Towards object-based image editing

Zhou, Hailing

2012

Zhou, H. (2012). Towards object-based image editing. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/50838>

<https://doi.org/10.32657/10356/50838>

NANYANG TECHNOLOGICAL UNIVERSITY

TOWARDS OBJECT-BASED IMAGE EDITING

HAILING ZHOU

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2012

Abstract

With the increasing use of images in web design, document processing, entertainment, medical analysis, virtual environment creation, etc., the demand is dramatically growing for effective editing techniques that can fast and accurately create, compose, render and manipulate image contents. In recent years, plenty of research has been conducted for these tasks. However, the current techniques for these tasks are still far away from being satisfactory. It usually needs extensive user guidance, with painstaking time and effort, to produce a desired result. Our research thus investigates new techniques and tools for effective creation, extraction, composition, and other manipulations of image contents.

We introduce an object oriented and vector based image editing framework. With the framework, we perceive an image as a set of objects represented by vector graphics so that image editing can be performed easily and semantically. The framework consists of four technical components: image segmentation, shape completion, image completion and image vectorization. The first three components are used to decompose an image into meaningful objects to support object-level editing. The last component is used to convert a raster image or an object of the image into a vector graphics representation to facilitate editing process. We have developed new algorithms for these components to implement the proposed framework.

In particular, to overcome the lack of effectiveness and accuracy in extracting objects from complex backgrounds (especially with textures or low contrasts), we propose to incorporate image texture information into energy functions in graph-based image segmentation, use structure tensors to modify the weight between two nodes in the image graph, adopt a constrained active contour model to handle region and boundary simultaneously, and design a “soft” brush interface for users to locally adjust segmentation results. The experiments show that our proposed method is robust to user inputs and is

able to produce more accurate and smoother boundaries when evaluated on the MSRC image segmentation benchmark.

To generate visually pleasing shape completion, we introduce a special arc spline called *Euler arc spline*, which can be considered as an extension of an Euler curve in the sense that the points in the Euler curve are replaced by arcs. Euler arc splines have nice properties desired by aesthetics of curves, in addition to computational simplicity and NURBS representation. We also develop a robust algorithm to construct such arc splines for shape completion. The development of the algorithm involves two optimization processes, which are converted into a single minimization problem in two variables solved by the Levenberg-Marquardt algorithm. Compared with previous methods, the proposed algorithm always guarantees the interpolation of two boundary conditions.

In patch-based image completion, we notice that the patch size affects how well the filled patch captures the local characteristics of known regions, affects the measurement accuracy of how similar two patches are and thus affects the final completion quality. Therefore we propose an algorithm to adaptively determine the patch size in the process of image completion and also a new similarity measurement for two patches. As a result, our image completion method can propagate colors and features plausibly.

Finally, we present a subdivision-based vector graphics representation and its vectorization algorithm for raster images or objects in the images. The representation is in a form of triangular mesh augmented with color attribute at vertices and feature attribute at edges. The Loop subdivision scheme is modified and applied to the mesh to define piecewise-smoothly varying images. We also develop an algorithm to automatically generate such a vector graphics representation from an input image or an object. Compared with existing image vectorization methods, the proposed representation and algorithm have the following advantages in addition to the common merits of vector representations (such as editability and scalability): (1) they support more flexible mesh topology and thus handle images or objects with complicated boundaries or features more effectively; and (2) they are able to faithfully reconstruct curvilinear features, especially with subtle shading effects.

Acknowledgments

I would like to thank Dr. Jianmin Zheng, my supervisor, for his time, effort and constant support during this research. His great knowledge and serious attitude to research work benefit me well, in particular pushing me to understand problems deeply and write high quality papers. Thank him very much for his instruction and patience.

I also want to thank Dr. Xunnian Yang and Dr. Xin Li, for their instructions and discussions in my research.

Thank all my friends and colleagues: Chang Liu, Wenxian Yang, Zhixiang Ren, Xiaoqun Wu, Yusha Li and so on. They accompany me to dine and play and bringing me happiness during the four-year PhD life.

I want to give deep thanks to my family. In particular, thank my elder sister for her encouragement and listening; and thank my husband Dr. Lei Wei for his support and love.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vii
List of Tables	xii
1 Introduction	1
1.1 Background	1
1.2 Problem Statement and Objectives	3
1.3 Contributions and Thesis Organization	4
2 Literature Review	8
2.1 Image Editing	8
2.1.1 Pixel based editing	8
2.1.2 Object based editing	9
2.2 Image Decomposition	11
2.2.1 Image segmentation	11
2.2.2 Techniques of completing shape and colors	14
2.3 Image Vectorization	20
3 Object-Based Image Editing Framework	23
4 Image Segmentation	34
4.1 Motivation and Our Work	34
4.2 Texture Aware Graph Cut Segmentation	36
4.2.1 Graph cut segmentation	36
4.2.2 Combining color and texture	38

4.2.3	Structure tensor	41
4.3	Convex Active Contour and Local Editing	43
4.3.1	Convex active contour	44
4.3.2	Local boundary editing	46
4.4	Experiments	49
4.4.1	Effects of individual components	49
4.4.2	Overall performance	52
4.4.3	Quantitative evaluation on the benchmark data set	55
5	Euler Arc Splines for Shape Completion	62
5.1	Motivation and Our Work	62
5.2	Arc Splines and Euler Arc Spline	64
5.2.1	G^1 continuous arc splines	64
5.2.2	Euler arc splines	68
5.3	Shape Completion Algorithm	72
5.3.1	Perturbation for interpolation	73
5.3.2	Optimal arc-length and initial curvature	75
5.3.3	Algorithm	78
5.4	Experiments and Discussions	79
6	Patch-Based Image Completion	84
6.1	Motivation and Our Work	84
6.2	Overview of Patch-based Image Completion	87
6.3	Improvements of the Completion Algorithm	88
6.3.1	Adaptive patch size	89
6.3.2	Similarity measurement	91
6.4	Experimental Results	93
7	Curvilinear Feature Driven Image Vectorization Using Subdivision-Based Representation	97
7.1	Motivation and Our Work	97
7.2	Subdivision-Based Image Representation	99

7.2.1	Curvilinear feature representation	100
7.2.2	Subdivision surfaces	102
7.2.3	Modeling color discontinuity	105
7.3	Image Vectorization Algorithm	109
7.3.1	Feature extraction	109
7.3.2	Initial mesh construction	110
7.3.3	Color optimization	113
7.3.4	Mesh refinement	114
7.4	Experimental Results and Discussions	116
7.4.1	Quality assessment	121
7.4.2	Applications	121
8	Conclusions	126
8.1	Summary	126
8.2	Future Work	128
	References	131
A	Author Publication List	142

List of Figures

2.1	Visual completion examples. (a): Object shape is partially missed due to occlusion; (c) illusory contours; (b) and (d) are the results of shape completion using our method.	15
2.2	Patch-based image completion. The target region is region T shown in red color and surrounded by a blur color boundary ∂T and patches are the rectangles centered at yellow color pixels.	19
3.1	Object segmentation and organization. The red and blue strokes in the top figures are user inputs to indicate what they want or not.	24
3.2	Fill holes on the background layer. (a) and (b) are the results with and without holes on the background layer.	25
3.3	Holes on the object layers (a) make the editing of object arrangements implausible (b).	26
3.4	From left to right: complete the shape of an object with the pink curve; a hole on the object is formed shown by red colors; fill colors in the hole; rearrange objects for editing. The red and blue point-orientation pairs in the first column are user inputs for specifying boundary conditions. . . .	27
3.5	The output of the framework.	28
3.6	Image scaling. (a): original image; (b) reconstructed image from our framework; (c) magnification ($\times 2$) of the enclosed area in (b); (d) magnification (original image $\times 8$) of the enclosed area in (c); (e) magnification ($\times 8$) using bicubic interpolation.	29

3.7	Sharpen and blur image edge features. (a) original image (left) with blur edge features and our reconstructed image (right) sharpening the original image; (b) original image (left) and feature editing (right) with sharpness (top) and local blur (bottom).	30
3.8	Image warping. (a) original raster image (left), intermediate (mid) and final editing results (right); (b) mesh representation and operations . . .	31
3.9	Image warping and composition. Input raster images and edited results are shown in the left and right respectively.	32
3.10	The pipeline in the framework.	33
4.1	Three images (top) and the segmentation results (bottom) generated by GrabCut [82].	35
4.2	A color image (left) and its texture descriptor (right).	40
4.3	The construction of the augmented image from an input color image. . .	41
4.4	Comparison of segmentation using different methods for β . Left: an input image with foreground (red) and background (blue) seeds; Middle: the result using a fixed β ; Right: the result using adaptive β	42
4.5	An image (left) and its structure tensor visualized using ellipses (right). .	42
4.6	Local boundary editing. (a) an input image with a problematic boundary C_0 (yellow) and a soft constraint Ω_0 (green); (b) two endpoints A and B ; (c) the inverted distance map I_{idm} and the shortest path P_{AB} ; (d) the problematic contour segment C_{AB}^2 is replaced by P_{AB} ; (e) the path P_{AB} is extended to $P_{A'B'}$, and $P_{A'B'}$ is to be deformed for a global smoothness; (f) the final optimized contour C_f	47
4.7	The setting for the active contour method to optimize the path in local boundary editing.	48
4.8	Segmentation results from different combinations of color and texture, where n stands for the size of pixel-groups and α stands for the combination coefficients. $\alpha = 0.2$ for the top row and $\alpha = 0.8$ for the bottom row are obtained from the KL distances.	50
4.9	Segmentation results with (b) and without(c) the active contour model. .	51

4.10	Effects of different combinations for h_r . (a) The image of h_r for $\omega_1 = 0$; (b) The image of h_r for $\omega_1 = 1$; (c-e) The segmentation results correspond to $\omega_1 = 0, 0.5$, and 1 , respectively.	51
4.11	Comparing the segmentation results without (middle) and with (right) the structure tensor.	52
4.12	Soft boundary constraints improve segmentations of an object with low contrast boundary. (a) foreground/background input1, (b) output1, (c) foreground/background input2, (d) output2, (e) with an additional soft boundary constraint (in green), (f) output3, (g) the result given in [104].	53
4.13	Segmentation on images with different features using our proposed method. Top row: synthetic textured images; Middle row: real-world images with textures; Bottom row: colored images. The parameters n and α are the size of pixel-groups and the KL distance in (Eq. 4.7).	54
4.14	Segmentation results of GrabCut [82].	54
4.15	The segmentation results using the proposed method but without incorporating the active contour model.	55
4.16	The segmentation results with different user inputs.	55
4.17	Comparison of segmentation results in terms of smoothness.	56
4.18	Four technique components.	56
4.19	The segmentation results on the MSRC data set using our method. The ε given in parenthesis is the error rate.	59
4.20	The continued part of Figure 4.19.	60
4.21	The statistics of processing times.	61
5.1	A G^1 continuous arc spline curve	65
5.2	One arc segment in an arc spline curve	66
5.3	One arc segment and its three Bézier control points	67
5.4	Four types of shape of an Euler arc spline	70
5.5	Shape completion with an EAS curve	72
5.6	Examples of Euler curves in pink color and a series of Euler arc splines shown in gray whose numbers of arcs are 10, 20, 30 and 40.	79

5.7	The shape completion for various point-orientation configurations.	80
5.8	Comparison of our method and Kimia <i>et al.</i> 's method. The curves generated by our method is shown in pink color and the curves generated by Kimia <i>et al.</i> 's method is shown in black.	81
5.9	Shape completion for the same tangent vectors but different angle values.	82
5.10	Shape completion with our proposed curves (pink) and the Hermite curves (green). The scale of the Hermite splines is determined manually.	82
5.11	Curvature illustrations of the EAS curves (pink) and the Hermite curves (green).	83
5.12	Shape completion using Euler arc spline curves.	83
6.1	Different patch sizes for completion	85
6.2	Different similarity measurement for completion.	86
6.3	Gradient angle histogram of an image.	90
6.4	Adaptive patch size: (a) an input image; (b) size map using Drori et al's method; (c) size map using our method; (d) a few patches based on our size map	92
6.5	Similarity measurement between target and source patches.	92
6.6	Image completion with different similarity measurements.	93
6.7	Completion results for 15 images using a fixed patch size and adaptive patch sizes	94
6.8	More examples for comparison.	95
6.9	Comparisons between image completion without and with shape completion.	96
7.1	A restricted B-spline curve corresponds to an initial polyline (in green) and a refined polyline (in blue).	102
7.2	Edge subdivision rules. Red lines denote crease edges and the disk denotes a corner vertex.	104
7.3	Vertex subdivision rules. Red lines denote crease edges and disks denote corner vertices.	104
7.4	Limit position masks for vertices. Red lines denote crease edges and disks denote corner vertices.	106

7.5	Color discontinuity modeling. Top: a sharp feature is created with sharpness = 0.9; Bottom: a semi-sharp feature is created with sharpness = 0.1.	107
7.6	Images with curvilinear edges and their meshes.	108
7.7	Feature extraction. (a) Input object; (b) Cleaned sharp (red) and semi-sharp (green) edges; (c) Some corners.	110
7.8	The second feature polyline (in pink) is created from the first one (in red).	112
7.9	Effects of image reconstruction using only single polylines marked in red in (d) and using feature polyline pairs marked in green in (e).	112
7.10	Adaptive sampling. (a): an error map; (b)-(d): the three quantified sub-maps; (e): the Gaussian pyramid using a Gaussian filter one, two, and four times with sigma equal to 2.0 from bottom to top; (f)-(h): interim sampling results with sizes of 2, 4 and 16 pixels respectively; (i): the final sampling result of (a).	115
7.11	An example of mesh refinement.	115
7.12	Vectorization results using our method. Left: input images; Middle: reconstructed meshes; Right: reconstructed images.	117
7.13	More vectorization results: original images (left) vs reconstructed results (right).	118
7.14	More vectorization results (cont).	119
7.15	Vectorization with/without use of sharpness.	120
7.16	Left: result of [57]; Right: ours.	122
7.17	Left: result of [102]; Right: ours.	122
7.18	Editing of image features.	123
7.19	Image creation from sketches: (a). The user sketches curvilinear edges (sharp and semi-sharp features are marked in red and green); (b) A tagged control mesh is automatically created; (c) The user paints color to vertices; (d) An image is created using the proposed subdivision surface.	125

List of Tables

4.1	Error rates of our method and its variants	57
4.2	Error rate comparison of our method and other methods	58
5.1	Statistics for approximate errors.	79
7.1	Image vectorization results	123

Chapter 1

Introduction

1.1 Background

With the increasing use of images in web design, document processing, entertainment, medical analysis, virtual environment creation, etc., the demand is dramatically growing for effective editing techniques that can fast and accurately create, compose, render and manipulate image contents [79, 7]. In recent years, a lot of research has been conducted for these tasks. However, the current techniques for these tasks are still far away from satisfactory. To produce a desired result, they usually need extensive user guidance, with painstaking time and effort. For this reason, research continues to look for better techniques that can reduce human burdens in editing image contents.

Due to the fact that hard devices acquire and display images in a discrete way, raster data had become a popular format to represent an image. Therefore the traditional image editing techniques including those used in Adobe Photoshop or GIMP rely on manipulating images as arrays of pixels. However, this representation has poor perception of image contents. When we look at an image, we do not see the grid of pixels. Instead, we see a bunch of objects and a background. When we want to move or change an object in an image, we do not think about how to change the pixels. We think about how to select the object and then how to change its location, shape or attributes. Without any

understanding of an image in an array of pixels, the traditional techniques are not easy to edit image contents freely.

On the other hand, vision techniques inspire and make it possible to understand image contents. Computer vision studies how to detect edges, regions, shape features, lighting properties and even 3D geometry. Thus vision techniques can help a computer “understand” what the user wants based on a minimum of input or “see” visual and perceptual data, which the computer is usually weaker to process than human. These techniques combining computer and user visions enable a user to process an image in a higher level. Some techniques and tools have been developed in this way for image editing including Adobe’s Magic Wand and Intelligent Scissors [71, 70]. Further extensions of these techniques are still needed for improving accuracy and reducing user burdens.

Meanwhile, graphics techniques accelerate the possibility of free editing image contents. For example, the graphics based modeling, deformation, and rendering techniques make it simple for users to create, warp, and render images or objects. In addition, since the manipulation on those geometrical data (i.e. vector graphics) has physical senses, it is more nature to edit an image in the vector-based way and in terms of objects. If we represent an image with vector graphics, users can perform image editing more effectively and intuitively. Some works and tools have been developed using this approach such as Adobe illustrator and methods in [34, 93, 57], in which a key problem is how to represent an image concisely and semantically. Research to find techniques solving this problem becomes appealing recently.

The convergence of vision and graphics provides a good opportunity to develop techniques that allow user to perform image editing based on contents and vector graphics. Some related works such as [7, 34, 93, 57, 79] have been developed recently, where image objects are first extracted and represented by meshes or patches. These techniques allow users to perform operations of selecting, scaling, warping, copying, pasting, deleting or

relocating objects in a higher level. Nevertheless, although existing works [7, 79] greatly reduce human burdens, for the quality and human efforts on editing, there are still a large room for further investigations.

1.2 Problem Statement and Objectives

This project researches fundamental image representations and algorithms for processing image contents. It aims to develop techniques and tools for image editing that enable users to effectively and efficiently create, manipulate, and render images. To this end, object-oriented and vector-based techniques are sought. Our basic idea is to borrow and combine various techniques from computer graphics, image processing and computer vision to maximize the use of the characteristics of input raster data to reduce human burden in image editing. Thus, given a raster image, we firstly understand image contents (i.e. objects), and then represent each object in the image using vector data such as mesh with color or intensity attributes. Two main research problems are involved in the idea: *image decomposition* and *vectorization*. As a result, a new image representation is formed by organizing all the vectorized components. An image based on this representation has semantical meanings and is easy to edit. The desired image editing can be achieved on this represented image. The objectives of our work are:

1. to design a flexible image editing framework supporting advanced image editing including manipulations of shape, size, location, color, and membership of image contents with fewer user interactions of assisting to accomplish these tasks;
2. to explore techniques to understand image contents comprehensively through segmenting objects accurately, and estimating occluded regions plausibly;
3. to develop a compact and precise vector graphics representation for images and develop an efficient and fully automatic algorithm for converting raster images into vector graphics.

1.3 Contributions and Thesis Organization

This thesis has made several contributions to the research in editing of image contents. These contributions also promote the research on fundamental image representation and algorithms.

To easily and semantically edit an image, we introduce an object oriented and vector based image editing framework, which is more comprehensive than the most relevant works [70, 34, 7, 79]. After careful analysis, the framework is designed to consist of two modules: image decomposition and image vectorization. The decomposition module involves an object extraction procedure as well as a hole filling procedure to ensure that object-based editing can be performed seamlessly. For object extraction, users only need to simply scratch some strokes in the image to guide segmentation and the objects will then be selected one by one. For hole filling, traditional image completion or inpainting techniques are not competent for filling all these holes plausibly, especially for those covering geometrically continuous boundary of objects. Therefore we propose that complete shapes or boundaries of objects first and estimate the color information of holes thereafter. The vectorization module involves converting either a rectangular background or objects with arbitrary topology and shapes into vector graphics representation. With the framework, an input raster image will be converted into another representation which consists of a set of vector objects and a vector background. Users are then able to perform image editing as conveniently as perform document processing using the MS Office Word with words as objects. The common and basic operations: copying, pasting, deleting, relocation, resizing, and recoloring could be achieved by simple mouse gestures.

We also develop four technical components—image segmentation, shape completion, image completion and image vectorization—to support the implementation of the proposed framework.

- Most existing interactive image segmentation techniques have difficulty in segmenting objects from complex background with textures or low contrasts. We improve them by extending graph cut based methods. First, we take textures into account in the segmentation process by creating an augmented image with both color and texture information and then applying graph cut on the augmented image. Second, we construct geometric structure tensors to describe local color (intensity) variations of the image and incorporate them into the graph cut model. Third, the continuous-domain convex active contour model is adapted and incorporated as a postprocess for graph cut in order to improve the segmentation. Fourth, we propose to include soft constraints into the segmentation process, which allow the user to scratch to indicate the region that the boundary should pass through. These techniques and their combinations in deed are applicable in any graph based segmentation method such as graph cut, random walk or geodesic methods [11, 61, 82, 36, 104, 107, 5, 24] to improve their performance.
- To fill a geometrically continuous boundary of an object interrupted by occlusion is known as the shape completion or curve completion. We introduce a special arc spline called an Euler arc spline for shape completion. Euler arc spline is considered as an extension of an Euler curve and has nice properties desired by aesthetics of curves. We then develop a robust algorithm for shape completion using Euler arc splines, that is, given two specified endpoints with associated orientations, an optimal Euler arc spline curve is constructed to complete the gap.
- To fill holes with colors, we propose to use a patch based image completion technique. We improve the existing patch based image completion algorithm by adaptively selecting the patch size in the process of filling the hole, which can effectively remove the artifacts caused by completing the image with patches of one fixed size.

We formulate the problem of determining the patch size as an optimization problem that minimizes an objective function involving image gradients and distinct and homogeneous features. In addition, we define a new similarity measurement for two patches, which not only considers the color differences but also the feature consistency within a patch. Based on these two improvements, visually pleasing completion results can be achieved by propagating colors and features in a hole plausibly and consistently.

- Vector graphics is used to represent an image as it is often compact, inherently scalable and easy to edit. Another advantage of vector graphics that it has potential to model/fit/reconstruct a raster image with high quality. We propose a subdivision-based vector graphics representation. The proposed representation is in a form of triangular mesh augmented with color attributes at vertices and feature attribute at edges, where the Loop subdivision scheme is modified and applied to the mesh so as to define piecewise-smoothly varying images. An associated algorithm is developed to automatically convert a raster image or object into this representation. The algorithm is driven by curvilinear edges, and generates the representation through constructing a constrained triangulation and minimizes an energy function. Compared to existing image vectorization techniques, the proposed representation and algorithm have two distinct advantages over existing ones: (1) they allow more flexible mesh topology and thus handle images or objects with complicated boundaries or features more effectively; and (2) they are able to faithfully reconstruct curvilinear features, especially in modeling subtle shading effects around feature curves.

The remainder of the thesis is organized as follows:

- Chapter 2 reviews prior art on image editing and related techniques on the four supporting techniques: image segmentation, image completion, shape completion and image vectorization.
- Chapter 3 describes our framework for object-based image editing.
- Chapter 4 presents four techniques to enhance graph-based interactive image segmentation.
- Chapter 5 introduces Euler arc spline and develops a robust and effective shape completion algorithm using Euler arc spline.
- Chapter 6 presents a patch-based image completion algorithm which adaptively selects the patch size in the process of hole filling.
- Chapter 7 presents a subdivision based image representation and its associated vectorization algorithm.
- Chapter 8 summarizes the thesis and points out some directions for future work.

For goals itemized previously, Chapter 3 achieves the first one; Chapter 4-Chapter 6 solve the second one; the third goal is realized in Chapter 7.

Chapter 2

Literature Review

This chapter reviews prior work on image editing and relevant techniques such as image segmentation, shape completion, image completion and vectorization.

2.1 Image Editing

2.1.1 Pixel based editing

Pixel based editing alters an image at the pixel level. Photoshop, MS Paint, and GIMP, for example, revolve around editing pixels. They contain many pixel editing techniques such as resizing, painting, cloning, filtering, blurring and sharpening. With these techniques, some visually plausible results can be produced. However, good results are usually limited to tasks which simply require recoloring or moving pixels regardless of object membership.

However, for some applications such as special effect creation in movies or computer-assisted animation, they often require to create, compose and manipulate image contents. Unfortunately, pixel based editing techniques do not have any understanding of images. They are not sufficiently competent for users' demands. For example, the processes of selecting an object from an image, changing its shape or gesture and then composing it with a new image could not be easily and intuitively accomplished using pixel-based editing approaches.

