the 3D viewpoint rotations between them.

5.2 Future work

An immediate future work is to improve the rendering by using solutions which work better than simple alpha blending. As is evident from the results shown in figures 4.1 and 4.3, alpha blending can result in reduced darkness when the alpha values are far from the extremes. An idea is to generate multiple versions of each sketch using a grayscale thinning operation, and then warping the required version according to the blending alpha value of the sketch. This is suitable for sketches since thinner lines represent reduced importance of a contour [Hla14]. Another improvement is on alignment: once the warps have been generated, the novel sketch that the user explores can be further refined by using iterative closest point (ICP) approach to iteratively align the warped edgemaps of the two sketches better. While this might result in loss of coherence along the design space, the generated sketches can potentially align much better. The edge detector we use does not work with sketches with shading, hatching and shadows. A possible improvement to the whole pipeline is to pre-filter the sketches using, for example, the rolling guidance filter described in [Zha+14], before the edge detection step. Another improvement will be to have a good measure of alignment of the sketches, and run the iterative warp until the alignment threshold has been crossed, instead of a fixed number of iterations.

In the longer term, we would like the method to be able to handle more general cases, especially the self-intersection failure cases as shown in figure 4.10. A possible modification is to run the algorithm separately for different *layers* of the sketches. For example, in figure 4.10, the right handle of the chairs can be treated as the *top* layer, and the rest of the parts as the *bottom* layer. This would enable the warp to model three-dimensional transformations which project to a two-dimensional transformation with significant discontinuities by separating out the 3D transformation into multiple 2D transformations. We would also like to improve user experience by reducing the number of correspondences she has to provide. One way to do this is to ask the user for correspondences between only a few pairs of sketches and then enforce cyclic consistency constraint on the matches to deduce correspondences

between the other pairs.

The ultimate goal is to use the matches generated and the warping transformations estimated to recover the 3D shape of the depicted object. This is a hard problem because of the inherent inaccuracies in ideation sketches as discussed in the text. But, one can hope to recover the general 3D shape represented by the sketches in order to aid the designer in the 3D modeling step, thereby reducing repeated artist effort.

Appendix A

Redrawing of Sketches

Since our method cannot work with sketches which include hatching or shadows, we have redrawn sketches originally drawn by professional sketch artists/designers by retracing the contours. We have tried to stay as honest as possible to the original drawings, taking care not to introduce or delete any contours. See figures A.1 and A.3 for a side-by-side comparison of all sketches in our dataset, along with the originals. Some sketches (figure A.2) were drawn by tracing edges of rendered 3D models taken from SketchUp 3D WareHouse [Tri].

Sketch sources:

- © Mike Serafin. `http://www.memikeserafin.com/110810/970511/work/sketchbook`. (All clothes iron images)
- © Spencer Nugent. Sketch-A-Day 286. (All shaver and sewing machine images)
- © Benjamin R. Lloyd. `http://www.benjaminrlloyd.com/sketchbook`. (All rally truck and motorcycle images, and cars 6, 7 and 8)
- All chair 3D models taken from SketchUp 3D Warehouse [Tri].
- © `http://dazza-mate.deviantart.com`. (Car 1 and 5)
- Mercedes Benz 2014 S-Class Coupe. Press release. (Car 2)
- © TJ Vaninetti. `https://thepureautostudio.wordpress.com/`. (Car 3)
- Audi Quattro concept car. Press release. (Car 4)

