10-21-2011 12:00 AM

Classic Mosaics and Visual Correspondence via Graph-Cut based Energy Optimization

Yu Liu, *The University of Western Ontario*

Supervisor: Dr. Olga Veksler, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Computer Science

Classic Mosaics and Visual Correspondence via Graph-Cut based Energy Optimization

(Thesis Format: Monograph)

by

Yu Liu

Graduate Program in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario
September, 2011

THE UNIVERSITY OF WESTERN ONTARIO
SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Advisor

Dr. Olga Veksler

Supervisory Committee

Examining Board

Dr. Steven Beauchemin

Dr. James Elder

Dr. James Lacefield

Dr. Charles Ling

The thesis by
Yu Liu
entitled

Classic Mosaics and Visual Correspondence via Graph-Cut based Energy Optimization

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date

Chair of Examining Board

Abstract

Computer graphics and computer vision were traditionally two distinct research fields focusing on opposite topics. Lately, they have been increasingly borrowing ideas and tools from each other. In this thesis, we investigate two problems in computer vision and graphics that rely on the same tool, namely energy optimization with graph cuts.

In the area of computer graphics, we address the problem of generating artificial classic mosaics, still and animated. The main purpose of artificial mosaics is to help a user to create digital art. First we reformulate our previous static mosaic work in a more principled global optimization framework. Then, relying on our still mosaic algorithm, we develop a method for producing animated mosaics directly from real video sequences, which is the first such method, we believe. Our mosaic animation style is uniquely expressive. Our method estimates the motion of the pixels in the video, renders the frames with mosaic effect based on both the colour and motion information from the input video. This algorithm relies extensively on our novel motion segmentation approach, which is a computer vision problem.

To improve the quality of our animated mosaics, we need to improve the motion segmentation algorithm. Since motion and stereo problems have a similar setup, we start with the problem of finding visual correspondence for stereo, which has the advantage of having datasets with ground truth, useful for evaluation. Most previous methods for stereo correspondence do not provide any measure of reliability in their estimates. We aim to find the regions for which correspondence can be determined reliably. Our main idea is to find corresponding regions that have a sufficiently strong texture cue on the boundary, since texture is a reliable cue for matching. Unlike the previous work, we allow the disparity range within each such region to vary smoothly, instead of being constant. This produces blob-like semi-dense visual features for which we have a high confidence in their estimated ranges of disparities.

Keywords: Computer Vision, Computer Graphics, Animated Mosaics, Classic Mosaics, Stereo, Visual Correspondence, Energy Optimization, Graph Cuts

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Dr. Olga Veksler, who guided me through this work. She was always there whenever I needed her help. She made this thesis possible with years of effort, patience, and help. Without her constant support, encouragement and technical guidance from all aspects, I could not have overcome any difficulties in my research.

I would like to thank Dr. John Barron and Dr. Steven Beauchemin for their valuable comments for my thesis proposal.

Special thanks are given to the members of my examining committee, Dr. Steven Beauchemin, Dr. James Elder, Dr. James Lacefield, and Dr. Charles Ling, for their valuable suggestions on revising and finalizing this thesis.

I would like to thank Dr. Lena Gorelick and Dr. Andrew Delong, who helped me a lot with my research.

I wish to thank Dr. Yuri Boykov for his valuable and helpful suggestions to my research.

I would like to thank people in the department, especially the staff members in the main office and the members in the system group, for their help and support over the period of both my M.Sc. and Ph.D. programs.

I wish to express my gratitude to the Department of Computer Science for providing me the opportunity to study here and the financial support.

I would also like to thank my friends, who were always beside me during the past years.

Last but not least, I would like to thank my parents and my husband, to whom I dedicate this work, for their endless support, encouragement, and love.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Energy Optimization in Computer Vision and Graphics

Computer vision is the science and technology that enables the machines to see. Its goal is to extract information from an image to “understand” a scene. Its applications range from simple tasks such as counting the number of bottles on a production line to the research in artificial intelligence and robotics with the goal to comprehend the world around them. On the contrary, computer graphics manipulates the visual content of the images to create digital synthesis of the real world. The main tasks of computer graphics include representation of three-dimensional objects (geometry), simulation of object deformation through time (animation), and generating realistic or stylized images from models (rendering).