2.1.2 Object based editing

2.1.2.1 Pixel based representation

Object based editing aims at editing an image at the object level. To perform this specific editing task, it is essential to extract objects from an image. Adobe's Magic Wand and Intelligent Scissors [70] are such tools for object selection. The magic wand tool starts with a small user-specified regions and then grows by connecting neighboring pixels with high similarity. The intelligent scissors requires users to place points along the desired contour of a foreground object and finds an optimal path between each neighboring point pairs.

However, to obtain a satisfactory object boundary, a large number of user interactions are needed in these techniques. Besides, these techniques maintain the pixel based representation for each object. Therefore some operations on objects such as scaling, warping, stretching, and bending are not easily performed. The applications with these techniques are limited to simple image composition.

2.1.2.2 Vector based representation

There are several works [34, 7, 79] aiming at object oriented and vector based image editing. In these works, objects are selected and then represented by vector graphics. Users are allowed to control object shape, size and placement using simple mouse gestures. Representing an object or image using vector graphics instead of pixels has at least two advantages: (1) it can avoid common problems associated with pixel-based image manipulation; (2) it is easy and natural for users to edit image based on geometrical vector data. The following three works are the most relevant to our research.

Froumentin *et al.* propose a method in [34] that decomposes an image into regions corresponding to manually specified color features and triangulates each region using a Delaunay triangulation enclosed by a NURBS boundary curve, forming a vector based

representation of the image. To warp an object in an image, the object is selected first by merging involved regions and then transformed using graphics techniques. In addition, holes/gaps on the background behind the object are estimated based on their surrounding colors and textures.

In [7], an object based image editing system is designed. They over-segment an image into a set of regions called TRAPs with borders aligning with edge features of objects or sub-objects. The objects are then selected by mouse click to collect their component regions. The selected objects are triangulated and the mesh are rendered by texture mapping triangles with corresponding region of the image. After that, users can edit the vectorized objects easily. In addition, the hole filling problem is solved by employing an inpainting technique to propagate TRAPs in the holes.

Price *et al.* [79] make great improvements/extensions on the work [7]. First, they use a graph cut based image segmentation method to extract objects, where users need to scratch some strokes to indicate what they want. This dramatically reduces human time and effort in selecting objects. Besides, they hierarchically organize objects so that it is fast to edit several objects simultaneously and independently. Second, they represent an object or an image using a regular Bézier mesh with color attributes, which is scalable and can represent raster data with high reconstruction quality. Third, for the hole filling problem, they propose to estimate colors of occluded mesh nodes using a Least Squares Fit.

It can be concluded that to achieve an object oriented and vector based image editing, three main problems are considered in these works: object selection, hole filling, and image/object vectorization. For each of them, there are large spaces for improvements. Recently, many advanced methods have been developed, which are reviewed in the following sections.

2.2 Image Decomposition

2.2.1 Image segmentation

Image segmentation refers to a process of dividing an image into several disjoint regions such that the pixels have homogeneous properties in each region and high differences between regions. Since the judgment of good or bad segmentation is subjective, incorporating human intentions can achieve more satisfactory segmentation results than the automatic image segmentation algorithms. Therefore the interactive image segmentation has become popular recently in pattern recognition, computer vision and image processing fields.

There are a large number of interactive image segmentation methods in literature. Early works utilize either boundary properties such as border tracing [91], active contours/Snake [51], and intelligent scissors [71] or regional properties such as blob coloring [6] and intelligent paint [81]. However, these methods require great cares to user interaction in order to obtain a satisfactory result.

Recently, graph based techniques that consider both boundary and regional properties have received considerable attention. Examples of such techniques include the graph cut based methods [11, 61, 82], the random walks based methods [36, 104, 107], and the geodesic methods [5, 24]. In these approaches, an image is modeled as a weighted graph, where each node represents a pixel, and the weight of each edge connecting two nodes is defined according to the color similarity between them. A segmentation is then produced by minimizing a certain energy function on that graph. Different energy functions present different behaviors of the corresponding methods [90]. In graph cut based algorithms, two virtual terminals are added in the graph to denote the foreground and background models. Each edge weight between a node and a terminal is defined as the possibility of the node fitting to foreground or background. After that, the segmentation is regarded as

a minimum cut problem in the weighted graph solved by a min-cut/max-flow algorithm. In random walk based methods, the probability of each unseeded pixel is computed in such a way that a random walker starting from the pixel first reaches the foreground or background seed regions, and then the pixels are grouped according to the maximal probability. The geodesic algorithms compute geodesic distances for each pixel to the foreground and background seed regions, and then classify the unlabeled pixels based on those distances. Since a graph cut based method has the ability to find globally optimal solutions for image segmentation, i.e., optimal segmentation boundaries and regions, it becomes more popular and has wider applications. Compared with the other two graph based methods, it can avoid segmentations with jaggy boundaries that do not snap to geometry features. Moreover, the use of the global color distribution model makes it relatively more robust.

Several graph cut based methods are reviewed briefly in the following, such as Interactive graph cut [11], LazySnapping [61], and GrabCut [82]. Interactive graph cut is for a general purpose interactive segmentation of monochrome N-dimensional images. The user specifies certain pixels as “foreground” or “background” seeds. They are modeled by histograms. Afterwards, the graph cut technique is used to find the globally optimal segmentation. LazySnapping builds a novel graph cut formation on a pre-computed image over-segmentation instead of image pixels to improve efficiency. It also improves the user interface (UI) of the graph cut method through designing a coarse-to-fine UI, which includes three tools: foreground or background strokes, overriding brush and direct vertex editing. For the overriding brush, a single stroke drawn by users is used as a “soft constraint” to guide the refinement of a problematic boundary, where much attention on smoothness and location of a stroke is required for the user drawing. The GrabCut method extends the graph cut framework [11] to color images and incomplete trimaps through developing an iterative version of the optimization. Besides, several types of user

inputs such as the incomplete trimaps, foreground or background strokes, and boundary matting brush are supported so that GrabCut provides a flexible control and editing of segmentation results for users.

However, these graph cut based methods have several inherent limitations: (1) they are sensitive to the number of seeds, pointed out in [90]; (2) due to the discrete graph based characteristic and the segmentation formulated as a binary hard labeling problem, the methods can not overcome the inherent grid bias (metrication errors); (3) since the graph cut tries to get a cut with the minimum total weights, the “short cut” problem is also inevitable. Although the boundary editing and matting tools can alleviate these problems, numerous and careful interactions are required to obtain a desired segmentation; (4) they have weak performances on segmentation of images with textures.

Meanwhile, the continuous domain convex active contour model [14] has been received much attention recently. It is one of most successful variational models in image segmentation. As an extension of the classical active contour model [51], not only boundary information [17], but also regional property [19] are considered to form a global convex model. An optimal solution can be found in the model through a fast numerical solver using a convex optimization [35]. The solution is a function defined in the image domain and can produce segmentation regions with smooth contours [14]. It is more accurate than the graph cut algorithm because it uses isotropic schemes to regularize the contour and it is also slightly faster. However, so far the convex active contour model is mainly utilized for unsupervised segmentation of gray images. In [73], a constrained active contour model is proposed, which incorporates the information from the user input and ensures the refined contour complying with the user input.

For all the above image segmentation methods, the partition of foreground and background regions is mainly guided by statistics involving image color or intensity values. However, the statistics on image values may not be enough to discriminate regions. In

such situations, texture information is more appropriate to serve as a discriminating feature. Several works attempt to combine both color (intensity) and texture features. A popular way to do such image segmentation is to integrate texture features into the graph cut framework [52, 64, 38], since an optimal solution of segmentation boundaries and regions can be achieved in the graph cut segmentation technique. The ways of combining texture and color features are one of the most important techniques in these works. In [52], a composite feature vector consisting of *RGB* color feature and texture feature derived from the Gabor filtered images is constructed, and an original input image is replaced by this vector-based image. In [64], the authors use an extended structure tensor that captures both texture information and intensity information as the graph cuts' data input. In the above two works, the color and texture features are combined together with the same weights to form an input feature. However, such equal mixture may cause the input feature to contain too much useless information leading to the negative side effect on segmentation. A method that can adaptively fuse the two features to improve the performance of segmentation is presented in [38]. They firstly extend the classical structure tensor by introducing a multiscale nonlinear structure tensor that describes texture features more effectively on texture orientation and scale. After that, an energy function, which fuses the two separate color and texture energy terms adaptively, is minimized followed the GrabCut framework. Superior performance of image segmentation is demonstrated, but the computational complexity and time are considerable on capturing texture properties. Besides, the inherent problems of the graph-cut based methods still exist, such as sensitive to the number of seeds, "short cut", and metrication errors.

2.2.2 Techniques of completing shape and colors

2.2.2.1 Shape completion

Segmentation process or some other processes may cause an object to miss some parts or to be separated into several disconnected components. This is especially true for

segmenting an occluded object. Human can complete broken shapes or curve segments smoothly with their visual system, referring to Figure 2.1 for example. Some researches have been conducted for achieving this function using computer, which are well known as shape completion, curve completion or gap completion techniques. The processing of shape completion is actually to find a curve to interpolate two specified endpoints with associated orientations, which we call *point-orientation pairs* in the thesis.

Figure 2.1: Visual completion examples. (a): Object shape is partially missed due to occlusion; (c) illusory contours; (b) and (d) are the results of shape completion using our method.

There exist many possible curves that meet the conditions of point-orientation pairs.

A straightforward approach to shape completion is to use cubic Hermite interpolation [32], since the given point-orientation pairs provide the first order geometric Hermite data. Hermit interpolation is simple to construct and compute, but it does not always provide satisfactory results as analyzed and shown in [39]. It is pointed out in [105] that one reason for Hermite interpolation to produce undesired shapes is unsuitable magnitudes of the given tangent vectors. Therefore Yong and Cheng present a new class of curves called optimized geometric Hermite curves for which the magnitudes of the endpoint tangent vectors in the Hermite interpolation process are optimized to make the strain energy of the curves be minimized [105].

In fact, the problem of shape completion is under-specified despite the appeal of our vision intuition for an optimal solution [53]. The solution really depends on the criteria regarding what constitutes the most “likely” or the most “pleasing” curve [53, 74]. Ullman suggests several criteria for completion curves [95]. That is, the curves should be invariant to rigid transformation, at least differentiable once, extensible, and minimize total curvature. Based on these criteria, he then proposes to use biarcs that minimize total square curvature as completion curves. However, it is later found that in many cases biarc completions have less pleasing appearance than the cubic polynomial completions [84] and Ullman’s biarc completion curve is generally not extensible [13]. Knuth considers the shape representation and construction of letters or symbols in typography from a collection of points [54], which is a problem similar to shape completion. He propose six criteria, somewhat similar to Ullman’s, which the most “pleasing” curve through a set specified points should satisfy. These criteria are known as similar transformation invariance, symmetry, extensibility, locality, smoothness, and roundedness [54, 53]. Since the last four criteria cannot be simultaneously satisfied, Knuth gives up the extensibility and roundedness properties but insists in the locality property, which leads to a cubic spline interpolation solution. This is the base of Knuth’s METAFONT system. However,

psychological studies have indicated that splines may not be a satisfactory primitive for shape completion [89].

A lot of work on aesthetic curve design proposes some energy functional and defines the curve as a solution to the problem of minimizing the energy functional. Elastica is one of such examples, in which the energy functional is the integral of a linear combination of the square curvature and the arc-length [45, 60, 67, 72]. Elastic is extensible, but not scale-invariant or rounded. On the other hand, it is argued that the energy functional capturing the elusive nature of the most “pleasing” curve should penalize curvature variation, but not necessarily curvature proper as in Elastica. In particular, minimizing the integral of the square of the derivative of curvature with respect to arc-length requires the curvature to be linear in arc-length, which leads to an Euler curve. Euler curves are also known as “Cornu spirals” or “Clothoid”. They have been used in various applications including highway and railroad design, computer aided design, and computer arts. 2D Euler curves have been generalized to 3D such that the curvature and torsion of the curve are linear with arc-length [40, 39] or to piecewise Euler spirals [96]. Since Euler curves are defined by complicated transcendental functions, some work devotes to approximating Euler curves by some simple representations such as Bézier curves and arc spline [69, 68, 98, 66, 86].

Kimia *et al.* [53] show that Euler curves are a suitable primitive for shape completion and various practical advantages of using Euler curve interpolation are examined. An algorithm for shape completion using Euler curves is described, which finds the completion curve by solving two non-linear equations in two unknowns involving Fresnel integrals. A biarc fitting is used for producing an initial guess and then the algorithm performs iteratively. Walton and Meek [97] improve the work by formulating the problem into two non-linear equations with only one unknown each, which gives a faster and more accurate algorithm.

2.2.2.2 Image completion

When an object is removed, a hole on the background will be left behind it; When an occluded object is selected, the occluded part will be exposed as a hole or gap. These are two example cases often happened in image editing. Those holes need to be filled in order to achieve seamless image editing.

Image completion is to address that problem by filling holes with visually plausible colors. Traditionally, image inpainting is used to restore scratches or stains in an image by propagating linear structures into the target region [18, 8] and texture synthesis is used to fill in regions with stationary or structured textures [29, 100, 28, 4]. Recently, Image completion combines these two techniques to propagate both structure and texture information to handle various filling problems such as filling removal regions, completing missing parts, and removing scratches from an image in a consistent way [26, 23].

Many image completion algorithms have been developed [26, 23, 94, 55, 48], among which patch-based approaches are popular ones due to their efficiency and effectiveness. The basic idea of patch-based approaches is to gradually completion the target region using patches from the source region. Refer to Figure 2.2 for an illustration. Let S and T denote source and target regions of an input image I . To fill a target patch tp centered at p , a search procedure is applied to find the source patch sp which has the best similarity with the target patch tp among all the patches of the same shape and size as tp in the source region. Then the source patch sp is composited into the target patch.

In particular, Drori *et al.* [26] propose a fragment based image completion algorithm which proceeds in a multi-scale fashion, iteratively approximates the target regions and composites adaptive image fragment (patch), which is selected from the most similar and frequent examples, into the image. The method works well in general. However, it is quite slow and sometimes introduces blur artifacts. Criminisi *et al.* [23] present an order-based contour evolution method that iteratively chooses the target patch with the

Figure 2.2: Patch-based image completion. The target region is region T shown in red color and surrounded by a blur color boundary ∂T and patches are the rectangles centered at yellow color pixels.

highest priority on the fill front (i.e., the boundary ∂T in Figure 2.2) and fills it using a source patch that best matches it. The “priority” is computed based on how much available neighbor information and outstanding structure at a target patch contains. This method is very fast and preserves better edge sharpness and continuity. While these two approaches basically belong to greedy filling, graph-based global optimal filling is also proposed for patch-based image completion. For graph-based global optimal filling, each patch is treated as a node and the completion process is to label each node on the target region from the source region to minimize reconstruction errors [55, 48]. These methods can give high-quality completion results, but the computation is extremely time-consuming. Sun *et al.* [94] present a patch-based completion algorithm which first propagates structure information based on optimal labeling and then fills the remaining unknown textures using an approach similar to Criminisi *et al.*’s. The algorithm needs user’s input to indicate some structure information.

It is known that the patch size has an important influence on the completion performance because it affects how well the filled patch captures the local characteristics of the source image. Note that in most of the existing patch-based image completion algorithms the patch size is fixed or specified by the user. However, it is difficult to choose a good patch size and it may not be appropriate to let all patches have the same size.

2.3 Image Vectorization

Image vectorization is the process of converting a raster image into a vector representation, which in general includes the technique of vectorizing an object in an image.

Various vectorization techniques have been proposed. Prior works focus on particular kinds of images [31, 109, 41, 108], such as black-white figures, line drawing, cartoon arts, or maps, where the vector graphics is limited to simple primitives (lines, curves, polygons) and regions in-between them are filled with uniform colors, or linear and radial gradients. Their corresponding algorithms are mainly to detect feature lines, trace contour and curve fitting. Besides, several softwares and commercial tools, for example VectorEye, VectorMagic, Autotrace, Corel Coreltrace, Adobe Livetrace, etc, have also developed techniques to provide such functionality. However, for realistic images with complex color distributions, to obtain convincing fidelities, all above techniques generates a prohibitive number of primitives, which make transmission and further editing challenging.

Recently, a lot of research works are interested in full-color photographic images, where the goal of a compact and accurate representation goes beyond the capabilities of those simple primitives. More complex representations and vectorization methods have been proposed. Roughly they can be categorized into two categories.

In the first category, an image is considered as a set of discrete pixels with color attributes. When the colors vary smoothly over an image, regular meshes such as a rectangular mesh are widely used for representing the image. For example, Price *et al.* propose a rectangular mesh consisting of cubic Bézier patches to represent an image [79]. Further refinement of the mesh is allowed to improve the reconstruction quality, but the local greedy error control may cause many tiny patches. A gradient mesh consisting of Ferguson patches is introduced by Sun *et.al* [93] and a semi-automatic technique is developed for creating an optimal mesh. A compact and accurate representation is obtained

with the method, but the user-assisted mesh placement takes much time and effort, especially for images with abrupt color changes. Lai *et al.* improves the work by developing an automatic technique to generate mesh lines for objects with arbitrary topology [57]. Since all involved procedures are linear, the algorithm is efficient and simple. However, all these methods may have difficulty in handling distinct image features due to the regular layout of a gradient mesh. A triangular mesh has flexible arrangement of triangles, which can address the problem. Su *et al.* [92] propose a pixel-level triangulation method, which connects all the pixels by order, forming a quadrilateral mesh, and then splits each quad into two triangles by inserting one of two diagonals. The determination of inserted diagonal is based on the local image features. Yu *et al.* [106] propose a data dependent triangulation, which iteratively swaps an input grid of triangles to locally minimize costs of involved triangle edges based on a look-ahead technique. The methods can reduce the visibility of artifacts and preserve edge features well. However, wide applications on image vectorization are not really their strength, since the large number of triangles make further editing formidable. An adaptive triangulations [25] are introduced to compress an image based on selecting important pixels, but the compression ratios are limited, in particular for an image with many significant curvilinear features or smooth color transitions. In general, traditional triangulation based representation has two disadvantages: (1) the linear function defined over a triangle make it have poor representation power, which tends to generate sharp color transitions on triangle edges; (2) the short line segments approximate curvilinear edge features, which requires a large number of lines to meet the continuity.

In the second category, an image is considered to be determined by a set of sharp edge features and smoothly varying colors in-between, where the vectorization involves image decomposition. Froumentin *et al.* give a vectorization method for realistic images in [34]. It decomposes an image into homogeneous regions corresponding to specified

color features, and then fits the contours and interior colors of the regions using NURBS curves and a constrained Delaunay triangulations (CDT). The curves associated with simple colors or triangulations inside form a scale independent representation of the whole image. Barrett *et al.* over-segment an image into a set of flat regions (e.g. TRAPs in [7]), and then represent an object or an image through using CDTs on TRAPs and texture mapping for color attributes. ARDECO detects smooth regions delimited by cubic spline curves, and approximates the interior colors using flat colors or a linear or circular gradient [58]. Although the prior extraction and approximation of edges produces a good reconstruction of features, the CDT-based color representations share the same shortcoming as a traditional triangulation. A novel vector graphics called diffusion curve is proposed in [77] to emphasize meaningful features. It is not mesh-based, but a geometric curve augmented with color and blur attributes. An image is represented by a set of these curves conforming to edges, and reconstructed by diffusing colors from them. However, it is possibly reconstruct an image with smooth colors inaccurately due to the lack of sufficient edge features. Moreover, the region-based color or shape editing is infeasible. Another important vectorization technique, less dependent on edge detection given in [102], is to decompose an image into a set of non-overlapping regions. Each region is parameterized as a triangular Bezier patch with curved boundaries, automatically aligning with image features. The inside colors are fitted by a smooth function. A non-linear optimization is formulated to guarantee the non-intersections of the curved boundaries.

Chapter 3

Object-Based Image Editing Framework

This chapter gives an overview of our framework for object-based image editing and briefly describes the technical components supporting this framework.

Image editing is a common task in many application fields. Different applications may have different requirements. For movies stunts or general image processing, operations such as copying, pasting, relocating, removing or warping objects are often used. Thereupon, the editing granularity from pixels or regions to meaningful objects becomes essential. On the web or mobile application where data amount needs to be controlled for transmission speed, a scalable and compact representation for images is highly desired. In animation and advertisement, image representation is expected to be scalable, easy to edit and stylize.

Considering these applications, we believe that it is essential to raise image editing granularity from the pixel level to the object level. For example, when users want to move an object, they do not think about how to change the pixels. What they consider is just how to select the object and then how to change its location. Therefore, image editing based on objects is more intuitive and meaningful. It is also more natural to the objects' topography, rather than operating in a way limited to a standard rectangular grid. On the other hand, considering the expectations of image representation in the

web, mobile, animation and advertisement applications, we need to find a more compact image representation for image editing instead of the traditional pixel representation. Pixel-based images are not scalable. In contrast, vector graphics has potential to satisfy this expectation. The graphics primitives such as lines, curves, polygons or mesh are defined geometrically or mathematically. They can offer a more compact representation, resolution independence and geometric editability. Vector based images are more easily animated and more readily stylized [77]. Meanwhile, geometrically-based editing has physical meanings and thus it is more natural or intuitive to users.

Figure 3.1: Object segmentation and organization. The red and blue strokes in the top figures are user inputs to indicate what they want or not.

For the above reasons, we introduce object oriented and vector based image editing as our framework. The framework starts from a raster image as input. It is an array of 2D pixels without embodying any semantics. Considering that our goal of editing images in the object level, we need to understand the image and then identify objects and the background. This requires an image decomposition procedure to decompose an image into meaningful components, which involves image segmentation, a fundamental task in computer vision and image processing. Image segmentation classifies pixels of an image into groups sharing certain visual characteristics. An interactive image segmentation method is considered here that allows a little user interaction to indicate user's intention

in the segmentation process. The involvement of user interaction effectively helps to achieve satisfactory segmentation results that are unattainable by the state-of-the-art of automatic image segmentation algorithms. After that, objects and the background are segmented and organized. An example is given in Figure 3.1, where an input raster image on the top-left corner is decomposed into six layers on the bottom row.

Figure 3.2: Fill holes on the background layer. (a) and (b) are the results with and without holes on the background layer.

Only the image segmentation procedure is not enough for our purpose. If users want to relocate, resize or warp the foreground object, a hole or seam will appear in the background. Refer to Figure 3.2 for illustration. To achieve a seamless image editing, the holes need to be filled. Image completion or inpainting techniques are therefore incorporated to handle the problem. It estimates the missed information in the hole regions according to some reasonable criteria (such as consistent features and colors across the gaps). Combining this procedure with the image segmentation procedure, the

input image can be decomposed into objects without any holes left on the background.

As a matter of fact, holes may exist not only on the background layer, but also on the foreground objects. Holes on an object means that the object misses some parts. Take Figure 3.3 for an example. The apple and banana are interrupted due to occlusions. If

Figure 3.3: Holes on the object layers (a) make the editing of object arrangements implausible (b).

users want to change the arrangement of objects such as moving the apple in front of the banana, the hole of the apple will be exposed. The completion of this hole is different from the one described in the previous paragraph, as a significant curvilinear features (i.e. the object’s boundary) is missing in the holes. Unfortunately, most existing image completion or inpainting methods are quite weak in propagating curvilinear features, while user vision is very sensitive to object shapes. Therefore, we propose to use a shape completion technique before the image completion procedure to resolve this problem.

Refer to Figure 3.4. The shape completion procedure is devoted to finding a “pleasing” curve (i.e. the pink curve) completing the boundary of an object and then the image completion procedure is followed to fill the hole (i.e. the red region) formed on this object using plausible colors. Furthermore, for the shape completion, boundary constraints (i.e. a point-orientation pair) are required to guarantee that the completion curve segment connects the rest object boundary continuously.

(a)

(b)

(c)

Figure 3.4: From left to right: complete the shape of an object with the pink curve; a hole on the object is formed shown by red colors; fill colors in the hole; rearrange objects for editing. The red and blue point-orientation pairs in the first column are user inputs for specifying boundary conditions.