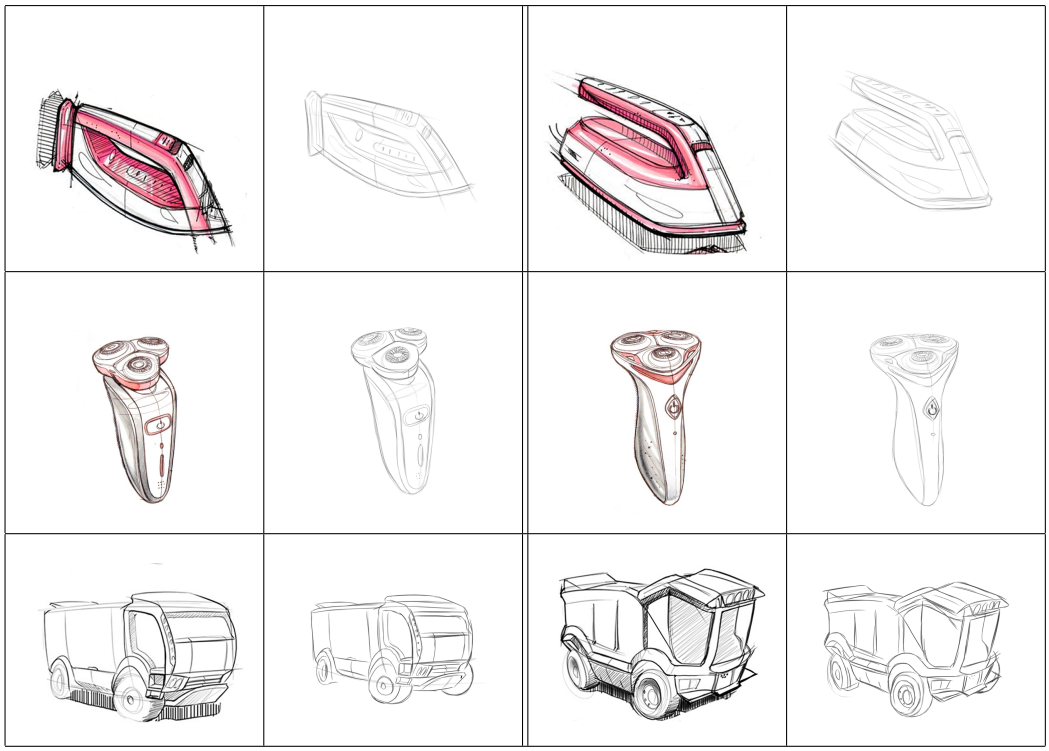


Figure A.1: Original sketches with their redrawn versions.

Figure A.2: Sketches drawn using 2D projections of stock 3D models.