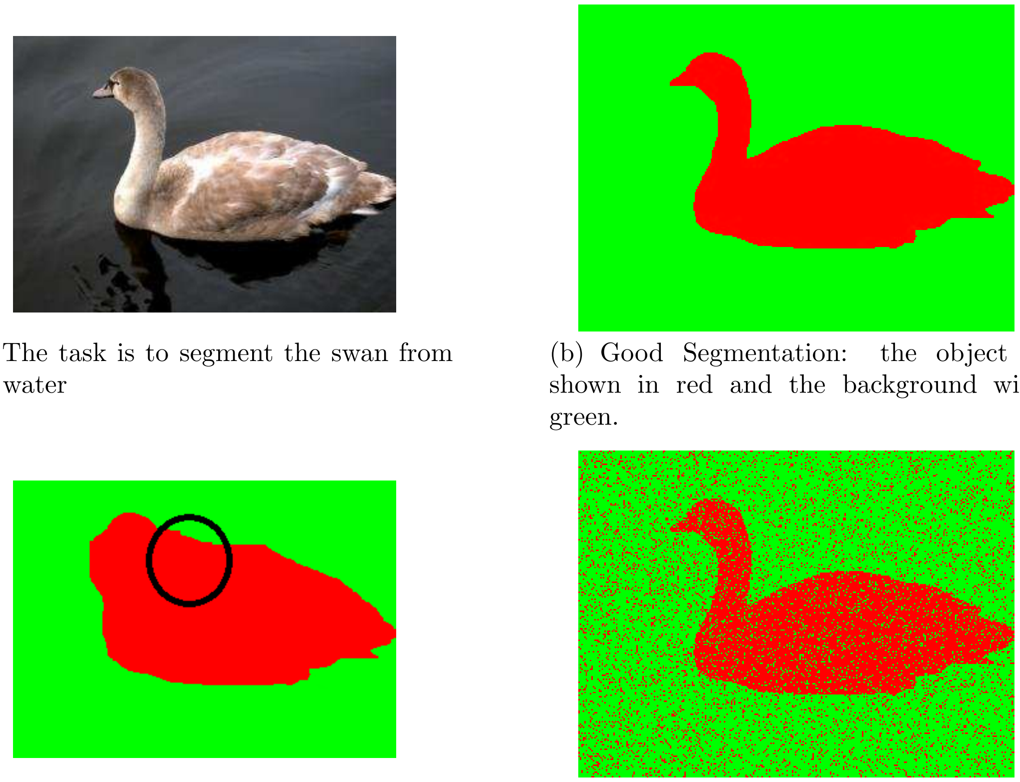
Traditionally, computer vision and graphics were two distinct fields since they focus on different topics of interest. However, recently, with the emerging of new techniques such as 3D television, telecommunication and virtual reality, vision and graphics are increasingly borrowing ideas and techniques from each other. These new techniques address the issues such as visually displaying the results in high quality, modeling user interaction with the environment naturally and intuitively. Since computer graphics devotes a lot on real-world synthesis, such as photo-realistic rendering [20, 72, 2] and object modeling [45, 88, 102], one successful approach is to use real images and 3D scans, which are usually viewed as computer vision methods. Computer vision also uses many techniques from computer graphics. For example, 3D modeling is widely used in recognition systems for analysis and synthesis [9, 70]. In order to deal with all these new challenges, we need to investigate the vision and graphics problems in a new way which integrates these two fields.

Most problems in computer vision and graphics are very challenging. Due to the ambiguities in visual interpretation, uncertainty and large dimensionality of the data, there are usually many possible solutions unless additional constraints from prior knowledge are imposed. Consider the image segmentation problem proposed in Figure 1.1. The task is to segment the light object (swan), from the darker background (water). Thus we want to segment the lighter pixels as the swan and the darker pixels as the background. Figure 1.1(b) to Figure 1.1(d) illustrate three different possible solutions to this segmentation problem. It is easy for a human to conclude that Figure 1.1(b) is the best solution among these three choices. However, an automatic computer vision system would need precise instructions on how to measure the "goodness" of a solution. Thus some mechanism is required to evaluate all the options and select the best one. The optimization approaches provide an expressive way to solve this problem.