Now that image decomposition is accomplished, object oriented image editing can

be achieved. The next goal is to perform editing tasks on vector based data. By now, the representations of image objects and the background are still pixel-based. Therefore a vectorization procedure is needed to convert a pixel representation into a vector one. This conversion is applied to both a rectangular background and objects with arbitrary topologies and shapes, shown in Figure 3.5.

Figure 3.5: The output of the framework.

The proposed framework outputs an image represented by vectorized objects, based on which the desired image editing can be performed. Thus we will be able to easily scale, bend, warp or move an object using simple gesture motions with a mouse on a mesh, where the editing of image edge features is also feasible by changing their sharpness. Some specific types of editing are illustrated in Figure 3.6-3.9. In Figure 3.8(a), the posture of a digital character (left) is edited via several mouse dragging operations, where red arrows indicate user bending or mouse moving directions. The underlying implementation is illustrated in Figure 3.8(b). Input and output meshes are shown in the left and right respectively. A mesh deformation process is involved in this editing, where colored mesh points denote the fixed (cyan), deformed (magenta) and moved (yellow) parts of a mesh. Figure 3.9 shows more examples of image warping and composition.

Figure 3.6: Image scaling. (a): original image; (b) reconstructed image from our framework; (c) magnification ($\times 2$) of the enclosed area in (b); (d) magnification (original image $\times 8$) of the enclosed area in (c); (e) magnification ($\times 8$) using bicubic interpolation.

(a)

(b)

Figure 3.7: Sharpen and blur image edge features. (a) original image (left) with blur edge features and our reconstructed image (right) sharpening the original image; (b) original image (left) and feature editing (right) with sharpness (top) and local blur (bottom).

(a)

(b)

Figure 3.8: Image warping. (a) original raster image (left), intermediate (mid) and final editing results (right); (b) mesh representation and operations

(a)

(b)

(c)

Figure 3.9: Image warping and composition. Input raster images and edited results are shown in the left and right respectively.

To summarize, to accomplish the proposed framework, we need four main technical components: image segmentation, shape completion, image completion, and image vectorization. The workflow of the framework is shown in Figure 3.10, where user interactions are marked by blue texts.

In the next four chapters (i.e., Chapters 4-7), we develop new algorithms for these four technical components. These algorithms are developed based on the requirements of our framework.

Figure 3.10: The pipeline in the framework.

Chapter 4

Image Segmentation

This chapter considers how to improve graph-based segmentation methods in order to provide more accurate and robust segmentations for colored and textured images. The graph cut algorithm [11] is chosen as our computational framework.

4.1 Motivation and Our Work

There have been many interactive image segmentation methods in the literature. Among them region-based approaches are popular ones, in which the user labels some pixels as foreground or background and then the algorithm completes the labeling for the rest [90, 107]. Examples of region-based approaches are interactive graph cut [11] or GrabCut [82], Random Walks [36, 104], and Geodesic [5, 24]. They basically treat an image as a weighted graph with nodes corresponding to pixels in the image and edges being placed between neighboring pixels, and minimize a certain energy function on the graph to produce a segmentation. One advantage of these approaches is that the cost of interactive effort is very modest since the user only needs to sketch foreground and background in a few locations. However, their performance is limited in terms of generality, quality and robustness. Specifically, these approaches are not so effective when there is no sufficient separation of foreground and background color distribution or when the images contain textures or low contrast boundaries. The segmentation contours are often jaggy and do

not adhere to the geometric features in the image. In addition, these approaches are sensitive to user inputs. Refer to Figure 4.1, for example, where three images and the segmentation results obtained by GrabCut [82] are presented. The left image contains both color and texture features, the middle image is a textured one, and the right image has a low contrast boundary. Obviously, these segmentation results are not satisfactory.

Figure 4.1: Three images (top) and the segmentation results (bottom) generated by GrabCut [82].

To overcome the aforementioned problems, this paper extends the graph cut based methods in four aspects. First, we take textures into account in the segmentation process. We design an effective and reliable texture detector, and use the detection result with the original image to create an augmented image. Then we apply graph cut on the augmented image, which makes graph cut texture aware and efficiently handle images with textures. Second, we construct geometric structure tensors to describe local color (in-

tensity) variations of the image and incorporate them into the graph cut model. Indeed, the structure tensors tell the directions and magnitudes corresponding to the maximal and minimal color (intensity) changes and thus guide the graph cut process to segment the image along the feature edges, giving more accurate results. Third, considering that the continuous-domain convex active contour model makes use of both the boundary and regional information to find a global “optimal” solution [14], we adapt it and incorporate it as a postprocess for graph cut in order to improve smoothness, accuracy and robustness of the segmentation. Fourth, when segmenting a difficult image such as camouflage, the user often struggles with scribbling under the graph cut framework. We propose to include soft constraints into the segmentation process, which allow the user to scratch to indicate the region that the boundary should pass through. This just puts a light load on the user, but efficiently guides the algorithm to find the satisfactory result.

The rest of the chapter is organized as follows. Section 4.2 reviews some related work. Section 4.3 presents our texture aware graph cut segmentation method. Section 4.4 describes how to adapt the active contour into the graph cut segmentation framework and how to design a new tool for local boundary editing. Experiments are conducted in Section 4.5.

4.2 Texture Aware Graph Cut Segmentation

This section first briefly describes the graph cut segmentation [11, 82], which serves as a foundation for our work. Then two developments are presented, which constructs a texture augmented image to replace the input image for segmentation and incorporates structure tensors into the graph cut model.

4.2.1 Graph cut segmentation

Our segmentation problem can be viewed as a binary labeling problem, as in [11, 82]. An N channel image is a set of pixels $V = \{v_{ij} : (i, j) \in \Omega\}$ where v_{ij} is an N -dimensional

vector, (i, j) is the position of v_{ij} and $\Omega \subset R^2$ is the domain of the image. For a gray image, v_{ij} is a gray value and for a color image, it is a RGB vector. Users label some pixels forming two subsets of feature vectors, namely, foreground seeds $F \subset V$ and background seeds $B \subset V$. The segmentation task is to infer a set of labels $A = \{a_{ij} : (i, j) \in \Omega\}$ that assigns each pixel v_{ij} a label $a_{ij} \in \{0, 1\}$ with 0 for background and 1 for foreground subject to the constraints that if $v_{ij} \in F$, $a_{ij} = 1$ and if $v_{ij} \in B$, $a_{ij} = 0$.

The labeling is formulated as a minimization problem:

$$\min_A (E_r(A) + \gamma E_b(A)), \quad (\text{Eq. 4.1})$$

where the energy function consists of a region term E_r and a boundary term E_b . $\gamma \geq 0$ is a tradeoff factor that is empirically set to 50 as suggested in [9]. For a color image, Rother *et al.* [82] model the region term using Gaussian Mixture Models (GMMs) for F and B so as to guide the segmentation and define $E_r(A)$ as follows:

$$E_r(A) = \sum_{(i,j) \in \Omega} (-\log(P_{GMM}(v_{ij}, a_{ij}))) \quad (\text{Eq. 4.2})$$

where $P_{GMM}(v_{ij}, a_{ij})$ denotes the probability of a pixel v_{ij} fitting the GMM of foreground ($a_{ij} = 1$) or background ($a_{ij} = 0$). The boundary item is defined to penalize both Euclidean distance and color (intensity) difference between neighboring pixels:

$$E_b(A) = \sum_{(ij,kl) \in \mathcal{E}} (1 - \delta(a_{ij}, a_{kl})) \frac{1}{\text{dist}(v_{ij}, v_{kl})} e^{-\beta \|v_{ij} - v_{kl}\|^2}, \quad (\text{Eq. 4.3})$$

where $\delta()$ is the Kronecker delta with $\delta(a_{ij}, a_{kl}) = 1$ for $a_{ij} = a_{kl}$ and $\delta(a_{ij}, a_{kl}) = 0$ for $a_{ij} \neq a_{kl}$, $\mathcal{E}$ is the set of pairs of neighboring pixels, $\text{dist}()$ is the Euclidean distance of neighboring pixels, and $\|\cdot\|$ is the L^2 -norm. The parameter β controls the smoothness and preciseness of the segmentation boundary. When $\beta = 0$, the segmentation encourages smoothness everywhere; when $\beta > 0$, the tendency to smoothness is relaxed in regions of high contrast. In general, β is chosen to be [11]

$$\beta = \frac{1}{2EP(\|v_{ij} - v_{kl}\|^2)} \quad (\text{Eq. 4.4})$$

with $EP()$ being the expectation over an image sample, which makes $E_b(A)$ switch appropriately between high and low contrast.

The minimization of (Eq. 4.1) is solved using a min-cut/max-flow algorithm [12]. It works iteratively for improving segmentation accuracy [82].

4.2.2 Combining color and texture

As observed, graph cut and GrabCut do not segment images with textures quite well due to the interference of textures. This motivates us to extend graph cut by taking textures into account. Specifically, we propose to detect textures of the image, create an augmented image that combines the original image with the texture detection result, and then apply graph cut in the augmented image to reduce the effects of the textures in the segmentation process. This requires two things: a robust and effective texture descriptor and an effective combination strategy, which are elaborated next.

To analyze the texture features of an image, the classic differential geometry theory provides an elegant tool if we consider that the image is obtained by discretizing a differentiable surface. Therefore for an N channel image $I = \{I_{ij} : (i, j) \in \Omega\}$, we represent it by a function $I(x, y) = (I_1(x, y), I_2(x, y), \dots, I_N(x, y)) : \Omega \rightarrow R^N$, with $I(i, j) = I_{ij}$. We define a *pixel-group* of size n (for an odd positive integer n) at (x, y) to be

$$\mathbf{P}(x, y) = \{I(x + k, y + l) : k, l = -\frac{n-1}{2}, \dots, \frac{n+1}{2}\} \quad (\text{Eq. 4.5})$$

which contains n^2 pixels in a region centered at (x, y) . While $I(x, y)$ just represents the color (intensity) of the image at (x, y) , $\mathbf{P}(x, y)$ represents the image in the local region, which is called the semi-local image information in [47]. When $n = 1$, $\mathbf{P}(x, y)$ degenerates to pixel $I(x, y)$.

We further construct a parametric surface $\mathbf{M}(x, y) = (x, y, \mathbf{P}(x, y))$. It has $(n^2N + 2)$ components. The first two components x and y are included to reflect the position in

space. From differential geometry, we know that the area of an element on the surface $\mathbf{M}(x, y)$ is $\|\frac{\partial \mathbf{M}}{\partial x} \times \frac{\partial \mathbf{M}}{\partial y}\| dx dy$ where

$$\frac{\partial \mathbf{M}}{\partial x} = (1, 0, \frac{\partial \mathbf{P}}{\partial x}), \quad \frac{\partial \mathbf{M}}{\partial y} = (0, 1, \frac{\partial \mathbf{P}}{\partial y}).$$

Discretely, $\frac{\partial \mathbf{P}}{\partial x} = \mathbf{P}(x+1, y) - \mathbf{P}(x, y)$ and $\frac{\partial \mathbf{P}}{\partial y} = \mathbf{P}(x, y+1) - \mathbf{P}(x, y)$. Let

$$G = \|\frac{\partial \mathbf{M}}{\partial x} \times \frac{\partial \mathbf{M}}{\partial y}\| = \sqrt{(1 + (\frac{\partial \mathbf{P}}{\partial x})^2)(1 + (\frac{\partial \mathbf{P}}{\partial y})^2) - (\frac{\partial \mathbf{P}}{\partial x} \cdot \frac{\partial \mathbf{P}}{\partial y})^2}$$

where “ $\cdot$ ” is the dot product. G actually represents the change rate of the area of a surface element to the area of a planar element. In a region containing textures, due to the variation of color (intensity), the corresponding surface tends to change much and have a large G value. Thus a texture descriptor T can be defined to be

$$T = e^{-G^2}. \quad (\text{Eq. 4.6})$$

The use of the Gaussian kernel acts as a low-pass filter in order to control the degree of details. A similar descriptor is defined in [47]. As pointed out in [47], textures are semi-local by nature. The use of the pixel-group $\mathbf{P}(x, y)$ makes our texture descriptor more robust to the noise, compared to some other texture descriptors [42, 20, 49, 83, 85, 46]. In practice, the size n is set to be 5 to 9 according to the size of texture pattern. Figure 4.2 shows one example where the left is an input color image and the right is generated using our texture descriptor with size $n = 5$.

So far we have obtained a gray image denoted by $I_t = \{t_{ij} : (i, j) \in \Omega\}$ where t_{ij} is generated from the texture descriptor T . For a texture region, the texture pattern is repeated in general. Thus the geometry of the constructed parametric surface $\mathbf{M}(x, y)$ has a tendency to be uniform in the region of the same texture pattern, which implies the G values are homogeneous in the region. Therefore the generated image I_t has the characteristic that the small variations are removed, which is just what we want.

Figure 4.2: A color image (left) and its texture descriptor (right).

Now we are in the position to construct an augmented image by combining the original image and I_t for segmentation. Figure 4.3 illustrates the construction of the augmented image from an input color image. Many previous works simply append each element of I_t to the corresponding element of the original image as a new channel. However, this does not always work well in our experience. We propose to appropriately fuse them such that for an image with well separated foreground and background, the original image $I(x, y)$ plays the major role in the augmented image and otherwise I_t contributes more. As such, we construct the augmented image $V = \{v_{ij} : (i, j) \in \Omega\}$ as follows:

$$v_{ij} = (\alpha I_{ij}, (1 - \alpha)t_{ij}) \quad (\text{Eq. 4.7})$$

where α is the weight. To determine the weight, we analyze the image and check whether the foreground and background are well separated by measuring the differences between the foreground and background as in [104, 38]. Two GMM models are established for the foreground and background. The weight α is then chosen to be the KL distance $KL(F, B)$, which is a quantity widely used to measure the similarity/dissimilarity between two pdfs. The KL distance $KL(F, B)$ is computed by

$$KL(F, B) = \frac{1}{M} \sum_{(i,j) \in \Omega} \left| \frac{\log P_{GMM}(I_{ij}, 1) - \log P_{GMM}(I_{ij}, 0)}{\log P_{GMM}(I_{ij}, 1) + \log P_{GMM}(I_{ij}, 0)} \right| \quad (\text{Eq. 4.8})$$

where M is the total number of pixels.

Figure 4.3: The construction of the augmented image from an input color image.

4.2.3 Structure tensor

The exponent part in the boundary term E_b of (Eq. 4.1) relies on a constant β and the color (intensity) difference. In general, β is fixed for an image. When an image contains regions with varying contrasts, the fixed value of β is not sufficient to adapt E_b to the regions. Refer to Figure 4.4 for example. The middle image shows the segmentation result using a fixed β , the bottom part of which does not fit the boundary well. Instead, if we use local geometric structure of the image to adaptively adjust β , the segmentation result is improved as illustrated in the right of Figure 4.4. In the following we describe how to adaptively adjust β .

Now we have had an augmented image V . Following the approach in Section 4.2.2, we similarly define a pixel-group $\mathbf{Q}(x, y)$ of size 3 from V . $\mathbf{Q}(x, y)$ can also be viewed as a parametric surface. Here we do not include x and y as additional components because the position information has already been considered by $1/\text{dist}(v_{ij}, v_{kl})$ in E_b . Consider the difference of the surface $\mathbf{Q}(x, y)$ at two points (x, y) and $(x + \Delta x, y + \Delta y)$:

$$d\mathbf{Q}(x, y) = \mathbf{Q}(x + \Delta x, y + \Delta y) - \mathbf{Q}(x, y) \approx \frac{\partial \mathbf{Q}(x, y)}{\partial x} \Delta x + \frac{\partial \mathbf{Q}(x, y)}{\partial y} \Delta y.$$

Figure 4.4: Comparison of segmentation using different methods for β . Left: an input image with foreground (red) and background (blue) seeds; Middle: the result using a fixed β ; Right: the result using adaptive β .

The squared difference is

$$\|d\mathbf{Q}(x, y)\|^2 \approx (\Delta x, \Delta y) S(x, y) \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \quad (\text{Eq. 4.9})$$

where

$$S(x, y) = \begin{pmatrix} \left(\frac{\partial \mathbf{Q}(x, y)}{\partial x}\right)^2 & \frac{\partial \mathbf{Q}(x, y)}{\partial x} \cdot \frac{\partial \mathbf{Q}(x, y)}{\partial y} \\ \frac{\partial \mathbf{Q}(x, y)}{\partial x} \cdot \frac{\partial \mathbf{Q}(x, y)}{\partial y} & \left(\frac{\partial \mathbf{Q}(x, y)}{\partial y}\right)^2 \end{pmatrix} \quad (\text{Eq. 4.10})$$

is a symmetric and semi-positive-definite matrix. S is called the structure tensor because it indicates the local geometry of the surface. Figure 4.5 visualizes S at some pixels (i, j) using ellipses defined by equation $(x - i, y - j) S(i, j) (x - i, y - j)^T = 1$.

Figure 4.5: An image (left) and its structure tensor visualized using ellipses (right).

Denote by λ_+ and λ_- the maximal and minimal eigenvalues of S and by θ_+ and θ_- the corresponding eigenvectors. Then θ_+ and θ_- are two orthogonal directions corresponding to maximal and minimal changes of the surface at (x, y) with the change rates λ_+ and

λ_- , respectively. In particular, when $\lambda_+ \gg \lambda_-$, the surface changes much more along θ_+ than along θ_- , implying that (x, y) may be located on an edge and θ_+ and θ_- are the directions across and along the edge, respectively. When $\lambda_+ \approx 0$, the local region around (x, y) is flat and may not contain any edge.

The above analysis suggests an approach for heuristically constructing a structure tensor based quantity to replace $-\beta\|v_{ij} - v_{kl}\|^2$ in E_b of (Eq. 4.1). Give a pair of neighboring pixels v_{ij} and v_{kl} , we define

$$w_s(v_{ij}, v_{kl}) = \lambda_+(i, j) \frac{\lambda_+(i, j)}{\lambda_-(i, j) + \epsilon(\lambda_+(i, j) - \lambda_-(i, j))} (k - i, l - j) S(i, j) \begin{pmatrix} k - i \\ l - j \end{pmatrix}$$

where $\epsilon \in [0, 1]$ is a small number introduced to avoid the case where λ_+/λ_- is too big (in our experiment, we set $\epsilon = 0.1$). In this definition, the first two factors are used to give a pixel with large values of λ_+ and λ_+/λ_- a big weight of being on the segmentation boundary. The last factor is approximately $\|v_{ij} - v_{kl}\|^2$, as shown in (Eq. 4.9), measuring the difference of color (intensity). Noting that $w_s(v_{ij}, v_{kl})$ is not symmetric with respect to pixels v_{ij} and v_{kl} , we introduce

$$st(v_{ij}, v_{kl}) = \kappa \frac{w_s(v_{ij}, v_{kl}) + w_s(v_{kl}, v_{ij})}{2} \quad (\text{Eq. 4.11})$$

where κ is global parameter, and finally we modify $E_b(A)$ of (Eq. 4.1) to be:

$$E_b(A) = \sum_{(ij, kl) \in \mathcal{E}} (1 - \delta(a_{ij}, a_{kl})) \frac{1}{dist(v_{ij}, v_{kl})} e^{-st(v_{ij}, v_{kl})}. \quad (\text{Eq. 4.12})$$

The parameter β in the original formula of $E_b(A)$ is replaced by κ , λ_+ and λ_- here, which avoids a fixed value of β for all the pixels in the image. Similar to β , κ can be chosen to be $EP(w_s(v_{ij}, v_{kl}) + w_s(v_{kl}, v_{ij}))$, the expectation over an image sample.

4.3 Convex Active Contour and Local Editing

This section describes how to adapt convex active contour and how to include a “soft brush” into the texture aware graph cut for further enhancement.

4.3.1 Convex active contour

The convex active contour based segmentation is expressed as the following minimization problem [14]:

$$\min_{u \in [0,1]} \left(\int_{\Omega} g_b \|\nabla u\| dx dy + \mu \int_{\Omega} h_r u dx dy \right) \quad (\text{Eq. 4.13})$$

where

- u is a classification function on image domain Ω that receives a value between 0 and 1 for each pixel. Image segmentation is to find the optimal u minimizing (Eq. 4.13), and then to extract the foreground by thresholding u , say $u(x, y) > 0.5$.
- g_b is a boundary function, commonly chosen to be

$$g_b(x, y) = 1 / (1 + \|\frac{\partial V(x, y)}{\partial x}\|^2 + \|\frac{\partial V(x, y)}{\partial y}\|^2) \quad (\text{Eq. 4.14})$$

for the augmented image $V(x, y)$.

- h_r is a region function that measures the foreground and background regions.

In (Eq. 4.13), the first term is a boundary term that favors segmentations having a tight boundary and the second term is a region term that encourages the segmentation complying with region coherence criteria defined in function h_r . Both terms are balanced by a tradeoff factor μ (which is empirically set to 10 in our experiments).

To incorporate the active contour method into the graph cut framework as a postprocess to improve the segmentation boundary, we run the following three steps after the texture aware graph cut:

- (1) Define h_r and g_b according to user's initial foreground/background inputs and the results outputted from the graph cut;
- (2) Use the Split Bregman method [35] to solve (Eq. 4.13) for u ;

- (3) Refine the segmentation by thresholding u .

What remains now is step (1) that is how to properly define h_r and g_b to ensure that the refined segmentation complies with the user inputs and respects the graph cut segmentation results.

To define h_r , three factors are considered:

- The user's initial inputs actually give a good description about the color distribution of the foreground and background regions. Thus we may use the foreground and background GMMs derived from the user's inputs to define h_r for pixel v .

$$h_r(v) = P_{GMM}(v, 0) - P_{GMM}(v, 1). \quad (\text{Eq. 4.15})$$

If $P_{GMM}(v, 0) > P_{GMM}(v, 1)$ that means the probability of pixel v fitting the GMM for background is bigger than for foreground, $h_r(v)$ is positive and thus u tends to be small in order to minimize (Eq. 4.13), which can lead to the classification of the pixel belonging to the background. Otherwise, if $P_{GMM}(v, 0) < P_{GMM}(v, 1)$, the minimization (Eq. 4.13) may result in the classification of the pixel belonging to the foreground.

- In order to incorporate the segmentation results obtained from the graph cut, we may define $h_r = 0.5 - P_{gc}(v)$ for pixel v where $P_{gc}(v)$ is a binary map with 0 for background and 1 for foreground based on the graph cut segmentation results. Then for pixel v with $P_{gc}(v) > 0.5$, h_r is negative and thus u tends to be large in order to minimize (Eq. 4.13), which favors classifying v into a foreground pixel. Likewise, a pixel v with $P_{gc}(v) < 0.5$ tends to be classified into a background seed.
- To assure that those pixels specified by the user to be classified correctly, we may directly impose a value on h_r . In particular, for a foreground seed v , we let $h_r(v) = -H$ with a very large positive number H , which makes $u(v)$ very small; and for a background seed v , we let $h_r(v) = H$.

By combining the above three situations, we have

$$h_r(v) = \begin{cases} -H, & v \in F \\ H, & v \in B \\ \omega_1(P_{GMM}(v, 0) - P_{GMM}(v, 1)) + (1 - \omega_1)(0.5 - P_{gc}(v)), & \text{otherwise} \end{cases} \quad (\text{Eq. 4.16})$$

where $\omega_1 \in [0, 1]$ is a tradeoff factor (typically set to 0.5).

Similarly, we can make use of GMMs to enhance g_b especially for weak edges where the boundary is likely to be smoothed out. Specifically, we use the map of $P_{GMM}(v, 1)$ to modify g_b at pixel v :

$$g_b = \omega_2 g_{prob} + (1 - \omega_2) g_{img}, \quad (\text{Eq. 4.17})$$

where g_{prob} and g_{img} are the results of applying the boundary function (Eq. 4.14) to the map P_F and the augmented image V , respectively, and the weight ω_2 is to control the influence of the two terms (typically set to 0.5).