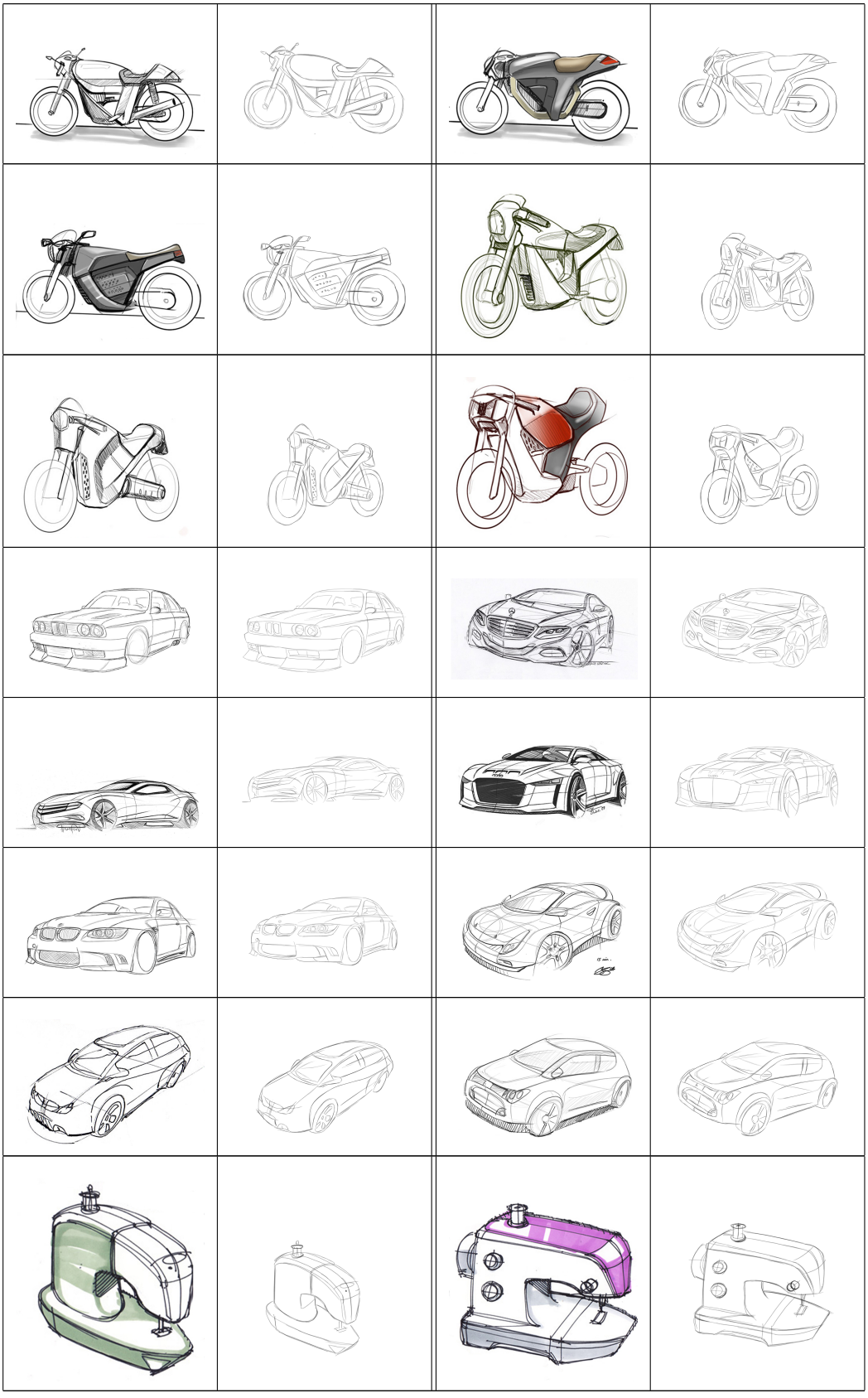


Figure A.3: Original sketches with their redrawn versions.

Bibliography

[AD96] H. Araujo and J.M. Dias. "An introduction to the log-polar mapping [image sampling]". In: *Cybernetic Vision, 1996. Proceedings., Second Workshop on*. Dec. 1996, pp. 139–144. DOI: 10.1109/CYBVIS.1996.629454.

[Bar07] Guido Bartoli. "Image Registration Techniques: A Comprehensive Survey". In: (2007).

[BBS08] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. "ILoveSketch: As-natural-as-possible Sketching System for Creating 3D Curve Models". In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. UIST '08. Monterey, CA, USA: ACM, 2008, pp. 151–160. ISBN: 978-1-59593-975-3. DOI: 10.1145/1449715.1449740. URL: http://doi.acm.org/10.1145/1449715.1449740.

[BC13] Alexandra Bonnici and Kenneth Camilleri. "A Circle-based Vectorization Algorithm for Drawings with Shadows". In: *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*. SBIM '13. Anaheim, California: ACM, 2013, pp. 69–77. ISBN: 978-1-4503-2205-8. DOI: 10.1145/2487381.2487386. URL: http://doi.acm.org/10.1145/2487381.2487386.

[BD03] Zafer Bilda and Halime Demirkan. "An insight on designers' sketching activities in traditional versus digital media". In: *Design Studies* 24.1 (2003), pp. 27 –50. ISSN: 0142-694X. DOI: http://dx.doi.org/10.1016/S0142-694X(02)00032-7. URL: http://www.sciencedirect.com/science/article/pii/S0142694X02000327.

[BMP02] S. Belongie, J. Malik, and J. Puzicha. "Shape matching and object recognition using shape contexts". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.4 (Apr. 2002), pp. 509–522. ISSN: 0162-8828. DOI: 10.1109/34.993558.

[BW05] Gary Robert Bertoline and Eric N Wiebe. *Fundamentals of Graphics Communication (McGraw-Hill Graphics)*. McGraw-Hill Science/Engineering/Math, 2005. ISBN: 0073136069.

[Can86] J Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (June 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: http://dx.doi.org/10.1109/TPAMI.1986.4767851.

[Che87] L. P. Chew. "Constrained Delaunay Triangulations". In: *Proceedings of the Third Annual Symposium on Computational Geometry*. SCG '87. Waterloo, Ontario, Canada: ACM, 1987, pp. 215–222. ISBN: 0-89791-231-4. DOI: 10.1145/41958.41981. URL: http://doi.acm.org/10.1145/41958.41981.

[CLL10] Minsu Cho, Jungmin Lee, and KyoungMu Lee. "Reweighted Random Walks for Graph Matching". English. In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Vol. 6315. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 492–505. ISBN: 978-3-642-15554-3. DOI: 10.1007/978-3-642-15555-0_36. URL: http://dx.doi.org/10.1007/978-3-642-15555-0_36.

[CSC14] Che-Han Chang, Y. Sato, and Yung-Yu Chuang. "Shape-Preserving Half-Projective Warps for Image Stitching". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. June 2014, pp. 3254–3261. DOI: 10.1109/CVPR.2014.422.

[CSD11] Gaurav Chaurasia, Olga Sorkine, and George Drettakis. "Silhouette-aware Warping for Image-based Rendering". In: *Proceedings of the Twenty-second Eurographics Conference on Rendering*. EGSR '11. Prague, Czech Republic: Eurographics Association, 2011, pp. 1223–1232. DOI: 10.1111/j.1467-8659.2011.01981.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2011.01981.x.