There are usually two major steps in the optimization framework. The first step is to formulate an objective function. The objective function maps any solution to a real number, this number being a measure of how good the solution is. A good objective function should incorporate the constraints that an acceptable solution must satisfy. The objective function should assign high goodness score to solutions that match the constraints well. In this thesis, we refer to the objective functions as *energy* functions.

An energy function assigns a *low* energy to a good solutions and *high* energy to a bad solutions. Figure 1.2 schematically illustrates an example of possible objective functions for the image segmentation problem proposed in Figure 1.1. The smaller is the value of the energy function, the better is the solution, as seen in Figure 1.2(b). A global minimum of the energy function, by definition, gives the optimal solution to the problem. In computer vision, the objective/energy functions usually have huge dimensions which makes them unfeasible to plot.

There are two commonly used constraints in designing an energy function, the data



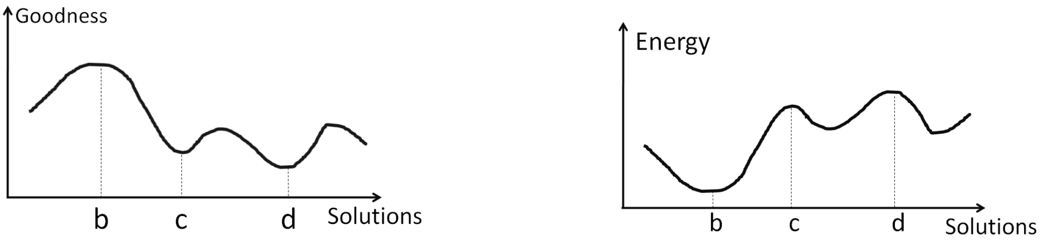
(a) The task is to segment the swan from the water

(b) Good Segmentation: the object is shown in red and the background with green.

(c) Deviation from observed images: observe the region inside the black circle. A great number of pixels are segmented as the swan. Comparing with (a), these pixels in fact have more similar colour to the water than to the swan. Segmenting these pixels as the object violates the data constraint.

(d) Deviation from prior knowledge: the swan and background shapes are not coherent, with many tiny "holes". This violates our prior knowledge that these object and background shapes tend to be contiguous in space.

Figure 1.1: *Example of image segmentation problem and its constraints.*



(a) Objective Function

(b) Energy Function

Figure 1.2: *An example of objective function and energy function for the image segmentation problem in Figure 1.1: Figure (a) is a possible objective function. Notice the solution in Figure 1.1(b), namely b, is assigned higher goodness score than c and d in Figure 1.1(c) and Figure 1.1(d). In this work, we usually refer the objective function as an* energy *function. Energy functions assign low energy to good solutions, as shown in Figure (b).*

constraint and prior constraint. The data constraint comes from the observed data. It requires a desired solution to be close to the observed data. For example, in Figure 1.1, it is easy to come up with the constraint that the pixels that belong to the object should have lighter colours and the background pixels should have darker colours. Otherwise they are violating the data constraint provided by the colour information of the image. The prior constraint is from our prior knowledge about a physically plausible solution to the problem. Most physical world objects are coherent in space, that is for most pixels in the object, all nearby pixels also belong to that object. In this case, our prior knowledge tells us that both the background and the object should be spatially coherent. We can encode more prior constraints in the energy functions, such as a preference to a particular shape, say a round shape prior, which encourages the object to have a round shape.

The second step of the optimization approach is to minimize the energy function. This is also a very challenging problem. Since computer vision and graphics usually deals with images and videos, the problem size is huge. It is impossible to find the optimal solution by simple enumeration. Moreover, the energy functions are usually not convex, which makes it hard to find their minima by using standard minimization method such as gradient descent. Many authors address this issue by making a compromise in the design of the energy function, that is a more difficult to deal with but more appropriate energy function is replaced by a less appropriate, but easier to optimize energy. However, even with compromises, optimization is still hard.