4.3.2 Local boundary editing

Note that the proposed segmentation method sometimes still has problems for segmenting along weak boundaries where there is no sufficient contrast. One way to improve the segmentation performance is that the user is provided a “soft brush” to locally scratch a narrow region which the desired segmentation boundary should pass through and then the algorithm computes the satisfactory result. This approach has advantage that the user is allowed to indicate his intention and the load of interactive effort is light. To implement this tool, our approach involves two steps: (1) generating an initial path and (2) optimizing the path. The whole process of local boundary editing is illustrated in Figure 4.6.

To generate an initial path from a scratched region, we basically adopt Yang *et al*’s method [104]. That is, we first find the two endpoints A and B which are on the intersections between the scratched region and initial segmentation contour and we then use

Figure 4.6: Local boundary editing. (a) an input image with a problematic boundary C_0 (yellow) and a soft constraint Ω_0 (green); (b) two endpoints A and B ; (c) the inverted distance map I_{idm} and the shortest path P_{AB} ; (d) the problematic contour segment C_{AB}^2 is replaced by P_{AB} ; (e) the path P_{AB} is extended to $P_{A'B'}$, and $P_{A'B'}$ is to be deformed for a global smoothness; (f) the final optimized contour C_f .

Dijkstra's shortest path algorithm to find the path. Since by Yang *et al*'s method the path tends to approach the "shortcut", we propose to add a distance constraint in the path in their method to pull the path towards the middle of the scratched region. This is motivated by an observation that the user is inclined to scratch a region around the desired segmentation contour. To this end, an inverted distance map I_{idm} is built, which records the inverse of the distance from a point to the boundary of the scratched region. Then for an edge within the region, we define its distance cost to be the average of the distances of the two endpoints bounding the edge. We add this distance cost into the edge weight in Yang *et al*'s method to affect the shortest path. Figure 4.6(c) shows the inverted distance map and the shortest path.

To optimize the path, we apply the convex active contour method in a local region.

Figure 4.7: The setting for the active contour method to optimize the path in local boundary editing.

Refer to Figure 4.6(e). First, the shortest path P_{AB} is extended in two directions along the initial path to $P_{A'B'}$. Then a band region is specified, which is obtained by expanding $P_{A'B'}$ and the scratched region by 10 pixels. Next, the foreground and background seeds are defined as the inner and outer boundaries of the band region respectively, which are RC_{inside} and $RC_{outside}$ in Figure 4.7. A function h_f is designed to describe the region information, which interpolates linearly in the band region with 0 along the path $P_{A'B'}$, -1 along RC_{inside} , and 1 along $RC_{outside}$. We use h_f as the soft constraint to refine h_r for the active contour model:

$$h_r(v) = (P_{GMM}(v, 0) - P_{GMM}(v, 1)) + h_f(v) \quad (\text{Eq. 4.18})$$

where $P_{GMM}(v, 0)$ and $P_{GMM}(v, 1)$ are defined similarly as the ones in (Eq. 4.15). With the definition of h_r , the two extended parts $P_{AA'}$ and $P_{BB'}$ are deformed mainly under the guidance of the term $(P_{GMM}(v, 0) - P_{GMM}(v, 1))$ and the contour P_{AB} is evolved mainly according to the item of h_f . This assures both smoothness and accuracy in the final refined contour. Furthermore, the boundary information derived from $P_{A'B'}$ is incorporated into the boundary function g_b by:

$$g_b = g_{img} + g_{seg} \quad (\text{Eq. 4.19})$$

where g_{img} and g_{seg} are the results of applying the boundary function (Eq. 4.14) to the augmented image V and the binary map which assigns 1 to the pixels in the region bounded by RC_{inside} and $P_{A'B'}$ and 0 to the pixels in the region bounded by $RC_{outside}$ and $P_{A'B'}$. This definition of g_b assures that the refined contour align with image edge feature and prevents it from drafting too far away from $P_{A'B'}$. Finally under such setting, we run the active contour method in the band region to find the optimal path. Figures 4.6 and 4.12 are examples of using the local boundary editing tool.

4.4 Experiments

In this section, we conduct experiments to evaluate the performance of our proposed method. We first test the effects of individual technical components visually, then test the proposed algorithm as a whole, and finally perform quantitative evaluation. The testing images include the synthesized textural images generated from the Brodatz data set [2], the natural images from the Berkeley segmentation data set [1] and MSRC ground truth data set [3]. The algorithm implementation is based on public libraries of the mincut/maxflow algorithm [12] and the split Bregman solver [35].

4.4.1 Effects of individual components

Our proposed method contains four technical components: texture features, the structure tensor, the active contour and local boundary editing. We test each of them in this subsection.

Figure 4.8 shows segmentation results using different combinations of color and texture for two images with different characteristics. The image on the top row is of different patterns but similar colors and the image on the bottom row is of different colors but similar textures. Different values are chosen for the combination coefficient α in (Eq. 4.7) to generate augmented-images for performing segmentation. The α values based on the

KL distances for the two images are 0.2 and 0.8 respectively. The generated cutting contours are displayed in light blue color. It can be observed that our proposed KL distances based combination produces the best segmentation results. Equally fusing color and texture, or only using color or texture may not generate good segmentation.

Figure 4.8: Segmentation results from different combinations of color and texture, where n stands for the size of pixel-groups and α stands for the combination coefficients. $\alpha = 0.2$ for the top row and $\alpha = 0.8$ for the bottom row are obtained from the KL distances.

Figure 4.9 compares the segmentation results with and without incorporating the active contour method. Using the active contour, a smoother and more accurate cutting contour can be obtained. In Figure 4.10, we analyze the effect of the region term h_r of (Eq. 4.16) in the active contour model. Figures 4.10(a) and (b) show the images of h_r for $\omega_1 = 0$ and $\omega_1 = 1$. When $\omega_1 = 0$, the active contour method is similar to the original GrabCut algorithm since the term $(0.5 - P_{gc})$ dominates h_r ; When $\omega_1 = 1$, the term $P_{GMM}(v, 0) - P_{GMM}(v, 1)$ guides the region function. That may cause h_r to cover the redundant parts around the horse feet and may also smooth out some contour corners such as the region around the horse ears. In our experiments, we set $\omega_1 = 0.5$ which works well.

Figure 4.9: Segmentation results with (b) and without (c) the active contour model.

Figure 4.11 shows three examples. The input images are displayed on the left column. The segmentation results using the original GrabCut are given in the middle column and the segmentation results using our proposed method with the structure tensor are on the right column. It can be seen that the use of the structure tensor makes the algorithm more respect the local geometric structure of the image and pushes the cutting contour

Figure 4.10: Effects of different combinations for h_r . (a) The image of h_r for $\omega_1 = 0$; (b) The image of h_r for $\omega_1 = 1$; (c-e) The segmentation results correspond to $\omega_1 = 0, 0.5$, and 1, respectively.

to align with feature edges of the image.

Figure 4.11: Comparing the segmentation results without (middle) and with (right) the structure tensor.

We now evaluate the proposed local boundary editing. Figure 4.12 shows an example where the image has a very low contrast around the upper-right part of the boundary between the object and the background. In this case, only scratching foreground and background seeds is usually not efficient. As illustrated in Figures 4.12 (b) and (d), the weak boundary cannot be detected precisely. We allow users to input a soft constraint on the problematic boundary by scratching a region indicating where the desired boundary should pass through, and then a much better result is obtained, which is also smoother than the result given in [104].

4.4.2 Overall performance

We apply the proposed algorithm on six images with different features. The first two are synthetic textured images and are shown in the top row of Figure 4.13. The second two

Figure 4.12: Soft boundary constraints improve segmentations of an object with low contrast boundary. (a) foreground/background input1, (b) output1, (c) foreground/background input2, (d) output2, (e) with an additional soft boundary constraint (in green), (f) output3, (g) the result given in [104].

are real-world images with textures, shown in the middle row of Figure 4.13. The last two are colored images in the bottom row of Figure 4.13. The segmentation result of each image is placed on the right side of the image. It can be found that our algorithm performs consistently well for all these images.

For comparison, Figure 4.14 presents the segmentation results obtained from the original GrabCut [82]. It can be seen that the GrabCut does not work well for those images with textures. In addition, if we remove the active contour process from the proposed texture aware segmentation method, the corresponding segmentation results are not smooth and accurate, as depicted in Figure 4.15.

We further examine our algorithm in terms of robustness to user inputs and smooth-

Figure 4.13: Segmentation on images with different features using our proposed method. Top row: synthetic textured images; Middle row: real-world images with textures; Bottom row: colored images. The parameters n and α are the size of pixel-groups and the KL distance in (Eq. 4.7).

Figure 4.14: Segmentation results of GrabCut [82].

ness of the output boundaries. It has been known that the classic graph cut based methods are sensitive to the number of seeds and have the “short cut” problem. In Figure 4.16, the first row displays the input images with different seeds, and the second and third rows display the segmentation results from GrabCut and our algorithm, respectively. We can see that GrabCut produces different segmentation results for different user inputs but our method can output consistent segmentation results. Moreover, the “short cut” problem is alleviated in our method. Figure 4.17 compares the smoothness

Figure 4.15: The segmentation results using the proposed method but without incorporating the active contour model.

of boundaries of the segmentation results. While the GrabCut algorithm outputs a jaggy boundary, our method produces a smoother and more accurate contour.

Figure 4.16: The segmentation results with different user inputs.

Each technique component of our method has been investigated individually in previous works. Refer to Figure 4.18 for illustration. Our proposed image segmentation method carefully improves and integrates them.

4.4.3 Quantitative evaluation on the benchmark data set

We perform quantitative evaluation using MSRC data set [3], which contains 50 test images associated with trimaps and ground truth. The MSRC data set is chosen because

Figure 4.17: Comparison of segmentation results in terms of smoothness.

Techniques	Papers						Our method
	[52]	[66]	[78]	[87]	[109]	[111]	
Graph cut		✓		✓			✓
Texture descriptor	✓						✓
Structure tensor						✓	✓
Constrained active contour			✓				✓
Local boundary editing		✓		✓	✓		✓

Figure 4.18: Four technique components.

it is the only publicly available data set with trimap and ground truth provided and has been commonly used in other research papers though the data set is somewhat biased due to the small unknown regions of trimaps.

The trimaps provided by the MSRC data set are used as the user inputs for all algorithms. The average error rate over all 50 images is utilized as the measurement of accuracy for segmentation. The error rate is defined as the ratio of the number of wrongly labeled pixels to the total number of unlabeled pixels. The wrongly labeled pixels are identified by differences between the ground truth images and the segmentation results.

We first test several variants of our method, where texture features, the structure tensor, and the active contour model are incorporated in the graph cut framework selectively. This helps to analyze the importance and effectiveness of individual components quantitatively. Table 4.1 shows the achieved error rates. It can be seen that the texture features and the active contour are more effective in improving the grabcut algorithm than the structure tensor.

Table 4.1: Error rates of our method and its variants

Methods	Error rate
Our proposed method	3.64%
without texture features	4.17%
without structure tensor	3.85%
without active contour	4.97%

We then compare our method with several state-of-the-art interactive image segmentation algorithms. Table 4.2 summarizes the error rates achieved by various algorithms. The error rates are either quoted directly from the best results reported in literature or obtained from our implementation. It can be seen that the proposed method achieves very low error rate, compared to the state-of-the-art. We visualize the segmentation results of the 50 test images in Figures 4.19 and 4.20, where the error rate ε for each image is also provided. In this test, the parameters in our method are set the same for

all the 50 images and the size of pixel-groups is chosen to be 5. It is worth mentioning that a lower error rate of 3.3% on the MSRC data set can be achieved by applying an adaptive threshold method (AT) as a post-processing step to the segmentation by transduction (SBT) [37] while the SBT itself produces an error rate of 5.4%, as reported in [27]. However, as explained in [27], the large reduction from 5.4% to 3.3% in the error rate is due to the particular form of the seeds in MSRC data set, where the unlabeled region only covers a small band along the object boundary. As shown in [73], the AT method may not work well for images with large and unregulated unknown regions. In contrast, our method is very general and works well on images even with large unknown regions as seen from the examples given in the paper.

Table 4.2: Error rate comparison of our method and other methods

Graph cut based methods:	Error rate
GraphCut of [11]	6.7% (reported in [80])
LazySnapping	6.65% (reported in [82])
GrabCut	5.66% (reported in [82])
Geodesic graph cut [5]	4.8% (reported in [80])
Our proposed method	3.64%
Other methods:	Error rate
GMMRF [9]	7.9% (reported in [27])
Random walker [36]	5.4% (reported in [27])
Segmentation by transduction [27]	5.4% (reported in [27])
Geodesic segmentation [5]	5.21% (reported in [73])
Constrained random walk [104]	4.08% (reported in [73])
Convex active contours [73]	3.77% (reported in [73])
Segmentation by transduction with AT [37]	3.3% (reported in [27])

The statistics on processing times of our method is shown in Figure 4.21, where the time in the final row is a mean value of segmenting all 50 images in MSRC. And the processing times for the two major technical parts: enhanced graph cut model and constrained active contour model are listed separately. It should be noted that the statistics do not include the times of user interactions. And the experiments run on a PC

Figure 4.19: The segmentation results on the MSRC data set using our method. The ϵ given in parenthesis is the error rate.

Figure 4.20: The continued part of Figure 4.19.

with Intel Core i7-2760QM 2.4 GHz CPU and 8GB RAM. With our proposed method, most images can be segmented in 10 seconds.

Images	GrabCut(sec)	Our method (sec)	
		Enhanced Graphcut model	Constrained active contour model
Fig. 4.7	1.42	4.07	0.06
	3.69	7.13	0.16
	3.74	5.89	0.21
	4.34	5.54	0.33
	4.17	5.45	0.24
	5.22	6.87	0.26
Fig. 4.10	1.07	2.76	0.06
	1.29	2.58	0.04
Fig. 4.11	3.70	5.19	0.30
Fig. 4.13	7.33	10.73	0.33
	25.76	29.89	0.32
	11.86	13.82	0.21
	6.84	8.61	0.12
Fig. 4.16	3.79	4.08	0.09
MSRC(mean)	5.95	9.76	0.30

Figure 4.21: The statistics of processing times.

Chapter 5

Euler Arc Splines for Shape Completion

This chapter considers shape completion that is a process of completing contours beyond occlusions or across gaps. We introduce a special arc spline called an *Euler arc spline* as the basic form for visually pleasing completion curves and develop an algorithm for shape completion using Euler arc spline.

5.1 Motivation and Our Work

Our approach is inspired by Kimia *et al.*'s work of using Euler curves for shape completion [53]. Psychological studies show that Euler curves have good fit to the way that the human eyes complete curves [89]. However, there are some drawbacks with Kimia *et al.*'s approach. First, due to complicated transcendental function representation of Euler curves, it is difficult or expensive to compute with Euler curves. In particular, for rendering purpose, Euler curves are often approximated by line segments or arc segments. Second, Kimia *et al.*'s algorithm finds the solution by a numerical approach that minimizes the distance between the second given point and the last point of the Euler curve. Sometimes the numerical approach might not converge or the approximate solution is not good enough. As a consequence, the resulting Euler curve will not pass through the

second point. This will violate the aesthetics of shape completion since human perception is sensitive to gaps. Third, Euler curves are not compatible with non-uniform rational B-splines (NURBS) which are an industry standard in CAD/CAM.

These limitations motivate us to explore new completion curve models and shape completion algorithms. It is observed by Horn [45] that the optimal multi-arcs that approximate the “smoothest” curve arcs tend to be of equal length and curvature changes more or less linearly along the curve. Therefore in this chapter, we propose a special arc spline called an *Euler arc spline* as the completion curve primitive for shape completion. The Euler arc spline consists of G^1 continuous arcs of the same length with curvature changing linearly from one arc to another. It is a good approximation to an Euler curve. Meanwhile, Euler arc spline curves are simple for computation, rendering and other geometric processing such as offsetting and distance query. They are non-uniform rational B-spline curves, facilitating the use with standard graphics packages. Another arc spline similar to an Euler arc spline has been introduced in [66], which is called discrete clothoid where the first and last arcs are half the length of the others. The approximation properties of discrete clothoid are analyzed. Particularly, if a clothoid is approximated by a discrete clothoid of n arcs, the approximation error is of order $O(1/n^2)$. However, no algorithm is provided to construct a discrete clothoid for shape completion.

We also propose a shape completion algorithm using the proposed Euler arc splines. We reduce the construction of completion curve from two point-orientation pairs to minimizing the sum of squares of nonlinear functions in two unknowns, which is then solved by the Levenberg-Marquardt algorithm. The underlying idea of our approach is to distribute the approximate error between the second given point and the last point of the completion curve to all the arcs. An optimization procedure is performed to minimize the errors. Consequently, the interpolation of two given point-orientation pairs with our

completion curve is always guaranteed, which overcomes the drawbacks of Kimia *et al.*'s algorithm.

The main contributions of this work therefore include: (1) introducing Euler arc splines as a new completion curve representation; (2) developing an algorithm to construct an Euler arc spline curve to interpolate two given point-orientation pairs for shape completion.

In Section 5.2, we first identify and analyze the parameters needed to specify an Euler arc spline and show that Euler arc spline curves exhibit similar properties as Euler curves. Then in Section 5.3, the shape completion algorithm using Euler arc spline curves is presented. Finally, experimental results are provided in Section 5.4.

5.2 Arc Splines and Euler Arc Spline

In this section, we first examine the representation of a G^1 continuous arc spline, then introduce Euler arc splines, and identify parameters for the construction of an Euler arc spline in a way that is suitable for shape completion. Some properties of Euler arc splines are also analyzed.

5.2.1 G^1 continuous arc splines

A directed arc can be uniquely defined by its starting point, orientation angle that is a rotation angle from the positive x -axis to the tangent direction of the arc at the starting point, an arc-length and a central angle. If the direction of the arc is clockwise, the central angle is negative; Otherwise, the angle is positive. Suppose there are n such directed arcs. They are connected one by one at the endpoints with tangent continuity as shown in Figure 5.1. Then they form a G^1 continuous arc spline.

Proposition 5.1 *A G^1 continuous arc spline consists of n arcs $\mathcal{A}_i, i = 1, 2, \dots, n$. If each $\mathcal{A}_i$ has an arc-length of s_i and a central angle of $\Delta\theta_i$, then the i -th arc $\mathcal{A}_i$*

 Figure 5.1: A G^1 continuous arc spline curve

($i = 1, 2, \dots, n$) has the following ending point and orientation angle θ_i at the ending point:

$$\theta_i = \theta_0 + \sum_{j=1}^i \Delta\theta_j \quad (\text{Eq. 5.1})$$

$$P_i = P_0 + \sum_{j=1}^i s_j \frac{2 \sin \frac{\Delta\theta_j}{2}}{\Delta\theta_j} T(\theta_0 + \phi_j) \quad (\text{Eq. 5.2})$$

where P_0, θ_0 are the starting point and the orientation angle at P_0 of the first arc $\mathcal{A}_1$, $\phi_j = \frac{\theta_{j-1} + \theta_j}{2} - \theta_0$, and $T(\theta)$ denotes unit vector $\begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}$. That is, a G^1 continuous arc spline with n arcs can be specified by $P_0, \theta_0, s_1, \Delta\theta_1, \dots, s_n, \Delta\theta_n$.

Proof: Since $\Delta\theta_j$ is the central angle, $\theta_j - \theta_{j-1} = \Delta\theta_j$. Thus $\theta_i = \theta_{i-1} + \Delta\theta_i = \theta_{i-2} + \Delta\theta_{i-1} + \Delta\theta_i = \dots = \theta_0 + \sum_{j=1}^i \Delta\theta_j$.

Consider vector $P_{i-1}P_i$ (see Figure 5.2). Its length is the chord-length of the arc $\mathcal{A}_i$. The radius of $\mathcal{A}_i$ is $\frac{s_i}{\Delta\theta_i}$. Therefore the chord-length is $2 \frac{s_i}{\Delta\theta_i} \sin \frac{\Delta\theta_i}{2}$. The direction of $P_{i-1}P_i$ can be obtained by rotating $T(\theta_{i-1})$ by an angle of $\frac{\Delta\theta_i}{2}$. $T(\theta_{i-1})$ can be obtained by rotating $T(\theta_{i-2})$ by an angle of $\Delta\theta_{i-1}$. We continue this and can conclude that the direction of $P_{i-1}P_i$ is obtained by rotating $T(\theta_0)$ by an angle:

$$\frac{\Delta\theta_i}{2} + \Delta\theta_{i-1} + \dots + \Delta\theta_1 = \frac{\theta_{i-1} + \theta_i}{2} - \theta_0 = \phi_i.$$

Thus $P_{i-1}P_i = 2\frac{s_i}{\Delta\theta_i} \sin \frac{\Delta\theta_i}{2} M(\phi_i)T(\theta_0)$ where $M(\phi_i) = \begin{bmatrix} \cos \phi_i & -\sin \phi_i \\ \sin \phi_i & \cos \phi_i \end{bmatrix}$ is a rotation matrix,. Then we have $P_i = P_{i-1} + P_{i-1}P_i = P_{i-2} + P_{i-2}P_{i-1} + P_{i-1}P_i = \dots = P_0 + \sum_{j=1}^i P_{j-1}P_j = P_0 + \sum_{j=1}^i s_j \frac{2 \sin \frac{\Delta\theta_j}{2}}{\Delta\theta_j} M(\phi_j)T(\theta_0) = P_0 + \sum_{j=1}^i s_j \frac{2 \sin \frac{\Delta\theta_j}{2}}{\Delta\theta_j} T(\theta_0 + \phi_j)$.

Figure 5.2: One arc segment in an arc spline curve

It is easy to see that for $i = 1, \dots, n-1$, P_i and θ_i are also the starting point and its associated orientation angle of the $(i+1)$ -th arc. While P_i and θ_i depend on the coordinate system, central angles $\Delta\theta_i$ and arc-lengths s_i are coordinate system independent.

An arc spline can be represented as a NURBS curve. Now we show how to convert a G^1 continuous arc spline into a quadratic NURBS curve. Consider arc $\mathcal{A}_i$. Once we have P_{i-1} , P_i and $\Delta\theta_i$, we can represent $\mathcal{A}_i$ as a quadratic rational Bézier curve. To ensure the positivity of weights, we require $|\Delta\theta_i| < \pi$.

In case $|\Delta\theta_i| \geq \pi$, we split the arc into two sub-arcs by inserting a new point $P_{i-\frac{1}{2}}$ such that the two sub-arcs have the same central angle $\frac{\Delta\theta_i}{2}$. By some calculations, the new point $P_{i-\frac{1}{2}}$ has an expression:

$$P_{i-\frac{1}{2}} = P_{i-1} + s_i \frac{2 \sin \frac{\Delta\theta_i}{4}}{\Delta\theta_i} T\left(\frac{\Delta\theta_i}{4} + \theta_{i-1}\right).$$

If $|\Delta\theta_i| < \pi$, then the arc can be defined by a rational Bézier curve whose three control points and weights are P_{i-1} , $P_{i-1} + \frac{s_i}{\Delta\theta_i} \tan \frac{\Delta\theta_i}{2} T(\theta_{i-1})$, P_i and $1, \cos \frac{\Delta\theta_i}{2}, 1$. Refer to Figure 5.3 for an illustration.