[CW93] Shenchang Eric Chen and Lance Williams. "View Interpolation for Image Synthesis". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 279–288. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166153. URL: http://doi.acm.org/10.1145/166117.166153.

[D.85] Sturman D. "Interactive keyframe animation of 3D articulated models". In: *Course notes, SIGGRAPH Course Number 10, Computer Animation: 3D Motion Specification and Control* (July 1985), pp. 17–25.

[DB06] M. Donoser and H. Bischof. "Efficient Maximally Stable Extremal Region (MSER) Tracking". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. June 2006, pp. 553–560. DOI: 10.1109/CVPR.2006.107.

[Duc77] Jean Duchon. "Splines minimizing rotation-invariant semi-norms in Sobolev spaces". English. In: *Constructive Theory of Functions of Several Variables*. Ed. by Walter Schempp and Karl Zeller. Vol. 571. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1977, pp. 85–100. ISBN: 978-3-540-08069-5. DOI: 10.1007/BFb0086566. URL: http://dx.doi.org/10.1007/BFb0086566.

[DZ13] Piotr Dollár and C. Lawrence Zitnick. "Structured Forests for Fast Edge Detection". In: *Proceedings of the 2013 IEEE International Conference on Computer Vision*. ICCV '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1841–1848. ISBN: 978-1-4799-2840-8. DOI: 10.1109/ICCV.2013.231. URL: http://dx.doi.org/10.1109/ICCV.2013.231.

[ES11] K. Eissen and R. Steur. *Sketching: The Basics*. BIS, 2011. ISBN: 9789063692537. URL: https://books.google.co.in/books?id=HoSlcQAACAAJ.

[GR96] Steven Gold and Anand Rangarajan. "A Graduated Assignment Algorithm for Graph Matching". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 18.4 (Apr. 1996), pp. 377–388. ISSN: 0162-8828. DOI: 10.1109/34.491619. URL: http://dx.doi.org/10.1109/34.491619.

[Hla14] George Hlavács. *The Exceptionally Simple Theory of Sketching: Easy to Follow Tips and Tricks to Make your Sketches Look Beautiful Paperback – June 3, 2014*. BIS Publishers, 2014. ISBN: 9063693346.

[Lee12] Kyoung Mu Lee. "Progressive Graph Matching: Making a Move of Graphs via Probabilistic Voting". In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 398–405. ISBN: 978-1-4673-1226-4. URL: http://dl.acm.org/citation.cfm?id=2354409.2354847.

[LH96] Marc Levoy and Pat Hanrahan. "Light Field Rendering". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 31–42. ISBN: 0-89791-746-4. DOI: 10.1145/237170.237199. URL: http://doi.acm.org/10.1145/237170.237199.

[Lia+14] Jing Liao et al. "Automating Image Morphing Using Structural Similarity on a Halfway Domain". In: *ACM Trans. Graph.* 33.5 (Sept. 2014), 168:1–168:12. ISSN: 0730-0301. DOI: 10.1145/2629494. URL: http://doi.acm.org/10.1145/2629494.

[Liu+09] Feng Liu et al. "Content-preserving Warps for 3D Video Stabilization". In: *ACM Trans. Graph.* 28.3 (July 2009), 44:1–44:9. ISSN: 0730-0301. DOI: 10.1145/1531326.1531350. URL: http://doi.acm.org/10.1145/1531326.1531350.

[Low04] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60 (2004), pp. 91–110.

[Mah+09] Dhruv Mahajan et al. "Moving Gradients: A Path-based Method for Plausible Image Interpolation". In: *ACM Trans. Graph.* 28.3 (July 2009), 42:1–42:11. ISSN: 0730-0301. DOI: 10.1145/1531326.1531348. URL: http://doi.acm.org/10.1145/1531326.1531348.

[MB95] Leonard McMillan and Gary Bishop. "Plenoptic Modeling: An Image-based Rendering System". In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 39–46. ISBN: 0-89791-701-4. DOI: 10.1145/218380.218398. URL: http://doi.acm.org/10.1145/218380.218398.

[Mun57] James Munkres. *ALGORITHMS FOR THE ASSIGNMENT AND TRANSPORTATION PROBLEMS*. 1957.

[Nea+07] Andrew Nealen et al. “FiberMesh: Designing Freeform Surfaces with 3D Curves”. In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301. DOI: 10.1145/1276377.1276429. URL: http://doi.acm.org/10.1145/1276377.1276429.