Although it is difficult to formulate appropriate energy functions as well as to optimize them, there are many advantages to the optimization based approach. First, it provides a common framework which abstracts useful constraints from details of each particular problem. Once an energy function is formulated, the standard optimization approaches can be applied to solve it. Secondly, it enables us to apply our prior knowledge to solve the problem by encoding it in the energy function. Thus we can expect the desired solution to have some nice global properties, such as the overall smoothness in the image segmentation example in Figure 1.1. Finally, the value of the energy function provides an effective way to evaluate the solution and can be used as a guide in the optimization algorithm.

Great effort has been made towards developing effective energy optimization algorithms. Among all these approaches, graph cut algorithm [12, 13, 53] has been proven to be an effective tool for computer vision and graphics applications. As a global optimization method, the graph cut algorithm can find the exact minimum of certain energy functions (the everywhere smooth prior [44], defined in Section 2.1). For a wider range of energy functions (e.g. piecewise smooth prior, defined in Section 2.1) which are NP-hard to optimize, it can find a local minimum within known factor from the optimal [12]. Although the class of energy functions that can be optimized by the graph cut approach is restricted, it is useful enough to be applied to a wide range of vision and graphics problems.

In this thesis, we address two graphics and vision applications: rendering artificial classic mosaics and finding semi-dense cues for visual correspondence. We formulate both the animated mosaic problem and visual correspondence problem in the energy optimization framework. Graph cut algorithm is used as the energy optimization tool in our work to solve these two problems. We illustrate how we formulate the energy functions in these two applications, together with how we adapt our energy functions so that they can be optimized by graph cuts.

1.2 Artificial Classic Mosaics

1.2.1 Still and Animated Mosaics

Computer graphics is not only about simulating realistic images, a large area of graphics is devoted to creating artful effects to enhance the images taken from the real world. For example, people often use tools such as Photoshop to create different effects on the real world pictures so that the result images may be more appealing to the user. There are also tools helping the users to create art works, such as artificial oil paints, line drawings and mosaics etc. With all these tools based on computer vision and graphics techniques, more ordinary people will be given the freedom of creating their own art work. The users would be even more interested if one can create animations of their own, with computer aided tools.

Non-photorealistic rendering (NPR) is an area of comptuer graphics which deals with rendering real world photographs in an artistic style. NPR rendering creates stylized images by arranging drawing primitives, for instance painting strokes, line segments or tiny dots, in an artful way. The purpose of NPR is to assist humans in creating digital art and to render images in such a way that the important information that they contain in emphasized. Recently, there has been a great interest in non-photorealistic rendering, such as artificial line drawing [24, 25], digital painting [68, 54, 39], stippled drawing [82, 21, 71], and classic mosaics [36, 23, 63, 64].

Mosaic is one of the most durable and ancient art forms. As early as ancient Roman times, people use this durable art forms to decorate walls, ceilings and furniture etc. It is usually composed of thousands of primitives, such as colourful stones, ceramic tiles and glass fragments, arranged along the important edges of the desired scene. If

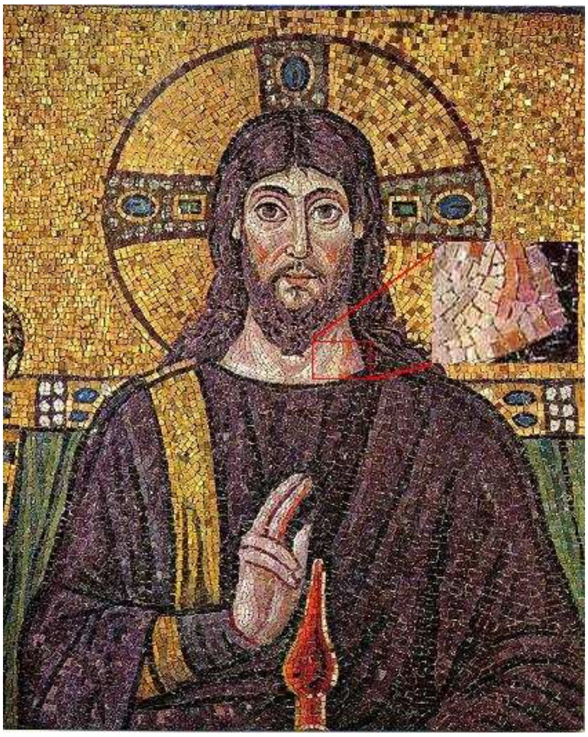


Figure 1.3: *Classic mosaic example: Christ surrounded by angels and saints, from basilica of Sant'Apollinare Nuovo in Ravenna, Italy. Notice inside the red rectangle, the artist broke the square tiles into irregular shapes to adapt to the image.*

the drawing primitives of a mosaic image, namely the "tiles", have square or rectangle shape, then this mosaic belongs to the category of classic mosaic. Figure 1.3 shows an example of classic mosaics. Simulating static mosaics from digital images is one area of non-photo-realistic rendering and it has been widely investigated, see [36, 23, 5, 64].