Figure 5.3: One arc segment and its three Bézier control points

Summarizing the above discussion, we can design an algorithm that convert the arc spline into a quadratic NURBS curve:

Input: G^1 continuous arc-spline

Output: a quadratic NURBS curve

step 1. Find P_{i-1} , P_i and $\Delta\theta_i$ for each arc using Proposition 5.1.

step 2. Check each arc. If $|\Delta\theta_i| \geq \pi$, split the arc into two sub-arcs.

step 3. Re-organize all the arcs. Suppose now we have m arcs and they are defined by points $Q_0, Q_1, Q_2, \dots, Q_m$, central angles $\Delta\theta_1, \Delta\theta_2, \dots, \Delta\theta_m$, and arc-lengths $s_1, s_2, \dots, s_m$. The central angles are all less than π .

step 4. Construct the NURBS curve with control points

$$Q_0, Q_{\frac{1}{2}}, Q_1, Q_{\frac{3}{2}}, Q_2, \dots, Q_m$$

where $Q_{i-\frac{1}{2}} = Q_{i-1} + \frac{s_i}{\Delta\theta_i} \tan \frac{\Delta\theta_i}{2} T(\theta_{i-1})$, weights

$$1, \cos \frac{\Delta\theta_1}{2}, 1, \cos \frac{\Delta\theta_2}{2}, \dots, \cos \frac{\Delta\theta_m}{2}, 1,$$

and knot sequence

$$\{0, 0, 0, 1, 1, 2, \dots, m-2, m-1, m-1, m, m, m\}.$$

5.2.2 Euler arc splines

Definition 5.1 *An Euler arc spline (EAS) is defined to be a curve consisting of several arcs satisfying the three conditions: (1) the arcs have the same arc-length; (2) the arcs are joined to form a G^1 continuous curve; and (3) the curvatures of the arcs vary linearly from one arc to another.*

An Euler arc spline is a special arc spline that has constraints on arc-lengths and curvatures. It is so named because it can be considered as an extension of an Euler curve in the sense that each point on the Euler curve is replaced by an arc.

Consider an Euler arc spline with n arcs $\mathcal{A}_i, i = 1, 2, \dots, n$. Assume $\mathcal{A}_i$ starts at P_{i-1} with orientation angle θ_{i-1} and ends at P_i with orientation angle θ_i , has curvature κ_i and arc-length s . Based on the definition, the curvature can be defined by $\kappa_i = \alpha s(i-1) + \kappa_1$ for $i \geq 1$ where κ_1 and α are two constants, called the initial curvature and curvature slope, respectively. From κ_i and s , we can derive

$$\Delta\theta_i = \kappa_i s = \alpha s^2(i-1) + \kappa_1 s.$$

From Proposition 5.1, we can derive P_i and θ_i . That is,

Proposition 5.2 *An Euler arc spline can be defined by six parameters: $P_0 = (x_0, y_0), \theta_0, n, s, \alpha$ and κ_1 , from which we have*

$$\begin{aligned} \theta_i &= \theta_0 + i\kappa_1 s + \frac{(i-1)i}{2} s^2 \alpha \\ P_i &= P_0 + \sum_{j=1}^i \frac{2 \sin \frac{\kappa_j s}{2}}{\kappa_j} T(\phi_j + \theta_0) \end{aligned}$$

where $\phi_j = \frac{\theta_{j-1} + \theta_j}{2} - \theta_0 = \frac{s^2(j-1)^2 \alpha + (2j-1)\kappa_1 s}{2}$.

Moreover, there are constraints on s, α, κ_1, n to ensure that each arc is not longer than a full circle, which gives $-2\pi \leq \Delta\theta_i \leq 2\pi$. Substituting κ_i into this constraint, we obtain

$$-2\pi \leq s^2\alpha(i-1) + \kappa_1 s \leq 2\pi, \quad i = 1, 2, \dots, n. \quad (\text{Eq. 5.3})$$

Proposition 5.3 *For an Euler arc spline with each arc segment not longer than a full circle, its initial curvature κ_1 and curvature slope α satisfy:*

$$-\frac{2\pi}{s} \leq \kappa_1 \leq \frac{2\pi}{s}, \quad (\text{Eq. 5.4})$$

$$-\frac{4\pi}{(n-1)s^2} \leq -\frac{2\pi + \kappa_1 s}{(n-1)s^2} \leq \alpha \leq \frac{2\pi - \kappa_1 s}{(n-1)s^2} \leq \frac{4\pi}{(n-1)s^2}. \quad (\text{Eq. 5.5})$$

Proof: The proof is straightforward just by letting $i = 1$ and $i = n$ in (Eq. 5.3), respectively.

Proposition 5.3 shows that when n increases, the range of α decreases if s remains unchanged. If α remains unchanged, then when n increases, s must decrease. In particular, when n goes infinity, s tends to zero, which implies that an Euler arc spline converges to an Euler curve when the number of arcs goes to infinity.

Furthermore, we have

Proposition 5.4 *Given constants $\kappa_1, \alpha, s(> 0)$, and a natural number n satisfying $|\kappa_1 s| \leq 2\pi$ and $|s^2\alpha(n-1) + \kappa_1 s| \leq 2\pi$, there exists an Euler arc spline with n arc segments having a total arc-length of ns and curvature $\kappa_i = \alpha s(i-1) + \kappa_1$ varying linearly in arc index i . The Euler arc spline is unique up to a rigid transformation.*

Proof: When $\alpha \geq 0$, for all $i = 1, 2, \dots, n$, we have

$$-2\pi \leq \kappa_1 s \leq \alpha(i-1)s^2 + \kappa_1 s \leq \alpha(n-1)s^2 + \kappa_1 s \leq 2\pi.$$

Similarly, when $\alpha < 0$, we have

$$-2\pi \leq \alpha(n-1)s^2 + \kappa_1 s \leq \alpha(i-1)s^2 + \kappa_1 s \leq \kappa_1 s \leq 2\pi.$$

We let the central angles $\Delta\theta_i = \kappa_i s$. Then $|\Delta\theta_i| \leq 2\pi$. If we arbitrarily choose a starting point P_0 and an orientation θ_0 , Proposition 5.4 gives the expression of an Euler arc spline satisfying the requirements. It also shows that such an Euler arc spline is unique up to a translation dependent of P_0 and a rotation dependent of θ_0 .

Figure 5.4: Four types of shape of an Euler arc spline

The shape of an Euler arc spline curve can be classified into four types depending on the signs of α and κ_1 , which are illustrated in Figure 5.4.

In addition to having curvature change linearly from one arc to another, Euler arc splines also have other nice properties as listed below. These properties include or are similar to similarity transformation invariance, symmetry, extensibility, smoothness, and roundedness, which are required by the aesthetics of curves.

- An Euler arc spline is invariant to translation, rotation, and scaling.

In fact, suppose an Euler arc spline curve $\mathcal{A}$ is defined by $P_0, \theta_0, n, s, \alpha$ and κ_1 . A translation applied to $\mathcal{A}$ is achieved by just applying the translation to P_0 . If we rotate $\mathcal{A}$ around the origin by an angle ϕ , this is achieved by applying the rotation to P_0 and meanwhile updating θ_0 to $\theta_0 + \phi$. If $\mathcal{A}$ is multiplied by c , the result is a new Euler arc spline defined by $cP_0, \theta_0, n, cs, \frac{\alpha}{c^2}$ and $\frac{\kappa_1}{c}$.

- The Euler arc spline defined by parameters $P_0, \theta_0, n, s, \alpha$ and κ_1 coincides with the Euler arc spline defined by $P_n, \theta_n + \pi, n, s, \alpha$ and $-\kappa_n$, where $\kappa_n = \alpha s(n-1) + \kappa_1$, $\theta_n = \theta_0 + n\kappa_1 s + \frac{n(n-1)}{2}s^2\alpha$ and $P_n = P_0 + \sum_{j=1}^n \frac{2 \sin \frac{\kappa_j s}{2}}{\kappa_j} T(\frac{s^2(j-1)^2\alpha + (2j-1)\kappa_1 s}{2} + \theta_0)$.

To prove this, we denote the first curve by $r(t), t \in [0, ns]$ and the second one by $\bar{r}(t), t \in [0, ns]$ and show that $r(hs+l) = \bar{r}((n-h-1)s+s-l)$ for $h = 0, 1, \dots, n-1$ and $l \in [0, s]$. In fact,

$$\begin{aligned} r(hs+l) = P_0 &+ \sum_{j=1}^h \frac{2 \sin \frac{\kappa_j s}{2}}{\kappa_j} T(\phi_j + \theta_0) \\ &+ \frac{2 \sin \frac{\kappa_{h+1} l}{2}}{\kappa_{h+1}} T(\theta_h + \kappa_{h+1} \frac{l}{2}) \end{aligned}$$

and

$$\begin{aligned} \bar{r}((n-h-1)s+s-l) &= P_n \\ &+ \sum_{j=1}^{n-h-1} \frac{2 \sin \frac{\bar{\kappa}_j s}{2}}{\bar{\kappa}_j} T(\bar{\phi}_j + \theta_n + \pi) \\ &+ \frac{2 \sin \frac{\bar{\kappa}_{n-h} (s-l)}{2}}{\bar{\kappa}_{n-h}} T(\bar{\theta}_{n-h-1} + \bar{\kappa}_{n-h} \frac{s-l}{2}) \end{aligned}$$

where $\bar{\kappa}_j = \alpha s(j-1) - \kappa_n = -\alpha s(n-j) - \kappa_1 = -\kappa_{n-j+1}$, $\bar{\theta}_j = \pi + \theta_{n-j}$, and $\bar{\phi}_j = \frac{\alpha s^2(j-1)^2 - (2j-1)\kappa_n s}{2}$. Thus $\bar{r}((n-h-1)s+s-l) = P_0 + \sum_{j=1}^n \frac{2 \sin \frac{\kappa_j s}{2}}{\kappa_j} T(\phi_j + \theta_0) + \sum_{j=1}^{n-h-1} \frac{2 \sin \frac{\kappa_{n+1-j} s}{2}}{\kappa_{n+1-j}} T(\phi_{n+1-j} + \theta_0 + \pi) + \frac{2 \sin \frac{\kappa_{h+1} (s-l)}{2}}{\kappa_{h+1}} T(\pi + \theta_{h+1} - \kappa_{h+1} \frac{s-l}{2}) = P_0 + \sum_{j=1}^h \frac{2 \sin \frac{\kappa_j s}{2}}{\kappa_j} T(\phi_j + \theta_0) + \frac{2 \sin \frac{\kappa_{h+1} l}{2}}{\kappa_{h+1}} T(\theta_h + \kappa_{h+1} \frac{l}{2}) = r(hs+l)$.

- Two Euler arc curves defined by $P_0, \theta_0, m, s, \alpha, \kappa_1$, and $P_m, \theta_m, n-m, s, \alpha, \alpha sm + \kappa_1$, coincide with the Euler arc curve defined by $P_0, \theta_0, n, s, \alpha, \kappa_1$, each in its own section.
- An Euler arc spline curve is at least G^1 continuous.
- If the two point-orientation pairs lie on a circle, then there exists an EAS interpolating the point-orientation pairs, which coincides with the circle. This is because a circle is a special case of an Euler arc spline with $\alpha = 0$.

5.3 Shape Completion Algorithm

The problem of shape completion using an EAS can be stated as follows: Given two points $P_A = (x_A, y_A)$ and $P_B = (x_B, y_B)$ with orientations θ_A and θ_B , find an EAS curve consisting of n arcs that interpolates them, which is illustrated in Figure 5.5.

Figure 5.5: Shape completion with an EAS curve

As discussed in Section 5.2.2, an Euler arc curve can be defined by $P_0, \theta_0, n, s, \alpha$ and κ_1 . For our problem, we obtain the first three parameters immediately from the interpolation condition at one given point: $P_0 = P_A, \theta_0 = \theta_A$. When n is specified, only s, α and κ_1 are to be determined. By Proposition 5.4, $\theta_n = \theta_0 + n\kappa_1 s + \frac{(n-1)n}{2}s^2\alpha$. Letting $\theta_n = \theta_B$ gives

$$\alpha = \frac{2(\theta_B - \theta_A - n\kappa_1 s)}{(n-1)ns^2} \quad (\text{Eq. 5.6})$$

We further need to let the last point of the Euler arc spline curve interpolate the second point P_B . Thus

$$P_B = P_A + \sum_{j=1}^n \frac{2 \sin \frac{\kappa_j s}{2}}{\kappa_j} T(\phi_j + \theta_A) \quad (\text{Eq. 5.7})$$

which gives two equations for determining s and κ_1 . Unfortunately, these equations are highly non-linear, which makes them difficult to be solved. Kimia *et al.* [53] propose an numerical method by sampling s and κ_1 values to find the solution. Since usually an

approximate solution is obtained, it sometimes happens that the approximate curve does not touch P_B . However, human perception is sensitive to the gap.

In this section, we present a new approach to find s and κ_1 . Our basic idea is that we perturb the arc-length of each arc by an δ_i to ensure the interpolation, which is somewhat equivalent to distributing the deficiency between the last point of the Euler arc spline and P_B to all the arcs, and we want to find the best s and κ_1 so that the perturbation amount is minimized. One advantage of our approach is that the constructed curve always interpolate the point-orientation pairs. Thus instead of solving (Eq. 5.7), we search for a solution to an optimization problem:

$$(s, \kappa_1) = \arg \min_{s, \kappa_1} \left(\min_{\substack{\delta_i \\ P_n^\delta = P_B}} \sum_{i=1}^n \delta_i^2 \right) \quad (\text{Eq. 5.8})$$

where P_n^δ is the last point of the perturbed curve. The next two sub-sections explain how to solve this optimization problem.

5.3.1 Perturbation for interpolation

Suppose we have already had an EAS that may not interpolate P_B . Now we fix the central angles, and change the arc-length of each arc to $s_i = s + \delta_i$ with a perturbation δ_i . There are many choices for δ_i to achieve interpolation. We determine δ_i by solving the following minimization problem:

$$\begin{aligned} & \min \sum_{i=1}^n \delta_i^2 \\ & \text{subject to } P_n^\delta = P_B \end{aligned} \quad (\text{Eq. 5.9})$$

This is a constrained optimization problem. Based on Proposition 5.1,

$$P_n^\delta = P_A + \sum_{j=1}^n (s + \delta_j) \frac{2 \sin \frac{\Delta\theta_j}{2}}{\Delta\theta_j} T(\theta_A + \phi_j).$$

Let

$$V_j = \frac{2 \sin \frac{\Delta\theta_j}{2}}{\Delta\theta_j} T(\theta_A + \phi_j) = \frac{2 \sin \frac{\Delta\theta_j}{2}}{\Delta\theta_j} \begin{bmatrix} \cos(\theta_A + \phi_j) \\ \sin(\theta_A + \phi_j) \end{bmatrix}.$$

By introducing Lagrangian multipliers λ_1 and λ_2 , we convert the minimization problem (Eq. 5.9) into an unconstrained one:

$$\begin{aligned} \min G(\delta_1, \dots, \delta_n, \lambda_1, \lambda_2) = & \min \left\{ \sum_{i=1}^n \delta_i^2 \right. \\ & + \lambda_1 (P_A^x - P_B^x + \sum_{j=1}^n s V_j^x + \sum_{j=1}^n \delta_j V_j^x) \\ & \left. + \lambda_2 (P_A^y - P_B^y + \sum_{j=1}^n s V_j^y + \sum_{j=1}^n \delta_j V_j^y) \right\} \end{aligned}$$

where the superscripts “ x ” and “ y ” stand for the x - and y -components of vectors.

To solve the optimization problem, we take the partial derivatives of $G(\delta_1, \dots, \delta_n, \lambda_1, \lambda_2)$ with respect to δ_k , and λ_1, λ_2 , respectively, and set the results to zero. This results in the following equations:

$$\frac{\partial G}{\partial \delta_k} = 2\delta_k + \lambda_1 V_k^x + \lambda_2 V_k^y = 0, \quad k = 1, \dots, n \quad (\text{Eq. 5.10})$$

and

$$\begin{aligned} \frac{\partial G}{\partial \lambda_1} &= P_A^x - P_B^x + \sum_{j=1}^n s V_j^x + \sum_{j=1}^n \delta_j V_j^x = 0 \\ \frac{\partial G}{\partial \lambda_2} &= P_A^y - P_B^y + \sum_{j=1}^n s V_j^y + \sum_{j=1}^n \delta_j V_j^y = 0 \end{aligned} \quad (\text{Eq. 5.11})$$

From Eq.(Eq. 5.10), we obtain

$$\delta_k = -\frac{\lambda_1 V_k^x + \lambda_2 V_k^y}{2}. \quad (\text{Eq. 5.12})$$

Substituting them into Eq.(Eq. 5.11) arrives at two linear equations in λ_1 and λ_2 :

$$\begin{aligned} \sum_{j=1}^n (V_j^x)^2 \lambda_1 + \sum_{j=1}^n (V_j^x V_j^y) \lambda_2 &= 2(P_A^x - P_B^x + s \sum_{j=1}^n V_j^x) \\ \sum_{j=1}^n (V_j^x V_j^y) \lambda_1 + \sum_{j=1}^n (V_j^y)^2 \lambda_2 &= 2(P_A^y - P_B^y + s \sum_{j=1}^n V_j^y) \end{aligned}$$

The solutions of the two linear equations are

$$\begin{aligned} \lambda_1 &= \frac{2 \sum_{j=1}^n (V_j^y)^2 (P_A^x - P_B^x + s \sum_{j=1}^n V_j^x) - 2 \sum_{j=1}^n (V_j^x V_j^y) (P_A^y - P_B^y + s \sum_{j=1}^n V_j^y)}{\sum_{j=1}^n (V_j^x)^2 \sum_{j=1}^n (V_j^y)^2 - (\sum_{j=1}^n V_j^x V_j^y)^2} \\ \lambda_2 &= \frac{2 \sum_{j=1}^n (V_j^x)^2 (P_A^y - P_B^y + s \sum_{j=1}^n V_j^y) - 2 \sum_{j=1}^n (V_j^x V_j^y) (P_A^x - P_B^x + s \sum_{j=1}^n V_j^x)}{\sum_{j=1}^n (V_j^x)^2 \sum_{j=1}^n (V_j^y)^2 - (\sum_{j=1}^n V_j^x V_j^y)^2} \end{aligned} \quad (\text{Eq. 5.13})$$

Thus we can conclude

Proposition 5.5 *Given two point-orientation pairs (P_A, θ_A) and (P_B, θ_B) , and also n, s, α, κ_1 , let $\Delta\theta_i = \alpha(i-1)s^2 + \kappa_i s$ and δ_i be defined by (Eq. 5.12). Then the G^1 continuous arc spline defined by $P_A, \theta_A, s+\delta_1, \Delta\theta_1, \dots, s+\delta_n, \Delta\theta_n$ interpolates the two point-orientation pairs.*

5.3.2 Optimal arc-length and initial curvature

We have seen that the perturbations are actually functions of s and κ_1 . Now we want to find the optimal s and κ_1 such that the sum of squares of δ_i , $\sum_{i=1}^n \delta_i^2$, is minimized. In the ideal situation, all $\delta_i = 0$ and the curve is an EAS.

5.3.2.1 Bound estimation on parameters

First, the length of the arc spline curve must be greater than the length of line segment connecting P_A and P_B , which implies

$$s \geq \frac{\|P_A P_B\|}{n}. \quad (\text{Eq. 5.14})$$

Second, substituting (Eq. 5.6) into (Eq. 5.5) gives

$$-2\pi \leq \frac{\theta_B - \theta_A}{n} - \kappa_1 s \leq 2\pi. \quad (\text{Eq. 5.15})$$

Combining this with inequality (Eq. 5.4) leads to the following bounds on $\kappa_1 s$:

$$\max(-2\pi, -2\pi + \frac{\theta_B - \theta_A}{n}) \leq \kappa_1 s \leq \min(2\pi, 2\pi + \frac{\theta_B - \theta_A}{n}). \quad (\text{Eq. 5.16})$$

Third, the combination of (Eq. 5.15) with (Eq. 5.4) also gives $-4\pi \leq \frac{\theta_B - \theta_A}{n} \leq 4\pi$.

Thus

$$n \geq \frac{|\theta_B - \theta_A|}{4\pi}. \quad (\text{Eq. 5.17})$$

5.3.2.2 Levenberg-Marquardt solver

We introduce variable transformation, $\eta = \kappa_1 s$. Without ambiguity, we denote δ_i of (Eq. 5.12) by $\delta_i(s, \eta)$. Let $\mathbf{f}(s, \eta) = (\delta_1(s, \eta), \dots, \delta_n(s, \eta))^T : \mathcal{R}^2 \rightarrow \mathcal{R}^n$ be a vector function. Then we formulate our problem as a minimization problem:

$$\min_{(s, \eta) \in \Omega} \|\mathbf{f}(s, \eta)\|^2 = \min_{(s, \eta) \in \Omega} \sum_{i=1}^n \delta_i^2(s, \eta)$$

where $\Omega = \{(s, \eta) : s \geq \frac{\|P_A P_B\|}{n}, \max(-2\pi, -2\pi + \frac{\theta_B - \theta_A}{n}) \leq \eta \leq \min(2\pi, 2\pi + \frac{\theta_B - \theta_A}{n})\}$.

This is a non-linear least square problem. We use the Levenberg-Marquardt (LM) algorithm to find the solution. The LM algorithm is one of the most widely used optimization algorithms [59, 65]. It uses an effective damping strategy that makes it be able to converge quickly from a wider range of initial guesses.

The LM algorithm is an iterative procedure. Starting from an initial guess for (s, η) , each iteration step replaces the current (s, η) by a new estimate $(s, \eta) + (\Delta s, \Delta \eta)$, where $(\Delta s, \Delta \eta)$ are the solution to a linear system

$$(\mathbf{J}^T \mathbf{J} + \mu \text{diag}(\mathbf{J}^T \mathbf{J})) \begin{pmatrix} \Delta s \\ \Delta \eta \end{pmatrix} = -\mathbf{J}^T \mathbf{f}(s, \eta)$$

with the Jacobian matrix

$$\mathbf{J} = \begin{pmatrix} \frac{\partial \delta_1}{\partial s} & \frac{\partial \delta_1}{\partial \eta} \\ \vdots & \vdots \\ \frac{\partial \delta_n}{\partial s} & \frac{\partial \delta_n}{\partial \eta} \end{pmatrix},$$

the diagonal matrix $\text{diag}(\mathbf{J}^T \mathbf{J})$ consisting of the diagonal elements of $\mathbf{J}^T \mathbf{J}$, and the damping factor μ . The damping factor is adjusted at each iteration. If reduction of the objective function is rapid, a smaller value is used for μ ; Otherwise, if an iteration gives insufficient reduction, μ is increased [63]. The pseudocode of this optimization is given in Algorithm 1, which is adapted from [63]. Here we heuristically set the initial estimation:

$s = \frac{1.5\|P_A P_B\|}{n}$ and $\eta = 0$.

Algorithm 1 (LM-Solver)

Input: A vector function $\mathbf{f}(s, \eta) = (\delta_1(s, \eta), \dots, \delta_n(s, \eta))$ and an initial parameter estimation (s, η) .

Output: Optimal (s, η) .