[OK11] Gunay Orbay and Levent Burak Kara. “Beautification of Design Sketches Using Trainable Stroke Clustering and Curve Fitting”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.5 (2011), pp. 694–708. ISSN: 1077-2626. DOI: http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.105.

[Ols+09] Luke Olsen et al. “Technical Section: Sketch-based Modeling: A Survey”. In: *Comput. Graph.* 33.1 (Feb. 2009), pp. 85–103. ISSN: 0097-8493. DOI: 10.1016/j.cag.2008.09.013. URL: http://dx.doi.org/10.1016/j.cag.2008.09.013.

[RID10] Alec Rivers, Takeo Igarashi, and Frédo Durand. “2.5D Cartoon Models”. In: *ACM Trans. Graph.* 29.4 (July 2010), 59:1–59:7. ISSN: 0730-0301. DOI: 10.1145/1778765.1778796. URL: http://doi.acm.org/10.1145/1778765.1778796.

[SCL12] Yumin Suh, Minsu Cho, and Kyoung Mu Lee. “Graph Matching via Sequential Monte Carlo”. In: *Proceedings of the 12th European Conference on Computer Vision - Volume Part III*. ECCV’12. Florence, Italy: Springer-Verlag, 2012, pp. 624–637. ISBN: 978-3-642-33711-6. DOI: 10.1007/978-3-642-33712-3_45. URL: http://dx.doi.org/10.1007/978-3-642-33712-3_45.

[SD96] Steven M. Seitz and Charles R. Dyer. “View Morphing”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 21–30. ISBN: 0-89791-746-4. DOI: 10.1145/237170.237196. URL: http://doi.acm.org/10.1145/237170.237196.

[Sha+12] Cloud Shao et al. “CrossShade: Shading Concept Sketches Using Cross-Section Curves”. In: *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)* 31.4 (2012). URL: http://www.crossshade.com.

[Sha+13] Tianjia Shao et al. “Interpreting Concept Sketches”. In: *ACM Trans. Graph.* 32.4 (July 2013), 56:1–56:10. ISSN: 0730-0301. DOI: 10.1145/2461912.2462003. URL: http://doi.acm.org/10.1145/2461912.2462003.

[Shr+11] Abhinav Shrivastava et al. “Data-driven Visual Similarity for Cross-domain Image Matching”. In: *ACM Trans. Graph.* 30.6 (Dec. 2011), 154:1–154:10. ISSN: 0730-0301. DOI: 10.1145/2070781.2024188. URL: http://doi.acm.org/10.1145/2070781.2024188.

[SK00] Harry Shum and Sing B Kang. “Review of image-based rendering techniques”. In: *Visual Communications and Image Processing 2000*. International Society for Optics and Photonics. 2000, pp. 2–13.

[Sok] Karina Sokolava. *Basic Guidelines to Product Sketching*. URL: http://www.hongkiat.com/blog/basic-guidelines-to-product-sketching/ (visited on 06/30/2015).

[TKR13] L. Torresani, V. Kolmogorov, and C. Rother. "A Dual Decomposition Approach to Feature Correspondence". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.2 (Feb. 2013), pp. 259–271. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2012.105.

[TN10] Yuandong Tian and S.G. Narasimhan. "A globally optimal data-driven approach for image distortion estimation". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. June 2010, pp. 1277–1284. DOI: 10.1109/CVPR.2010.5539822.

[TZY08] Zhuowen Tu, Songfeng Zheng, and Alan Yuille. "Shape matching and registration by data-driven EM". In: *Computer Vision and Image Understanding* 109.3 (2008), pp. 290–304.

[Xu+14] Baoxuan Xu et al. "True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization". In: *Transactions on Graphics (Proc. SIGGRAPH 2014)* 33.4 (2014). DOI: 2601097.2601128.

[Zha+09] Guo-Xin Zhang et al. "A Shape-Preserving Approach to Image Resizing". In: *Computer Graphics Forum* 28.7 (2009), pp. 1897–1906.

[Zha+14] Qi Zhang et al. "Rolling Guidance Filter". English. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Vol. 8691. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 815–830. ISBN: 978-3-319-10577-2. DOI: 10.1007/978-3-319-10578-9_53. URL: http://dx.doi.org/10.1007/978-3-319-10578-9_53.

[The] The Mathworks, Inc. *Documentation: Solve systems of linear equations $Ax = B$ for x*. URL: http://in.mathworks.com/help/matlab/ref/mldivide.html (visited on 06/30/2015).

[Tri] Trimble Navigation Limited. *3D WareHouse*. URL: https://3dwarehouse.sketchup.com/?hl=en (visited on 07/03/2015).