As already mentioned above, one goal of NPR rendering is to create images with a more profound impact on the viewer. It is even more true for an NPR animation, since it has one more dimension(time) of expressiveness. Although there are great number of works on static NPR rendering, such as [24, 25, 39, 36, 64], relatively little work has been done towards generating NPR animations automatically or iteratively [85, 18, 50]. In this thesis, we propose a method which renders animations with classic mosaic effects from real video.

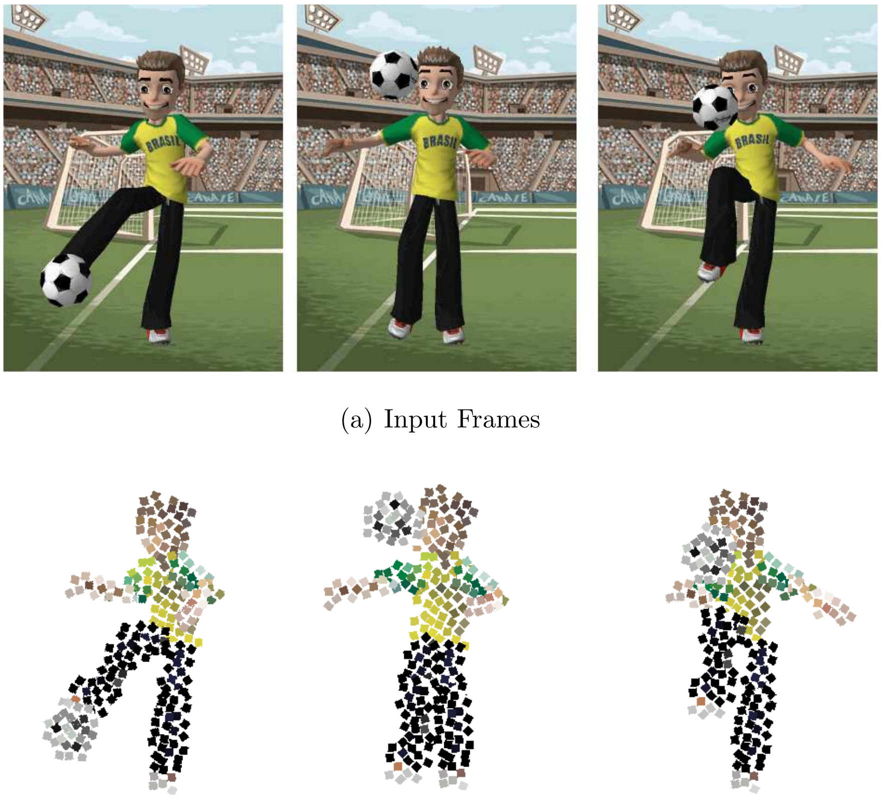
Little work has been reported on generating mosaic animations automatically, especially for animated mosaics from real video. The animations generated by our approach are composed of hundreds of colourful square tiles, which are arranged to present the shape and colour of the objects inside the given video, moving in a timely coherent manner. Each frame of the resulting animation is a classic mosaic image composed of a great number of square tiles, which are located along the important edges inside the given scene. Between the consecutive frames, the tiles are moved according to the motion of their center pixels. Therefore, the whole animation will have a consistent motion effect.

Figure 1.4 shows a synthetic example of our desired mosaic animation. The input video shows a boy playing soccer. The resulting frames of the mosaic animation are shown in Figure 1.4(b). The square tiles are moved from one frame to the next according to the motion information detected at their center pixels, which generates a temporally coherent motion effect.