Algorithm:

$\nu \leftarrow 2, \mu \leftarrow 10^{-3}$

$\epsilon_1 \leftarrow 10^{-10}, \epsilon_2 \leftarrow 10^{-10}, k_{max} = 100$

$stop \leftarrow false$

$k \leftarrow 0$

$A \leftarrow \mathbf{J}^T \mathbf{J} + \mu \text{diag}(\mathbf{J}^T \mathbf{J}), G \leftarrow -\mathbf{J}^T \mathbf{f}$

while $((stop == false) \text{ or } (k < k_{max}))$ **do**

$k \leftarrow k + 1$

repeat

 Solve the equation: $(A + \mu \text{diag}(\mathbf{J}^T \mathbf{J}))(\Delta s, \Delta \eta)^T = G$

if $(\|(\Delta s, \Delta \eta)\|^2 \leq \epsilon_1 \|(s, \eta)\|^2)$ **then**

$stop \leftarrow true$

else

 Find a positive h such that $(s, \eta) + h(\Delta s, \Delta \eta) \in \Omega$

$(s_{new}, \eta_{new}) \leftarrow (s, \eta) + \min(1, h)(\Delta s, \Delta \eta),$

$\mathbf{f}_{new} \leftarrow \mathbf{f}(s_{new}, \eta_{new}),$

$\mathbf{J}_{new} \leftarrow \mathbf{J}(s_{new}, \eta_{new})$

$d = \frac{\|\mathbf{f}\|^2 - \|\mathbf{f}_{new}\|^2}{(\Delta s, \Delta \eta)(\mu \text{diag}(\mathbf{J}^T \mathbf{J})(\Delta s, \Delta \eta)^T + G)}$

if $(\|\mathbf{f}_{new}\|^2 < \|\mathbf{f}\|^2)$ **then**

$(s, \eta) \leftarrow (s_{new}, \eta_{new}), \mathbf{f} \leftarrow \mathbf{f}_{new}, \mathbf{J} \leftarrow \mathbf{J}_{new}$

$A \leftarrow (\mathbf{J}^T \mathbf{J}), G \leftarrow (-\mathbf{J}^T \mathbf{f})$

$stop \leftarrow ((\|G\|_\infty \leq \epsilon_1) \text{ or } (\|\mathbf{f}\|^2 \leq \epsilon_2))$

$\mu \leftarrow \mu \max\{\frac{1}{3}, 1 - (2d - 1)^3\}$

$\nu \leftarrow 2$

else

$\mu \leftarrow \mu \nu$

$\nu \leftarrow 2\nu$

end if

end if

until $(d > 0) \text{ or } (stop)$

end while

5.3.3 Algorithm

Now we are ready to summarize the shape completion algorithm. Given two point-orientation pairs (P_A, θ_A) and (P_B, θ_B) as input, the algorithm proceeds as follows:

step 1. Use Algorithm 1 to find the optimal arc-length s and initial curvature κ_1 .

step 2. Use Eq.(Eq. 5.6) to compute α .

step 3. Use Eq.(Eq. 5.12) to compute δ_k .

step 4. If all δ_k are zero, we obtain an Euler arc curve defined by $P_A, \theta_A, n, s, \alpha, \kappa_1$.

In case not all δ_k are zero (probably due to the behavior of the numerical solver), we modify the Euler arc curve to a G^1 arc spline, which is defined by $P_A, \theta_A, s + \delta_1, \Delta\theta_1, \dots, s + \delta_n, \Delta\theta_n$ where $\Delta\theta_i = \alpha s^2(i-1) + \kappa_1 s$. In this way, the output curve is guaranteed to interpolate the two point-orientation pairs.

Note that in the above procedure, the user is required to specify the number of arcs. In practice, we can let the algorithm automatically determine it. The basic idea is as follows: First, an initial number is given as the number of arcs. We compute the Euler arc spline interpolating the given point-orientation pairs. Second, we increase the number of arcs by a constant and re-compute the Euler arc spline. Third, we compare the distance between the two Euler arc spline curves. Since the two curves may have different arc-lengths, we evenly sample both curves, compute the distance between two corresponding points, and use the maximum distance as an upper bound on the distance of the two curves. If the distance bound is greater than a given tolerance, we go back to the second step to increase the number of arcs. This process continues until the distance bound is smaller than the tolerance. Then we output the curve with the larger number of arcs.

5.4 Experiments and Discussions

We have implemented our algorithm using C++. This section provides experimental results to validate the algorithm.

We first show that when the number of arcs increases, Euler arc curves converge to the Euler curves, which complies the results found in [66]. Figure 5.6 shows three Euler curves (in pink color) and their approximation by Euler arc splines (in gray) with different numbers of arcs. All the Euler arc spline curves interpolate the boundary conditions. The statistics of the length of the Euler curves and the approximation errors is given in Table 5.1. The approximation error is computed by evenly sampling the curves and computing the distance between the corresponding points, and we use the maximum distance as the error. It can be seen that the errors drop quickly as n increases.

Figure 5.6: Examples of Euler curves in pink color and a series of Euler arc splines shown in gray whose numbers of arcs are 10, 20, 30 and 40.

Table 5.1: Statistics for approximate errors.

curve	length of the Euler curve	maximum error between the Euler curve and the Euler arc spline of n arcs			
		$n = 10$	$n = 20$	$n = 30$	$n = 40$
Figure 5.6(a)	597	37.7	12.1	3.9	0.88
Figure 5.6(b)	569	33.1	8.6	1.7	0.75
Figure 5.6(c)	445	26.5	10.9	4.9	0.82

Paper [53] has revealed that completion curves using Euler curves are intuitive and natural. Thus our Euler arc splines are also expected to provide intuitive and natural curve completion. Figures 2.1(b) and (d) show the results of shape completion using our proposed method. Figure 5.7 demonstrates our method with a variety of point-orientation configurations. In Figure 5.7(a), the start angle is $\theta_A = -\frac{\pi}{2}$ and the end angle θ_B changes around from π to $-\pi$. In Figure 5.7(b), $\theta_A = -\frac{\pi}{2}$ with θ_B varying around from $-\pi$ to -2π .

Figure 5.7: The shape completion for various point-orientation configurations.

We also implement the methods of [53] for comparison. In the following experiments, the starting point and its orientation are shown in red colors and the ending point and its the orientation are shown in blue. Figure 5.8 show the results of shape completion using Kimia *et al.*'s method and our method. It is observed that Kimia *et al.*'s method does not produce an Euler curve that interpolates the ending point while our algorithm guarantees the interpolation of both endpoints. Figure 5.9 shows that with the same tangent vectors at the two endpoints we can set different orientation angle values to achieve different completions, which may be useful in practice. Note that Walton and Meek's method [97] has a limitation on the angle range.

Figure 5.8: Comparison of our method and Kimia *et al.*'s method. The curves generated by our method is shown in pink color and the curves generated by Kimia *et al.*'s method is shown in black.

A straightforward approach to shape completion is to use cubic Hermite interpolation. Although the Hermite interpolation is simple to construct and compute, it does not always provide satisfactory results as analyzed and shown in [39] which focuses on 3D curves. Here, we take their figures as 2D images for our examples in order to show that our curves have pretty nice shapes compared with Hermite curves in Figure 5.10. Further, we illustrate curvature variations of our proposed curves and the Hermite curves in Figure 5.11. It can be seen that curvatures of the EAS curves are changed monotonically. Thus an circular shape can be guaranteed in an EAS curve, where curvatures are same along the curve shown in the top right figure.

Finally, the use of Euler arc splines for shape completion is illustrated in Figure 5.12, where the constructed Euler arc splines are overlayed on synthetic occluders.

Figure 5.9: Shape completion for the same tangent vectors but different angle values.

Figure 5.10: Shape completion with our proposed curves (pink) and the Hermite curves (green). The scale of the Hermite splines is determined manually.

Figure 5.11: Curvature illustrations of the EAS curves (pink) and the Hermite curves (green).

Figure 5.12: Shape completion using Euler arc spline curves.

Chapter 6

Patch-Based Image Completion

This chapter considers the image completion problem that is to fill holes of an image by propagating both structure and texture information. We present an effective patch-based filling algorithm, which improves the existing techniques by enhancing the two basic elements in the patch-based image completion process: (1) adaptive patch size and (2) novel criteria on similarities between two patches.

6.1 Motivation and Our Work

In the existing patch-based image completion algorithms, the patch size is usually fixed as a default number or specified by the user. For instance, Criminisi *et al.* choose a default window size of 9×9 pixels or let the user set it to be slightly larger than the largest distinguishable texture element in the source region [23]. However, it is difficult to choose a good patch size and it may not be appropriate to let all the patches have the same size. Moreover, it is known that the patch size has an important influence on the completion performance because it affects how well the filled patch captures the local characteristics of the source image. An inappropriate patch size may result in an unsatisfactory completion. Refer to a synthetical image in Figure 6.1, for example. The region marked by green color in the left figure is the unknown region. If the size of the yellow square is used for completion, the linear structure will not be propagated correctly

as shown in the bottom-right figure. Intuitively, the patch size should be smaller than or equal to the red one and after the propagation of the linear structure, the remainder region should be filled by patches with larger size for better capturing statistical characteristics. The result of completion in this way is shown in the top-right figure. Motivated by the above observation, this paper proposes an optimal algorithm to determine adaptive patch sizes for image completion. Different from Drori et al's approach [26] where the adaptive size is chosen to be inversely proportional to the spatial frequency of the image and to capture features of various scales, our approach considers not only the spatial frequency of the image but also the structure variance. With adaptive patch sizes determined by our algorithm, more appropriate patches can be selected for matching.

Figure 6.1: Different patch sizes for completion

We also find that most patch-based image completion algorithms use the sum of square of difference (SSD) of pixel colors in the overlapping part to judge whether a source patch is suitable for filling a target patch. Here we call a measurement that judges whether a source patch matches a target patch as *similarity measurement*. Refer to Figure 6.2 for example, where the black and red squares denote the target and matching source patches. With most existing similarity measurement, a source patch with abrupt color changes but with small differences in the overlapping part is chosen to fill the target

(a)

(b)

(c)

Figure 6.2: Different similarity measurement for completion.

region as shown in Figure 6.2(b), which however reduces the completion quality since human perception prefers to propagate colors and structures consistently or continuously without any abrupt color changes as shown in Figure 6.2(c). Thus we propose to add a consistence property of a patch to enhance the similarity measurement.

The main contributions of this work are: (1) we propose an algorithm to determine adaptive patch sizes for image completion; (2) we present a new similarity measurement that gives consideration of consistency inside a patch.

We first give an overview of the patch-based image completion technique in Section 6.2. Then, Section 6.3 describes our improvements of the completion algorithm in details. Finally, experiment and comparison results are presented in Section 6.4.

6.2 Overview of Patch-based Image Completion

The basic idea of patch-based image completion is to complete an image by gradually filling the fill front of the target region using patches from source regions, which have certain similarity with the patches along the fronts. In general, the patch-based image completion algorithm involves the following steps:

- (i) Determine patch's property
- (ii) Decide filling order in target regions
- (iii) Measure similarities between patches
- (iv) Complete a target patch with a matching source patch
- (v) Repeat steps (ii)-(iv) and update data until no target patch exists

Determining patch's property is to specify the shape, size and transformation information of a patch. In paper [26], circular patches with adaptive size are used. Moreover, patches can be transformed for comparison. Although patches of arbitrary shape can be used, circular and square patches are most popular due to their simplicity. In order to reduce the computational complexity and utilize source regions sufficiently, square patches with adaptive size are used in our algorithm.

Filling order is critical since it has high influences on the final completion result. The scheme proposed by paper [94] pays much attention to image structure information and generally achieves good propagation of both textures and linear structures.

When a target patch has been chosen for completion according to the filling order given in step 2, the next step is to find a matching source patch and then to propagate the target patch with the matching source patch. Therefore, the similarity measurement between the source patch and the target patch controls the searching result. Most

completion algorithms use the sum of square of difference (SSD) of pixel colors in the overlapping part to define the distance between two patches. Nevertheless, as we observe, the color gradients are also indispensable in this comparison procedure.

After that the most matched source patch has been selected, we fill the unknown part of the target patch with the corresponding part of the source patch.

In general, there are two types of implementation for patch-based completion methods: greedy growing and global optimization. Greedy growing is just a local optimal algorithm without global consideration, which usually leads to some visual inconsistency. Formulating the patch-based filling as a graph labeling problem that finds an optimal labeling solution minimizing an energy function is the concept of processing this problem with global optimization. Recently, the Belief Propagation (BP) algorithm is frequently used to solve this kind of optimization problem. However, the algorithm is time-consuming. For an image with N target patches and K source patches, the run time of BP is $O(NK^2)$ [48]. Although a coarse-to-fine scheme is introduced to reduce the computational complexity of the standard BP algorithm in paper [48], a new challenge of how to specify the parameters $K1$ for K -means algorithm and $K2$ for the maximum number arises. On the other hand, for the greedy growing way, the computational complexity is $O(NK)$. Thus our filling algorithm in this chapter is implemented based on the greedy mechanism.

6.3 Improvements of the Completion Algorithm

In this section, we describe a greedy patch-based image completion algorithm, which follows the main framework proposed in previous work [26, 23, 101, 21], with enhancements in the basic components: adaptive patch size and similarity measurement.

6.3.1 Adaptive patch size

Given a source region of an input image, our goal is to generate adaptive source patches of different sizes, using which a patch-based image completion algorithm can fill the target region to achieve an efficient and high-quality completion. Intuitively, we should choose a small patch in an area that covers many distinct structures and a big patch in an area with few distinct structures. To illustrate this idea, in this work we choose the square as the patch for simplicity. For a source pixel P , if the square of $k \times k$ pixels centered at P lies in the source region (where k is an odd number), it is a patch of P with size k . Let $S = P$ denote the set of all source pixels P and $SQ(P, k)$ denote the square of size k centered at P . For a pixel P , once the size k is given, the square is specified. Thus the problem is to determine the size of square for each source pixel so that the square is optimal in some measurement.

First, the optimal square should be as large as possible with the smallest structure changes inside. To distinguish structures with various directions, a gradient angle histogram is built based on the image gradients over a square. For each gradient vector, its magnitude m_θ corresponds to its directional angle θ truncated to an integer degree. A gradient angle histogram records the tabulated sum of gradient magnitudes for each gradient angle (see Figure 6.3). The gradient angle histogram is an important tool for measuring variance of structures.

Traditional methods for computing gradients are sensitive to small color changes and then the histogram may be misled by trivial and repeated structures such as grass or trees. To overcome this problem, a saliency map that emphasizes distinct and homogenous features is introduced to improve the gradients for the gradient angle histogram. The saliency map is computed by a method given in [15]. After that, we multiply the saliency value to the corresponding gradient vector, generating an improved gradient field.

Figure 6.3: Gradient angle histogram of an image.

Now for each source pixel P , a sequence of gradient angle histogram over $SQ(P, k)$ with different sizes k are created based on the improved gradients. The more differences two histograms have, the less similarity the corresponding two patches have. Therefore a patch with the minimum structure changes is defined to be the one that has the largest gradient angle histogram difference with its adjacent patch of a larger size. For patch $SQ(P, k)$, its gradient angle histogram difference with $SQ(P, k + 2)$ is defined as:

$$HD(P, k) = \left(\frac{1}{(k+2)^2 - k^2} \right) \sum_{\theta=0}^{360} |H(P, k+2, \theta) - H(P, k, \theta)|, \quad (\text{Eq. 6.1})$$

where $H(P, k, \theta)$ is the ordinate of the histogram of $SQ(P, k)$ at θ .

Second, we observe that only based on the histogram difference, the optimal patch may be misled by some noises. The patch sizes should also be locally consistent, which can make the choice of the optimal patch not sensitive to the noises. Therefore we suggest that the optimal patch sizes for all the source pixels be the solution of the following minimization problem:

$$\min \sum_{P \in \mathcal{S}} \left(\frac{((k+2)^2 - k^2)}{HD(P, k(P))} + \lambda \left(k(P) - \frac{\sum_{Q \in N(P) \cap \mathcal{S}} k(Q)}{\|N(P) \cap \mathcal{S}\|} \right) \right), \quad (\text{Eq. 6.2})$$

where $k(P)$ is the patch size of pixel P , $N(P)$ is the 8-neighborhood of P , $\|R\|$ gives the number of pixels within R , and λ is a tradeoff factor (typical value is 1). The first

term in the above bracket measures structure variance in a patch and the second term measures the size consistency of the patch with its neighboring patches.

Finally, we need to solve the optimization problem (Eq. 6.2), which actually is not trivial. One way use simulated annealing to find the globally optimal solution, which is extremely time-consuming. Here we propose a simple optimization algorithm which consists of the following steps:

- (1) For each pixel P , find $k(P)$ that minimizes $\frac{((k+2)^2 - k^2)}{HD(P,k)}$.
- (2) For each pixel P , modify $k(P)$ and fix all the other patch sizes so that (Eq. 6.2) is minimized.
- (3) Repeat step (2) until there is no update.

Obviously, this is a local algorithm. Using the algorithm, we can quickly obtain a solution through sometimes it may be a locally optimal one. Figure 6.4 is an example that shows an image with the target region marked in green color and its corresponding patch size maps where high intensities indicate large sizes. Different from Drori et al.'s method [26], our method outputs a large patch for pixels on linear structures, which is shown by a red square in Figure 6.4(d).

6.3.2 Similarity measurement

Let us denote the color and gradient vectors at a pixel i by I_i and G_i . Refer to Figure 6.5 for illustration, where shows a target patch tp and a source patch sp . tp is partitioned into two parts $q1$ and $q2$ corresponding to the known and unknown regions respectively, and sp is divided into two parts $p1$ and $p2$ by the same partition trace. In previous methods [26, 23, 94, 55, 48, 33], the similarity between sp and tp is measured by the following distance:

$$D(tp, sp) = \sum_{i \in Known} ||I_i^{tp} - I_i^{sp}||^2, \quad (\text{Eq. 6.3})$$

Figure 6.4: Adaptive patch size: (a) an input image; (b) size map using Drori et al's method; (c) size map using our method; (d) a few patches based on our size map

which is to compare the color differences between the $q1$ and $p1$ parts. Here we propose to add a constraint on a source patch such that target regions are filled with consistent colors or structures. The constraint is inner consistency defined as:

$$c(sp) = ||\overline{G}_{p1} - \overline{G}_{p2}||^2 + ||\overline{C}_{p1} - \overline{C}_{p2}||^2, \quad (\text{Eq. 6.4})$$

where $\overline{(\cdot)}$ denotes the mean value of a part.

Figure 6.5: Similarity measurement between target and source patches.

If the scheme of adaptive patch size is chosen, the size of the target patch will vary according to each source patch. To avoid the trend of selecting patches with as small

size as possible, $D(tp, sp)$ should be divided by the number of known pixels in the target patch.

As a result, a matching source patch not only has similar colors with a target patch, but also has a consistent property inside. Take Figure 6.6 for an example. An input image with a hole marked by green color and a target patch marked by a black square is shown in Fig 6.6(a). Using Criminisi’s criterion, a source patch marked by red color, which has the minimum color difference with the target patch, is selected for filling (Figure 6.6(b)). In Figure 6.6(c), with our proposed similarity measurement, another source patch is chosen for filling, which obviously produces a more faithful completion result.

Figure 6.6: Image completion with different similarity measurements.

6.4 Experimental Results

We have implemented our algorithm and incorporated it into Criminisi *et al.*'s completion algorithm. To conduct experiments and comparison, we took various images from www.image.google.com and papers [26, 55]. These images are also served as the groundtruth. We randomly remove some regions in these images and then use the original Criminisi *et al.*'s algorithm and the algorithm augmented with our adaptive patch

size and similarity measurement to complete the holes. For objective evaluation, we use a peak signal-to-noise ratio (PSNR) to measure completion quality:

$$PSNR = 10 \log_{10} \left(\frac{255}{\frac{1}{M} \sum_{P \in T} |I_o(P) - I_c(P)|} \right), \quad (\text{Eq. 6.5})$$

where $I_o(P)$ and $I_c(P)$ are the values of pixel P in the original and the completed images, $|I_o(P) - I_c(P)|$ is the absolute difference in color space, T is the target region, and M is the number of target pixels. Figure 6.7 shows the PSNR values tested on 15 images, from which we can see that with our enhancement techniques, the filling results usually have a higher PSNR.

Figure 6.7: Completion results for 15 images using a fixed patch size and adaptive patch sizes

Figure 6.8 shows some completion results produced by our approach and other algorithms. It can be seen that our approach can enhance the visual quality of the completed image by correctly propagating structures and colors.

Finally, as mentioned in Chapter 3, performing shape completion before image completion can further improve our completion results, especially on preserving curvilinear features. Figure 6.9 shows such examples. Figure 6.9(a) are three images with occluders and their corresponding completed images by our proposed patch-based image completion. In Figure 6.9(b), we perform the shape completion first. The completion curves

Input image

Criminisi's method

Proposed method

Figure 6.8: More examples for comparison.

separate the objects from the backgrounds and split the occluders into two sub-regions. Then we apply image completion on two sub-regions separately to allow for proper filling-in of the sub-regions. In this way, the structure of the objects is well preserved and unnecessary diffusion is avoided in image inpainting, which is depicted in Figure 6.9(b).

(a) without shape completion

(b) with shape completion

Figure 6.9: Comparisons between image completion without and with shape completion.

Chapter 7

Curvilinear Feature Driven Image Vectorization Using Subdivision-Based Representation

This chapter considers image vector representation and the representation conversion algorithm. After an image is decomposed into multiple parts such as objects and background by the processes of segmentation, shape completion and image completion, they can be further processed by vectorization in order to produce an object-oriented and vector-based image representation.

7.1 Motivation and Our Work

Two typical representations for images are raster graphics and vector graphics. Raster graphics is a grid of pixels, with each pixel storing either a color/grey value or an index into a color palette. Pixel based image representation reveals its shortcomings for applications in many fields, such as image editing in movies. Vector graphics is not based on discrete points of pixels, but on the geometry of objects like curves. Unlike the raster representation, the vector representation is scalable, compact and resolution independent. It also facilitates many image editing processes. The process of converting a raster image into a vector representation is known as image vectorization.

Various vector representations and their associated vectorization algorithms have been proposed. The earlier vector graphics primitives are points, lines, curves, or polygons. They only support linear or radial color gradients rather than complex gradients, upon which images with simple colors are their objectives. For realistic full-color images, edge features and smooth color transitions are important. Rectangular mesh based graphics representations [79, 93, 57] were proposed to handle smoothly varying colors. Basically these representation use bicubic parametric surfaces to describe images. Similar techniques can also be found in commercial software such as Adobe Illustrator and CorelDraw. Although they have achieved success in many applications, they still have difficulty in representing images with complex color gradients. The challenge with these vector-based image representations lies in the fact that natural images usually have complicated curvilinear features. The complexity of curvilinear features has two-fold meanings. First, the curvilinear features have plenty shapes and orientations. However, the topological constraints imposed by these representations' mesh structure make them difficult to have a highly flexible spatial layout. Second, the curvilinear features often yield subtle shading effects. Due to the geometric continuity constraint, these quad mesh based representations are still too rigid to simulate the subtle color change effects accurately.

The above observation motives us to propose a new vector representation for images. The mathematical tool at the heart of the new representation is Loop subdivision scheme which applies to triangular meshes. The triangular meshes have flexible mesh structure and define shapes of arbitrary topology, and subdivision surfaces are smooth and compact. However, directly using Loop subdivision for image representation is not feasible for two main reasons: (1) Loop subdivision generates surfaces that are smooth everywhere while for images there often exists discontinuity across the curvilinear edges; and (2) There is no obvious connection between Loop subdivision surfaces and curvilinear features. The contributions of this work are as follows:

- We introduce a special cubic B-spline curve called restricted B-spline to model curvilinear features and we extend Loop subdivision by developing new subdivision rules to accurately model curvilinear features.
- We propose to use the developed subdivision surfaces in R^5 as a new image representation. Such representation can define images with piecewise-smoothly varying colors. We also introduce a way to model the shape edges and blur edges.
- We present an automatic algorithm to convert an image or an object in an image into the proposed vector representation. The algorithm is driven by curvilinear features.

Compared to previous work, the proposed representation and algorithm have the following advantages in addition to the common merits of vector representation such as editability and scalability: (1) they support flexible mesh topology and thus are able to handle images or objects with complicated features or boundaries effectively; (2) they are able to faithfully reconstruct curvilinear features, especially in modeling subtle shading effects around feature curves. The effectiveness of our proposed image vectorization method is confirmed by a large number of experimental results.

In Section 7.2 we first propose our vector representation for images. Then in Section 7.3 we present the associated vectorization algorithm. Experiments and discussions follow in Section 7.4.

7.2 Subdivision-Based Image Representation

We first introduce some notations for our descriptions. A point $v \in R^5$ is denoted by $v = (x, y, R, G, B)$, where the first two components (x, y) stand for the Cartesian coordinates in the xy -plane and the last three components (R, G, B) stand for RGB channels of color. Two projection operators LOC and $COLOR$ are defined. $LOC : (x, y, R, G, B) \rightarrow (x, y)$

projects each point in R^5 to its projection on the xy -plane by extracting its first two components. $COLOR : (x, y, R, G, B) \rightarrow (R, G, B)$ takes the color values from a point in R^5 by extracting the last three components. We also use LOC and $COLOR$ for curves, surfaces and point sets in R^5 to extract their respect dimensional components unambiguously.

Now we present our subdivision based image representation. An image may contain curvilinear features. We propose to use a special B-spline curve in R^5 to represent the curvilinear features (see Section 7.2.1) and a subdivision surface in R^5 to represent the image (see Section 7.2.2). The first two dimensions of the B-spline curves and the subdivision surface describe the shape or location of features and the image on the 2D plane; The last three dimensions specify the color of features and the image. We develop new subdivision rules to model images with piecewise-smoothly varying colors and their curvilinear features.

7.2.1 Curvilinear feature representation

A curvilinear edge is a smooth curve. We distinguish two cases and handle them differently.

The first case is that the curvilinear edge is a smooth closed curve. Then an uniform cubic periodic B-spline is used to represent it. Then use of B-spline representation is due to the fact that B-spline curves can be created using subdivision of their control polygons, which makes them easy to be included in subdivision surfaces. If we refine such a B-spline curve by inserting new knots evenly into the uniform knot sequence, a new control polygon consisting of new edge points and new vertex points is created. Each edge point is the midpoint of the corresponding edge; Each vertex point is the linear combination of the corresponding vertex and its two neighboring vertices with coefficients $6/8$, $1/8$ and $1/8$, respectively. This combination is also expressed by a mask: 1-6-1.

The second case is that the curvilinear edge is a smooth curve segment bounded by two endpoints. We represent such a segment by a curve uniquely determined by a polyline $P_0P_1\cdots P_n$. The underlying representation of the curve is actually a special cubic B-spline curve with control points $P_0, P_0, P_1, \dots, P_n, P_n$ and knot sequence $\{0, 0, 0, 0, 1, 2, \dots, n, n, n, n\}$, which we call the *restricted cubic B-spline curve* due to the duplicated of two endpoints. We can also refine the restricted B-spline curve by inserting a knot in the middle of each nonzero knot interval. Then the new knot sequence becomes $\{0, 0, 0, 0, 0.5, 1, 1.5, 2, \dots, n-1, n-0.5, n, n, n, n\}$ and the control points become $P_0, P_0, E_1, P'_1, E_2, P'_2, \dots, P'_{n-1}, E_n, P_n, P_n$ where

$$\begin{aligned} E_1 &= \frac{3P_0 + P_1}{4}, & E_n &= \frac{P_{n-1} + 3P_n}{4}, \\ E_i &= \frac{P_{i-1} + P_i}{2} & \text{for } i &= 2, \dots, n-1, \\ P'_1 &= \frac{3P_0 + 11P_1 + 2P_2}{16}, & P'_{n-1} &= \frac{2P_{n-2} + 11P_{n-1} + 3P_n}{16}, \\ P'_i &= \frac{P_{i-1} + 6P_i + P_{i+1}}{8} & \text{for } i &= 2, \dots, n-2. \end{aligned}$$

The above formulae are easily derived using B-spline knot insertion algorithms [22]. They can also be explained using masks. That is, the masks for E_1 , E_i , and E_n are 3-1, 1-1, and 1-3, respectively. There are two exceptional cases that have special masks:

- $n = 1$: The curve is a straight line segment. The mask for E_1 is 1-1.
- $n = 2$: The masks for E_1 and E_2 are 3-1 and 1-3, and the mask for P'_1 is 3-10-3.

One nice thing of this scheme is that the refinement of the restricted B-spline is equivalent to simply subdividing the initial polyline into the refined polyline $P_0E_1P'_1E_2\cdots E_nP_n$ (see Figure 7.1 for illustration). This subdivision has the same pattern as the refinement of an uniform cubic B-spline curve. That is, each edge corresponds to a new edge point and each old vertex corresponds to a new vertex point. However, the curve we define here is a non-uniform B-spline curve and it interpolates the two end points. Moreover, such specially designed curve formulation masks themselves can be easily integrated into the surface subdivision scheme introduced in the next section.

Figure 7.1: A restricted B-spline curve corresponds to an initial polyline (in green) and a refined polyline (in blue).

7.2.2 Subdivision surfaces

Consider a triangular mesh M in R^5 , which is a pair (V, K) where $V = (v_1, v_2, \dots, v_m)$, $v_i \in R^5$ is a set of vertices and K is a simplicial complex specifying the connectivity of the vertices. Triangular meshes have flexible topology and thus are convenient for our purposes.

A subdivision surface is defined by repeatedly refining an initial control mesh. Loop subdivision is one subdivision scheme applied to triangular meshes, which generalizes C^2 quartic triangular B-splines to arbitrary topology [62]. The refinement step of Loop subdivision proceeds by splitting each triangle into four subtriangles. The vertices of the refined mesh are weighted averages of the vertices in the unrefined mesh. They are classified into two classes: edge points and vertex points corresponding to edges and vertices of the unrefined mesh, respectively. As the refinement goes to infinity, Loop subdivision leads to a tangent plane continuous surface.

We aim to use a subdivision surface to define an image or an object in an image. The object could have an arbitrary boundary, holes, or many features which occur as sharp or semi-sharp curvilinear edges geometrically. An image can also be viewed as an object

with a regular boundary and possibly many features. Since color variances across the sharp or semi-sharp features are not smooth, we need to modify the Loop subdivision scheme to model objects with curvilinear features and boundaries.

Given an open triangular mesh, if there exist polylines consisting of edges of the mesh, which define B-spline curves for curvilinear features, they are called feature polylines. The related edges and vertices are tagged as *crease*. In particular, the boundary of the mesh is viewed as feature polylines. Feature polylines may contain endpoints. The endpoints are tagged as *corner*. In addition, a vertex lying on two feature polylines is also considered to be a corner vertex. In this way, the pair (V, K) becomes a tagged simplicial complex. The Loop subdivision masks are modified so that the feature polylines define the demanded curvilinear features and the tangent plane continuity across crease edges is relaxed. In the subdivision process, new edges corresponding to an old crease edge are tagged as crease, a vertex corresponding to an old crease vertex is tagged as crease, and similarly a vertex corresponding to an old corner vertex is tagged as corner.

Subdivision rules at crease edges and crease/corner vertices must be chosen carefully in order to match the rules for creating the curvilinear features given in Section 7.2.1. Figure 7.2 shows our edge subdivision rules and Figure 7.3 shows our vertex subdivision rules where $\alpha(n) = \frac{n(1 - a(n))}{a(n)}$ with $a(n) = \frac{5}{8} - \frac{(3 + 2 \cos(2\pi/n))^2}{64}$ and n is the valence of the vertex. The zeros in the crease subdivision masks completely decouple the behavior of the surface on one side of the crease from the behavior on the other side. Note that different subdivision rules have been presented in [44, 43] in order to create piecewise smooth subdivision surfaces. Our subdivision rules are driven by the curvilinear features. Feature polylines without corner vertices generate uniform cubic B-spline curves and feature polylines with some corner vertices generate cubic B-spline curves with interpolating endpoints.

Figure 7.2: Edge subdivision rules. Red lines denote crease edges and the disk denotes a corner vertex.

Figure 7.3: Vertex subdivision rules. Red lines denote crease edges and disks denote corner vertices.

7.2.2.1 Computing limit points

We look at a vertex v^k of the mesh after k rounds of refinement and its 1-ring neighbors $v_1^k, \dots, v_n^k$ where n is the valence of v^k . Let v^{k+1} be the vertex of the mesh after $k+1$ rounds of refinement, associated with v^k , and $v_1^{k+1}, \dots, v_{n+1}^{k+1}$ be the 1-ring neighbors of v^{k+1} . Then we have

$$(v^{k+1}, v_1^{k+1}, \dots, v_{n+1}^{k+1})^T = S_{n+1}(v^k, v_1^k, \dots, v_n^k)^T \quad (\text{Eq. 7.1})$$

where S_{n+1} is a $(n+1) \times (n+1)$ matrix called the *local refinement matrix*. The local refinement matrix is important in analyzing point positions and smoothness of the limit surface. We perform eigenanalysis of matrix S_{n+1} as in [43, 87] and derive the masks for computing the limit points. The masks depend on the type of the vertex and they are given in Figure 7.4 where $\omega(n) = \frac{3n}{8a(n)}$. Note that for a crease vertex with one or two corner neighbors, it will become a normal crease vertex after one round of refinement. Therefore the eigenanalysis should be performed after one round of refinement.

7.2.3 Modeling color discontinuity

Not that a subdivision surface is piecewise smooth, divided by curvilinear features across which the surface is only C^0 continuous. C^0 continuity is necessary for the geometry part (i.e., the first two dimensions). It ensures that no gaps occur geometrically in the representation. Nevertheless, the property of C^0 continuity may be too strong for the color part (i.e., the last three dimensions), since it is quite often for an image to have color discontinuity across edge features.

The introduction of crease edges into subdivision well solves the non-smoothness issue in geometric modeling, but it does not effectively solve the discontinuity problem for images. This motivates us to use *feature polyline pair* to model color discontinuity. That is, we should design our triangular mesh such that feature polylines occur in pairs and

Figure 7.4: Limit position masks for vertices. Red lines denote crease edges and disks denote corner vertices.

each pair of feature polylines is closed geometrically but have different color attributes. Moreover, we further introduce a parameter “sharpness” denoted by s , which takes values from 0 to 1, to describe how much the color is discontinuous across an edge feature. The sharpness value of 1 means that the edge feature is sharp. The value falling in $(0, 1)$ implies a semi-sharp edge feature. We then use the value of sharpness to determine the distance between the feature polyline pair. Specifically, we let the distance be w and $w = \rho(1 - s)$, where $\rho > 0$ is a constant. A typical value of ρ is 5 and it works well in our experiments. When sharpness is close to 1, the distance is nearly zero. Then, the color will have a sudden change from one curvilinear edge to another, which gives color discontinuity effect. Figure 7.5 illustrates this idea.

Figure 7.5: Color discontinuity modeling. Top: a sharp feature is created with sharpness = 0.9; Bottom: a semi-sharp feature is created with sharpness = 0.1.

Figure 7.6 shows two images created using the proposed representation from triangular meshes. Tagged edges are marked by red and green colors for sharp and semi-sharp features.

(a) mesh and image with semi-sharp edges

(b) mesh and image with sharp edges

Figure 7.6: Images with curvilinear edges and their meshes.

7.3 Image Vectorization Algorithm

In this section, we describe our algorithm for automatically generating subdivision-based representation for a given image. Our algorithm consists of four phases: feature extraction, initial mesh construction, color optimization and mesh refinement. First, image analysis is performed to extract distinct curvilinear features. Second, the extracted curvilinear features are approximated by cubic B-spline curves and an initial triangular mesh with tagged information is constructed. Third, the color value for each vertex of the mesh is computed through an optimization procedure. Fourth, approximation error is checked. If the error is greater than a prescribed threshold, the mesh is refined and the algorithm goes back to phase 3. Phase 3 and phase 4 are repeated until satisfactory approximation is reached. In this process, the mesh size and reconstruction quality should be trade off. The technical details of these phases are explained below.

7.3.1 Feature extraction

Curvilinear edges are the curves or lines consisting of pixels with large gradient. With an input image, we employ several classic image processing methods to extract curvilinear features. First, Canny detector [16] is used to detect edges, where the high and low thresholds are chosen as 100 and 40 for our most examples based on the implementation with the OpenCV library [76]. Second, for the detected edges, we estimate the degree of blur b for each pixel on the edges using the method of [30]. The blur scale is normalized to $[0, 1]$. Third, the Otsu algorithm is adapted to minimize the within-group variances on the blur scales so that the curvilinear features are partitioned into sharp and semi-sharp two groups. And then the blue scales are quantized, where all the sharp edges have blur values of 0 and all the semi-sharp edges keep their own blur values. Fourth, Kovessi's method [56] is implemented for edge cleaning and arrangements and is performed for sharp and semi-sharp edges independently. Particularly, all the edges are traced and are

regulated as a set of separate segments. Edge segment's connections are performed to form a long and relatively smooth curve. Short edges with a length smaller than 10 pixels are removed. Finally, for each detected curvilinear edge, we associate it with a value of sharpness s , where $s = (1 - b)$.

After edges are extracted, corners are to be identified. All the endpoints of curvilinear edges are corner. Besides, corners may exist in the middle of a curvilinear edge. For those corners, we assume that they are only in the sharp curvilinear edges with sufficient length such as greater than 50 pixels. We use a simple corner feature detection algorithm to find them. (The algorithm is to check an angle of each point along an edge, where the angle is formed by two lines that connect the point with its forward and backward points along the edge. Corners are located at points with quite small angles.) Figure 7.7 shows an example of feature extraction.

Figure 7.7: Feature extraction. (a) Input object; (b) Cleaned sharp (red) and semi-sharp (green) edges; (c) Some corners.

7.3.2 Initial mesh construction

Mesh construction involves three steps: curvilinear curves reconstruction, feature polyline pair creation and triangular mesh generation.

Curvilinear curve reconstruction is to fit B-spline curves to the curvilinear features obtained in Phase 1. For each curvilinear edge, depending on whether it is a closed curve or an open curve, we choose the type of B-spline curves: a cubic periodic B-spline or a restricted cubic B-spline. Based on the length of the curvilinear curve which is denoted by len , we heuristically let the number of control points be $\max \{4, len/8\}$. Once the type of the curve and the number of control points are determined, the only unknowns are the control points. The conventional least squares B-spline fitting is used to find the control points. If the approximation error is larger than an user-specified threshold, the number of control points is increased and the least squares procedure repeated.

As explained in Section 7.2.3, we need to use feature polyline pairs to model (semi-)sharp edges. The above B-spline fitting has already produced one polyline. Therefore we can create another one by offsetting the first one along the "normal" direction by a certain distance (see Figure 7.8). Specifically, for each control point, we define the normal at this point to be the average of the normals of the two edges incident to it. Then each control point is moved along its normal by a distance. The distance depends on the sharpness and is computed by the formula given in Section 7.2.3. The tags on points and edges are correspondingly copied. It is worth pointing out that using a pair of feature polylines is important for generating high quality reconstruction. Figure 7.9 shows such an example, where (a) is the input image, (b) is the reconstruction result using single curvilinear features marked by red in (d), and (c) is the result using polyline pairs marked by green in (e).

Now the generated polylines are used as an input and a constrained Delaunay triangulation [88] is performed to create a triangular mesh in the image domain. The triangulation quality can be controlled by angles of triangles. In order to obtain a satisfied mesh, additional points are allowed to be inserted, but not on the polylines. As a result, an initial control mesh is constructed augmented with some tag information.

Figure 7.8: The second feature polyline (in pink) is created from the first one (in red).

(a) input image

(b) using single polylines

(c) using polyline pairs

(d) mesh for (b)

(e) mesh for (c)

Figure 7.9: Effects of image reconstruction using only single polylines marked in red in (d) and using feature polyline pairs marked in green in (e).

7.3.3 Color optimization

Suppose the input image is I and the subdivision surface in R^5 is S . Now $(LOC(V), K)$, the projection of M onto the image plane, is fixed. We want to find $COLOR(V)$, the color values for vertices. We determine the vertex colors by solving the following minimization problem:

$$\min_{COLOR(V)} \sum_{(x,y)} \|I(x,y) - COLOR(H(x,y))\|^2 \quad (\text{Eq. 7.2})$$

where $H(x,y) \in R^5$ is a point on S with $LOC(H) = (x,y)$.

In practice, however, computing $H(x,y)$ for given (x,y) is very complicated. Here we take an approach similar to the one used in [44]. We subdivide the original mesh M r times (typically we set $r = 2$) to generate a refined mesh M^r using the rules proposed in Section 7.2.2 and then push all the vertices of M^r to their limit positions using the position masks. In this way, we obtain a piecewise linear approximation $\tilde{S}$ to S . Since each vertex of M^r is a linear combination of the vertices of M and the limit position is also a linear combination of M^r , each of the vertices of $\tilde{S}$ can be written as a linear combination of the vertices in V . Furthermore, since $\tilde{S}$ is piecewise linear, any point on $\tilde{S}$ can be linearly represented by at most three vertices of $\tilde{S}$. In particular, if a 2D point $q = (x,y)$ is within $LOC(\tilde{S})$, it must be contained in a triangle formed by vertices in $LOC(\tilde{S})$. Assume the three vertices are $LOC(\tilde{v}_i)$, $LOC(\tilde{v}_j)$, and $LOC(\tilde{v}_k)$. Then

$$q = u_i LOC(\tilde{v}_i) + u_j LOC(\tilde{v}_j) + u_k LOC(\tilde{v}_k)$$

where (u_i, u_j, u_k) is the barycentric coordinates of q with respect to the triangle $LOC(\tilde{v}_i) LOC(\tilde{v}_j) LOC(\tilde{v}_k)$. If H is a point on $\tilde{S}$ satisfying $LOC(H) = q$, the linear relationship also holds for H in R^5 , meaning $H = u_i \tilde{v}_i + u_j \tilde{v}_j + u_k \tilde{v}_k$. Thus each point on $\tilde{S}$ —not just the vertices—can be written as a linear combination of the vertices in V . That is $H = C_q V$ where C_q is a row vector whose entries is a combination of the effects of r -fold subdivision followed by applications of position masks and barycentric coordinates.

The minimization problem Eq. 7.2 now becomes

$$\min_{COLOR(V)} \sum_{(x,y) \in LOC(\tilde{S})} \|I(x,y) - C_{(x,y)}COLOR(V)\|^2 \quad (\text{Eq. 7.3})$$

That is a typical least squares problem, which can be efficiently solved by linear solvers.

To sample (x, y) within $LOC(\tilde{S})$, we use scan conversion to each triangle of $LOC(\tilde{S})$. The coherence of location enables us to compute the barycentric coordinates of a point from the barycentric coordinates of the previous point or the point on the previous scan line by simple additions or subtractions. This speeds up the computation.

7.3.4 Mesh refinement

After color optimization, the optimal subdivision surface for the image reconstruction is obtained. The fitting error is then evaluated and rasterized to be a reconstruction error map. To reduce the fitting error, we proceed to sample the error map adaptively in order to refine the mesh with those sample points.

We develop an effective sampling technique, by which areas with large reconstruction errors are sampled densely, otherwise sampled sparsely. The main idea is to discriminate different error levels and select significant sampling points at each level. First, the error map is quantified into several sub-maps corresponding to different error spans, such as 3 in our examples. Second, they are processed as a Gaussian pyramid with three levels. Third, sampling with an adaptive size is performed on each level. Sampling the error map by this way produces a smaller number of points and captures more important points. Finally, we select the sample points whose distance to the existing mesh vertices or tagged control polylines are all greater than a given threshold. Taking Figure 7.10 for example, the first row shows the quantification results, and the second row shows the sample points by dots. Figure 7.10(e) shows the Gaussian pyramid.

After sampling, instead of refining the existing mesh by directly adding the new vertices corresponding to the selected sample points, we apply the constrained Delaunay

Figure 7.10: Adaptive sampling. (a): an error map; (b)-(d): the three quantified sub-maps; (e): the Gaussian pyramid using a Gaussian filter one, two, and four times with sigma equal to 2.0 from bottom to top; (f)-(h): interim sampling results with sizes of 2, 4 and 16 pixels respectively; (i): the final sampling result of (a).

triangulation of Phase 2 again to the vertices of the existing mesh, the new vertices, and the feature polylines. This usually gives a better quality mesh. Figure 7.11 shows one example of such refinement.

Figure 7.11: An example of mesh refinement.

7.4 Experimental Results and Discussions

We have implemented our vectorization method using C++. This section provides experimental results on various images to validate our proposed method. The experiments are run on a PC with Intel Xeon 2.0 GHz CPU and 2GB RAM. The reconstruction quality is not only evaluated by human vision but also quantified by quality measurements.

Our algorithm handles a whole image and an object on an image in the same way, except for a preprocess step for objects that is to cut out objects from images using some segmentation tool. For example, Figures 7.12(a), (c), Figure 7.13(b), Figure 7.16, and Figure 7.17 are all objects.

Figures 7.12-7.17 show images with various characteristics. Figure 7.12(c) and Figure 7.13(b) have complicated topology. They are two objects with many holes. Figure 7.12(b) and Figure 7.15 have complicated curvilinear features. Figure 7.12(a) has smooth color transitions. It can be seen that for all these examples, our vectorization results are very close to the original images visually. Refer to Table 7.1 for various statistics data of our experimental results. It can be found that our reconstructed mesh is compact. Meanwhile our algorithm is efficient. It took about 1.5 minutes to reconstruct each of these examples except for the durian model that cost 5 minutes.

Our method associates curvilinear features with sharpness. Sharpness enables our method to effectively handle subtle color changes, which is important for sincere reconstruction. Figure 7.15 shows how the puppet's shadow is reconstructed faithfully. The curvilinear edges of the image is detected as marked in Figure 7.15(b) where red indicates a large sharpness value and green indicates a small sharpness value. Figure 7.15(c) and (d) are the results of vectorization with and without using sharpness, respectively. The close-ups in Figure 7.15(e) show the difference.

We also compare our vectorization with two up-to-date methods in the literature. The objects in Figure 7.16 have fine details. As reported in [57], the method of [57]

(a) image with smoothly varying color

(b) image with curvilinear features

(c) image with complicated topology

Figure 7.12: Vectorization results using our method. Left: input images; Middle: reconstructed meshes; Right: reconstructed images.

(a)

(b)

Figure 7.13: More vectorization results: original images (left) vs reconstructed results (right).

(a)

(b)

Figure 7.14: More vectorization results (cont).

Figure 7.15: Vectorization with/without use of sharpness.

has difficulty in recovering the durian while our method can reconstruct the durian in quality with a reasonable mesh size. For the jade, our method produces a mesh with 1474 vertices, 5496 triangles, and 478 tagged edges, and mean error of 2.41, while the method of [57] produces a gradient mesh with about 63 vertical lines and 56 horizontal lines and mean error of 2.76. The images in the figure show that our reconstruction result contains more details. Figure 7.17 is the comparison of our method with the one of [102]. Our method results in 3417 vertices, 6257 faces, 1586 tagged edges and mean error of 0.85 while the method of [102] produces 380 patches with each patch containing 66 parameters and mean error of 0.98.

7.4.1 Quality assessment

We evaluate our reconstruction quality using three popular measurements: mean reconstruction error (ME), Peak Signal-to-Noise Ratio (PSNR) and mean Structural SIMilarity index (MSSIM). Particularly, SSIM has excellent performance relative to human subjective assessment [75].

Based on a curve reflecting the relationship between MSSIM/PSNR and Mean Opinion Scores (MOS) in [99], a test image with PSNR greater than 40 or MSSIM greater than 0.960 obtains a MOS higher than 60. The MOS is divided into five equal regions within [0,100] and these five regions are marked with adjectives “Bad”, “Poor”, “Fair”, “Good” and “Excellent”. Referring to Table 7.1, we can see that all our testing results have “good” or “excellent” reconstructions.

7.4.2 Applications

With our vector graphics representation, standard graphics operations such as magnification and rotation images can be performed easily and faithfully. Below we show two other processes.

(a)

(b)

Figure 7.16: Left: result of [57]; Right: ours.

Figure 7.17: Left: result of [102]; Right: ours.

Table 7.1: Image vectorization results

Examples	#Vertices	ME	PSNR	MSSIM
Fig. 7.12(a)	192	1.28	47.06	0.9806
Fig. 7.12(b)	3468	0.94	48.38	0.9972
Fig. 7.12(c)	1474	2.41	44.30	0.9638
Fig. 7.13(a)	3289	0.96	48.32	0.9905
Fig. 7.13(b)	6805	2.13	44.84	0.9727
Fig. 7.14(a)	6981	1.62	46.02	0.9823
Fig. 7.14(b)	3842	1.07	47.87	0.9903
Fig. 7.15(c)	1259	1.38	46.71	0.9917
Fig. 7.15(d)	1259	1.40	46.64	0.9898
Fig. 7.16(a)	15888	2.09	44.96	0.9770
Fig. 7.17	3417	0.85	48.82	0.9901
Fig. 7.18	475	1.14	47.54	0.9924

(a)

(b)

(c)

(d)

Figure 7.18: Editing of image features.

Vector-based editing. The image can be modified by changing control points: 2D positions on the image plane or RGB color values. A tool has been implemented to allow the user to select individual points or edges, to sketch a closed curve to select a region, and then to change them. Figure 7.18(a) and (c) are an input image and the reconstructed image. The user selects some control points that define the curvilinear features in the middle of the cup and move them to the new locations to warp the shape as shown in Figure 7.18(b). The cup color can also be retouched by changing the color values of the control points. Moreover, our vector graphics representation provides a natural way to adjust sharpness of curvilinear features. It is achieved by changing the tag of edges. Figure 7.18(d) shows the result of changing part of a curvilinear edge from blur to sharp, which is enclosed by an ellipse.

Curvilinear feature driven image creation. Our proposed technique actually also provides an intuitive method for creating images from sketches, which is in the favor of artists. For example, the user just sketches some curves and assign sharpness and color attributes to them. These curves are used as curvilinear features and a tagged triangular mesh could be generated using the method given in Section 7.3.2. Then an image is created. Figure 7.19 demonstrates such a process.

Figure 7.19: Image creation from sketches: (a). The user sketches curvilinear edges (sharp and semi-sharp features are marked in red and green); (b) A tagged control mesh is automatically created; (c) The user paints color to vertices; (d) An image is created using the proposed subdivision surface.

Chapter 8

Conclusions

8.1 Summary

This research studies fundamental image representations and algorithms for processing image contents. It aims to develop techniques and tools for image editing that allow users to effectively create and manipulate images. To this end, object-oriented and vector-based techniques are sought. Vision and graphics techniques are employed to guide the better and high level manipulation of images. An object-oriented and vector-based image editing framework is proposed, which consists of two modules: image decomposition and vectorization to understand and represent images. In fact, four problems need to be solved towards building such a framework: image segmentation, shape completion, image completion and vectorization. We have investigated into these problems and fulfilled the goals of our research.

For image segmentation, we have proposed several technical components to enhance graph cut based interactive image segmentation. In particular, we integrate color and texture to form an augmented image for segmentation and enhance the use of local geometric structures of images in the graph cut based segmentation framework by constructing structure tensors. We propose to adapt the active contour as a postprocess for the graph cut and design an interface tool to allow the user to refine unsatisfactory

segmentation. The integration of these components promises an interactive image segmentation algorithm that can well segment various images even containing textures or with low-contrast boundaries, produces smooth contour aligning with the real boundary, and is robust to user inputs. The experiments have demonstrated the effectiveness of the proposed algorithm.

For shape completion, we have proposed a method to complete the shape with a geometrical continuous curve. In particular, we introduce a special arc spline called an Euler arc spline (EAS) as the basic form for visually pleasing completion curves, which has several properties desired by aesthetics of curves, in addition to computational simplicity and NURBS representation. We also develop an associated shape completion algorithm that is to find an optimal EAS interpolating the boundary conditions, i.e. the positions and orientations. Experimental results show that our method has a better convergence and produces desirable shapes.

For image completion, we have proposed a patch based image completion algorithm with adaptive patch sizes. The patch size determination is formulated as an optimization problem that minimizes an objective function involving image gradients and distinct and homogeneous features. Our patch sizes can capture the local characteristics of the source image and thus improve the final completion accuracy. Moreover, we observe that to find a source patch matching a target patch only by color difference is not sufficient. Thus we add the consistency of the source patch into the similarity measurement. Experimental results show that our method can provide a significant enhancement to patch-based image completion algorithms.

For image vectorization, we have proposed a subdivision-based vector graphics representation and its vectorization algorithm for raster images or objects in the images. The representation is in a form of triangular mesh augmented with color attributes at vertices and feature attribute at edges. The Loop subdivision scheme is modified and

applied to the mesh and the color attribute to define piecewise-smoothly varying images. The algorithm is developed to automatically convert a raster image or object into this representation. It is driven by curvilinear edges, which are first detected and then reconstructed using cubic B-spline curves, and generates the representation by constructing a triangulation based on the image boundary, the control polygons of cubic B-spline curves created for curvilinear edges, and feature attributes, and minimizing the color difference between the reconstructed representation and the original image. Compared to existing image vectorization methods, the proposed method can represent an image or objects more faithfully, especially in modeling subtle shading effects around feature curves, and can handle images or objects with complicated boundaries or feature more effectively. Many experiment examples have confirmed the effectiveness of the proposed method.

8.2 Future Work

Our segmentation method focuses on extracting an object from complex background with a correct and smooth boundary contour. It may not provide user an instant feedback of segmentation. Thus one direction in future is to reduce the runtime. In particular, the complexity of the proposed method mainly lies in solving the min-cut/max-flow [12] in a graph. The min-cut/max-flow algorithm works iteratively and may have a slow convergence. To improve it, some work has been proposed. For example, in [38] adaptive criteria have been proposed to determine when the convergence happens. The criteria can also be used in our method, but this improvement is still limited. If we further implement our algorithm on the graphics processing unit (GPU), the response time can be reduced dramatically.

In our shape completion method, though we always find a solution for all the inputs we tried in our experiments, theoretical analysis of existence of an Euler arc spline for

given point-orientation pairs remains an open question, which is worth further investigation. The analysis of uniqueness of the solution is interesting as well. Besides, it is an interesting and challenging problem to complete an occluded portion that has important features such as corners or cusps. Some intelligence is required in this creative completion processing, which can be considered from two aspects: (1) machine learning techniques and (2) human vision.

For the image completion, though many image completion methods appear, little work has been done on the evaluation of color filling quality. The comparisons of different completion methods are most based on users' vision. Thus, it will be essential and useful to investigate evaluation of the completion results.

Our vectorization method faces a challenge of handling complex textures such as the durian example in Figure 7.16. Although we can reconstruct the durian with high quality, the mesh size is too large for further editing. A texture mapping technique [7] can be used for describing textures. Procedurally generating the desired textures such as using Gabor noise [50], is suggested to achieve an improved representation, in [78], which is an interesting research topic: a vector based representation for textures.

Moreover, future work on performing various image editing based on our proposed framework will be quite interesting, taking image morphing for an example. Morphing is a process that gradually transforms one image into another. Many morphing algorithms have been developed. For example, the paper [103] proposed to use a thin-plate spline to create morphing [10], which could produce an animation of an animal from images. Note that most existing morphing algorithms are based on pixel-based representations and they have difficulty in handling occluded shapes. Within our proposed framework, it is possible to develop more flexible morphing techniques which can handle more complicated shapes and produce more interesting visual effects.

Last but at least, it is interesting to extend our framework from image editing to video processing. An object oriented and vector based video editing will have a bright future

especially in film making and computer-assisted animation. To support video editing, the corresponding object selection, shape completion, hole filling, and vectorization techniques need to be extended applicable to a sequence of video frames. The extension is not straightforward, since the video editing is quite different from editing a single image, where it is essential to make operations consistent in temporal sequence. For example, if we select an object in a frame, the object existing in the subsequent frames should be selected correspondingly, as well as warping, moving, or deleting an object. Thus, it is important to design a feasible interface to efficiently acquire user feedback.

References

- [1] Berkeley segmentation data set. Available from: <http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/>.
- [2] Brodatz data set. Available from: http://perso.telecom-paristech.fr/~xia/invariant_texture/invariant_texture_brodatz/Brodatz_re.html.
- [3] Msrc ground true data set. Available from: <http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm>.
- [4] M. Ashikhmin. Synthesizing natural textures. In *SI3D*, pages 217–226, 2001.
- [5] X. Bai and G. Sapiro. A geodesic framework for fast interactive image and video segmentation and matting. In *Proceeding of IEEE International Conference Computer Vision*, pages 1–8, 2007.
- [6] D. Ballard and C. Brown. *Computer Vision*. Prentice-Hall Inc. Englewood Cliffs, New Jersey, 1982.
- [7] W. Barrett and A. Cheney. Object-based image editing. *ACM Transactions on Graphics*, 21(3):777–784, 2002.
- [8] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *SIGGRAPH 2000*, pages 417–424. ACM Press/Addison-Wesley Publishing Co.
- [9] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr. Interactive image segmentation using an adaptive GMMRF model. In *Proceeding of European Conference Computer Vision*, pages 428–441, 2004.

REFERENCES

- [10] F. Bookstein. Principal warps: thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567–585, 1989.
- [11] Y. Boykov and M. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *Proceeding of IEEE International Conference Computer Vision*, pages 105–112, 2001.
- [12] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 26(69):1124–1137, 2004.
- [13] M. Brady and W. Grimson. Shape encoding and subjective contours. In *Proc. 1st Annual National Conference on Artificial Intelligence.*, pages 15–17, 1980.
- [14] X. Bresson, S. Esedoglu, P. Vandergheynst, J. Thiran, and S. Osher. Fast global minimization of the active contour/snake model. *Journal of Mathematical Imaging and Vision*, 28(2):151–167, 2007.
- [15] N. Butko, L. Zhang, G. Cottrell, and J. Movellan. Visual saliency model for robot cameras. In *Proceedings of International Conference on Robotics and Automation*, 2008.
- [16] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
- [17] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22:61–79, 1997.
- [18] T. Chan and J. Shen. Non-texture inpainting by Curvature-Driven Diffusions (cdd). *Journal of Visual Communication and Image Representation*, 12:436–449, 2000.
- [19] T. Chan and L. Vese. Active contour without edges. *IEEE Transactions on Image Processing*, 10:266–277, 2001.
- [20] T. Chang and C. Kuo. Texture analysis and classification with tree-structured wavelet transform. *IEEE Transactions on Image Processing*, 2(4):429–441, 1993.

- [21] W. Cheng, C. Hsieh, S. Lin, C. Wang, and J. Wu. Robust algorithm for exemplar-based image inpainting. In *Proceeding of the International Conference on Computer Graphics, Imaging and Vision*, 2005.
- [22] E. Cohen, T. Tyche, and R. Riesenfeld. Subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 14:87–111, 1980.
- [23] A. Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 13(9):1200–1212, 2004.
- [24] A. Criminisi, T. Sharp, and A. Blake. Geos: Geodesic image segmentation. In *Proceeding of European Conference Computer Vision*, pages 99–112, 2008.
- [25] L. Ddmaret, N. Dyn, and A. Iske. Image compression by linear splines over adaptive triangulations. In *Proceeding of Signal Processing*, volume 86, pages 1604–1616, 2006.
- [26] I. Drori, D. Cohen-or, and H. Yeshurun. Fragment-based image completion. *22(3):303–312*, 2003.
- [27] O. Duchenne, J. Audibert, R. Keriver, J. Ponce, and F. Segonne. Segmentation by transduction. In *Proceeding of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.
- [28] A. Efros and W. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001*, pages 341–346.
- [29] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *Proceeding of IEEE International Conference on Computer Vision*, pages 1033–1038, 1999.
- [30] J. Elder and S. Zucker. Local scale control for edge detection and blur estimation. *IEEE Transactions on Pattern Analysis and Mechine Intelligence*, 20(7):699–716, 1998.

- [31] K. Fan, D. Chen, and M. Wen. A new vectorization-based approach to the skeletonization of binary images. In *Proceeding of The 3rd International Conference on Document Analysis and Recognition*, pages 627–632, 1995.
- [32] G. Farin. *Curves and surfaces for computer aided geometric design*, 2002. Academic Press.
- [33] I. Fidaner. A survey on variational image inpainting, texture synthesis and image completion. 2008.
- [34] M. Froumentin, F. Labrosse, and P. Willis. A vector-based representation for image warping. *Computer Graphics Forum*, 19(3):419–425, 2000.
- [35] T. Goldstein, X. Bresson, and S. Osher. Geometric applications of the split Bregman method: segmentation and surface reconstruction. *Journal of Scientific Computing*, 45(1-3):272–293, 2010.
- [36] L. Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.
- [37] J. Guan and G. Qiu. Interactive image segmentation using optimization with statistical errors. In *Proceeding of ECCV International Workshop on The Representation and Use of Prior Knowledge in Vision*, 2006.
- [38] S. Han, W. Tao, D. Wang, X. Tai, and X. Wu. Image segmentation based on grabcut framework integrating multiscale nonlinear structure tensor. *IEEE Transactions Image Processing*, 18(10):2289–2302, 2009.
- [39] G. Harary and A. Tal. 3d Euler spirals for 3d curve completion. In *Proceeding of Symposium on Computational Geometry*, pages 393–402, 2010.
- [40] G. Harary and A. Tal. The natural 3D spiral. *Journal of Computer Graphics Forum*, 30(2):237–246, 2011.
- [41] X. Hilaire and K. Tombre. Robust and accurate vectorization of line drawing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:890–904, 2006.

- [42] T. Hofmann, J. Puzicha, and J. Buhmann. Unsupervised texture segmentation in a deterministic annealing framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):803–818, 1998.
- [43] H. Hoppe. *Surface Reconstruction from Unorganized Points*. PhD Thesis, Department of Computer Science and Engineering, University of Washington, 1994.
- [44] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proceeding of SIGGRAPH Conference*, pages 295–302, 1994.
- [45] B. Horn. The curve of least energy. *ACM Transactions on Mathematics Software*, 9(4):442–460, 1983.
- [46] N. Houhou, J. Thiran, and X. Bresson. Fast texture segmentation model based on the shape operator and active contour. In *Proceeding of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.
- [47] N. Houhou, J. Thiran, and X. Bresson. Fast texture segmentation based on semi-local region descriptor and active contour. *Numerical Mathematics: Theory, Methods and Applications*, 2(4):445–468, 2009.
- [48] T. Huang, S. Chen, J. Liu, and X. Tang. Image inpainting by global structure and texture propagation. In *Proceeding of the 15th International Conference on Multimedia*, pages 517–520, 2007.
- [49] A. Jain and F. Farrokhnia. Unsupervised texture segmentation using Gabor filters. *Pattern Recognition*, 24(12):1167–1186, 1991.
- [50] S. Jeschke, D. Cline, and P. Wonka. Estimating color and texture parameters for vector graphics. *Computer Graphics Forum*, 30(2):523–532, 2011.
- [51] M. Kass, A. Witkin, and D. Terzopoulos. Snake: active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.
- [52] J. Kim and K. Hong. Color-texture segmentation using unsupervised graph cuts. *Pattern Recognition*, 42(5):735–750, 2009.

REFERENCES

- [53] B. Kimia, I. Frankel, and A. Popescu. Euler spiral for shape completion. *International Journal of Computer Vision*, 54(1):159–182, 2003.
- [54] D.E. Knuth. Mathematical typography. *Bulletin (new series) of the American Mathematical Society*, 1(2):337–372, 1979.
- [55] N. Komodakis and G. Tziritas. Image completion using global optimization. In *Proceeding of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 442–452, 2006.
- [56] P. Kovesi. MATLAB and octave functions for computer vision and image processing. Available from: <http://www.csse.uwa.edu.au/~pk/research/matlabfns/>.
- [57] Y. Lai, S. Hu, and R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Transactions on Graphics*, 28(3), 2009.
- [58] G. Lecot and B. Lévy. ARDECO: Automatic Region DEtection and CONversion. In *Proceeding of Eurographics Symposium on Rendering*, 2006.
- [59] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Journal of The Quarterly of Applied Mathematics*, 2(2):164–168, 1944.
- [60] R. Levien. The Elastica: a mathematical history, 2008. Technical Report UCB/EECS-2008-103, EECS Department, University of California, Berkeley.
- [61] Y. Li, J. Sun, C. Tang, and H. Shum. Lazy snapping. In *Proceeding of SIGGRAPH Conference*, pages 303–308, 2004.
- [62] C. Loop. *Smooth Subdivision Surfaces Based on Triangles*. Master thesis, Department of Mathematics, University of Utah, 1987.
- [63] M. Lourakis. A brief description of the Levenberg-Marquardt algorithm implemented by Levmar. Institute of Computer Science Foundation for Research and Technology-Hellas, 2005, Available from: <http://www.ics.forth.gr/~lourakis/levmar/levmar.pdf>.

- [64] J. Malcolm, Y. Rathi, and A. Tannenbaum. A graph cut approach to image segmentation in tensor space. In *Proceeding of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.
- [65] D. Marquardt. An algorithm for the least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.
- [66] D. Meek and D. Walton. An arc spline approximation to a clothoid. *Journal of Computational and Applied Mathematics*, 170(1):59–77, 2004.
- [67] E. Mehlum. Nonlinear splines. pages 173–207, 1974.
- [68] K. Mielenz. Computation of Fresnel integrals II. *Journal of Research of the National Institute of Standards and Technology*, 105(4):589, 2000.
- [69] N. Montes, A. Herraiez, L. Armesto, and J. Tornero. Real-time clothoid approximation by rational Bezier curves. In *Proceeding of International Conference on Robotics and Automation*, pages 2246–2251, 2008.
- [70] E. Mortensen and W. Barrett. Intelligent scissors for image composition. In *Proceeding of SIGGRAPH Conference*, pages 191–198, 1995.
- [71] E. Mortensen and W. Barrett. Interactive segmentation with intelligent scissors. *IEEE Transactions on Image Processing*, 60(5):349–384, 1998.
- [72] D. Mumford. *Elastica and computer vision*. pages 491–506, 1994.
- [73] A. Nguyen, J. Cai, J. Zhang, and J. Zheng. Robust interactive image segmentation using convex active contour. accepted by *IEEE Transactions on Image Processing*, 2012.
- [74] M. Nitzberg, D. Mumford, and T. Shiota. *Filtering, Segmentation and Depth*. Springer-Verlag New York, 1993.
- [75] K. Okarma. Colour image quality assessment using structural similarity index and singular value decomposition. In *Proceeding of IEEE International Conference on Computer Vision and Graphics*, 2008.

REFERENCES

- [76] OpenCV. Open source computer vision. Available from: <http://opencv.willowgarage.com/wiki/>.
- [77] A. Orzan, A. Bousseau, H. Winnemoller, P. Barla, J Thollot, and D. Salesin. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics*, 27, 2008.
- [78] B. Price. Image vectorization. *Master's thesis of Department of Computer Science of Brigham Young University*, 2006.
- [79] B. Price and W. Barrett. Object-based vectorization for interactive image editing. In *Proceeding of Pacific Graphics*, volume 22, pages 661–670, 2006.
- [80] B. Price, B. Morse, and S. Cohen. Geodesic graph cut for interactive image segmentation. In *Proceeding of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 3161–3168, 2010.
- [81] L. Reese. *Intelligent Paint: Region-based Interactive Image Segmentation*. Master's Thesis, Brigham Young University, 1999.
- [82] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: interactive foreground extraction using iterated graph cuts. In *Proceeding of SIGGRAPH Conference*, pages 309–314, 2004.
- [83] M. Rousson, T. Brox, and R. Deriche. Active unsupervised texture segmentation on a diffusion based feature space. In *Proceeding of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 699–706, 2003.
- [84] W.S. Rutkowski. Shape completion. *Computer Graphics and Image Processing*, 9:89–101, 1979.
- [85] C. Sagiv, N. Sochen, and Y. Zeevi. Integrated active contours for texture segmentation. *IEEE Transactions on Image Processing*, 15(6):1633–1646, 2006.
- [86] J. Sanchez-Reyes and J. Chacon. Polynomial approximation to clothoids via S-power series. *Journal of Computer Aided Design*, 35(14):1305–1319, 2003.

- [87] J. Schweitzer. *Analysis and Application of Subdivision Surface*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1996.
- [88] J. Shewchuk. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: towards Geometric Engineering*, volume 1148, pages 203–222. 1996.
- [89] M. Singh and J. Fulvio. Visual extrapolation of contour geometry. *National Academy of Sciences*, 102(3):939–944, 2005.
- [90] A. Sinop and L. Grady. A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *Proceeding of IEEE International Conference Computer Vision*, pages 1–8, 2007.
- [91] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. The 2th Edition Pacific Grove, 1999.
- [92] D. Su and P. Willis. Image interpolation by pixel-level data-dependent triangulation. *Computer Graphics Forum*, 23:189–201, 2004.
- [93] J. Sun, L. Liang, F. Wen, and H. Shum. Image vectorization using optimized gradient meshes. In *Proceeding of SIGGRAPH Conference*, 2007.
- [94] J. Sun, L. Yuan, J. Jia, and H. Shum. Image completion with structure propagation. In *Proceeding of SIGGRAPH Conference*, pages 861–868, 2005.
- [95] S. Ullman. Filling-in the gaps: the shape of subjective contours and a model for their generation. *Biological Cybernetics*, 25(1):1–6, 1976.
- [96] D. Walton and D. Meek. A controlled clothoid spline. *Computers & Graphics*, 29(3):353–363, 2005.
- [97] D. Walton and D. Meek. An improved Euler spiral algorithm for shape completion. In *Proceeding of Canadian Conference on Computer and Robot Vision*, pages 237–244, 2008.

REFERENCES

- [98] L. Wang, K. Miura, E. Nakamae, T. Yamamoto, and T. Wang. An approximation approach of the clothoid curve defined in interval $[0, \pi/2]$ and its offset by free-form curves. *Journal of Computer Aided Design*, 33(4):1049–1058, 2001.
- [99] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [100] L.Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 2000*, pages 479–488. ACM Press/Addison-Wesley Publishing Co.
- [101] J. Wu and Q. Ruan. Object removal by cross isophotes exemplar-based inpainting. In *Proceeding of the 18th International Conference on Pattern Recognition*, 2006.
- [102] T. Xia, B. Liao, and Y. Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Transactions on Graphics*, 28, 2009.
- [103] X. Xu, L. Wan, X. Liu, T. Wong, L. Wang, and C. Leung. Animating animal motion from still. *ACM Transactions on Graphics*, 27:117:1–117:8, 2008.
- [104] W. Yang, J. Cai, J. Zheng, and J. Luo. User-friendly interactive image segmentation through unified combinatorial user inputs. *IEEE Transactions on Image Processing*, 19(9):2470–2479, 2010.
- [105] J.H. Yong and F. Cheng. Geometric hermite curves with minimum strain energy. *Computer Aided Geometric Design*, 21(3):281–301, 2004.
- [106] X. Yu, B. Morse, and T. Sederberg. Image reconstruction using data-dependent triangulation. *IEEE Computer Graphics and Applications*, 21:62–68, 2001.
- [107] J. Zhang, J. Zheng, and J. Cai. A diffusion approach to seed image segmentation. In *Proceeding of IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2125–2132, 2010.

REFERENCES

- [108] S. Zhang, T. Cheng, Y. Zhang, S. Hu, and P. Martin. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics*, 15:618–629, 2009.
- [109] J. Zou and H. Yan. Cartoon image vectorization based on shape subdivision. In *Proceeding of Computer Graphics International*, pages 225–231, 2001.

Appendix A

Author Publication List

1. Hailing Zhou, Jianmin Zheng, Lei Wei, “OVIE: Object-oriented and vector-based image editing”, In Proceedings of International Conference on Multimedia & Expo (ICME) Workshop, 2012.
2. Hailing Zhou, Jianmin Zheng, “Curvilinear feature driven image vectorization using subdivision-based representation”, 2012
3. Hailing Zhou, Jianmin Zheng, Lei Wei, “Texture Aware Image Segmentation Using Graph Cuts and Active Contours”, 2012
4. Hailing Zhou, Jianmin Zheng, Xunnian Yang, “Euler arc spline for shape completion”, Computers & Graphics, 36(6), Pages 642-650, 2012
5. Hailing Zhou, Jianmin Zheng, “Adaptive patch size determination for patch-based image completion”, In Proceedings of International Conference on Image Processing, pp.421-424, 2010
6. Hailing Zhou, Jianmin Zheng and Xin Li, “Image Reconstruction using Loop Subdivision”, In Proceedings of Asia Pacific Signal and Information Processing Association, 2010
7. Hailing Zhou, Jianmin Zheng, Hock Soon Seah, “Converting bitmap images into scalable vector graphics”, In Proceedings of International Workshop on Advanced Image Technology, 2009