Our method estimates the motion of the pixels inside the video, renders the frames with mosaic effect based on both the colour and motion information from the input video. We aim at a tool that requires minimal help from the user to finish the task of generating animated mosaics. We hope with the help of our animation tool, more people will be able to create their own mosaic animation without professional training.

1.2.2 Constraints on Static and Animated Mosaics

Most of the work on rendering static mosaics is inspired by observing the artists. To obtain a visually appealing mosaic, there are some basic rules that almost all methods follow. First, the edges of the mosaic tiles should be parallel to the important edges of the underlying image, as shown in Figure 1.5(c). Which edges are essential can be decided in various ways. Some of the previous works require the users to input a set



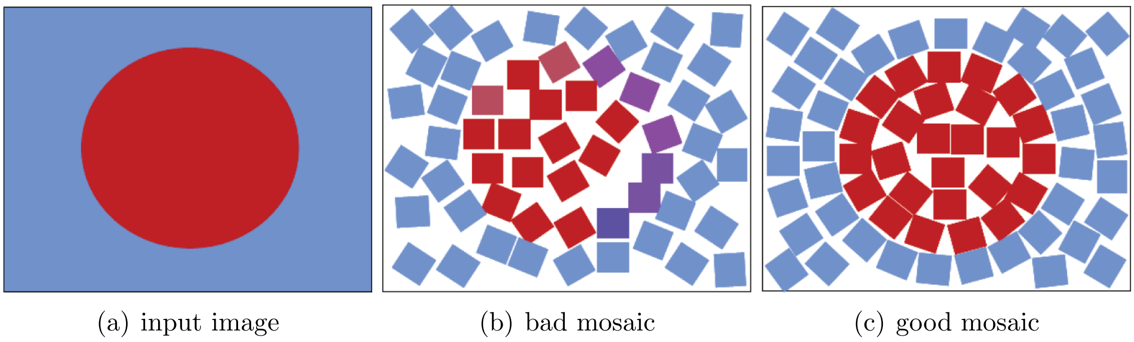
(a) Input Frames

(b) Mosaic Frames

Figure 1.4: *One example with animated mosaics: the input video shows a boy playing soccer, shown in Figure (a). The resulting frames of the mosaic animation are in Figure (b). The square tiles are moved from one frame to the next according to the motion information detected at their center pixels, which generates a temporally coherent motion effect.*

of curves to indicate the objects they want to emphasize in the images, such as the work of Hausner [36] and of Elber and Wolberg [23]. The others, for instance, the work of Di Blasi et al. [10] and Battiato et al.[6], adopt edge detection algorithms to generate a set of principal curves and use them as the guide lines for tile orientation. Besides these two approaches, some of the static mosaic approaches encode the edge information as a soft cue when computing the tile orientations, such as Liu et al. [64] and Battiato et al. [5].

In addition, to reduce the gap space inside the mosaic image, tiles must be packed



(a) input image (b) bad mosaic (c) good mosaic

Figure 1.5: *A synthetic example of a good and a bad mosaic: Figure (a) shows the synthetic input image. Figure (b) shows a bad tiling example. The tiles inside Figure (b) are not aligned with the edge of the circle and the neighbouring tiles have tile orientations not emphasizing the circular shape of the central object. This results in a visually unappealing mosaic with large gap space and blurred colours. Figure (c) shows a good tiling example, where tiles are aligned with the shape of the circle and packed tightly.*

as tightly as possible, see Figure 1.5(c). It is NP-hard to find the optimal solution to the tile packing problem, since it can be considered as a bin packing problem. Many previous works on rendering classic mosaics are based on heuristics, such as Hausner [36], Elber and Wolberg [23], and Battiato et al. [5]. However, it is still possible to find an approximate good solution, after transforming the tile packing problem into a labeling problem.

While it is possible to create simple animations with mosaic effects manually, it is very challenging to create animated classic mosaics automatically from real video. There are two main difficulties in rendering animated mosaics from video. First, each frame of the animated mosaics should itself be an appealing static mosaic. Thus animated mosaics have to obey all the constraints for a static mosaic: the tile orientation should be parallel to the principal edges in the input image, and the tiles should be packed tightly to minimize the gap space inside the mosaic. Second and more important, to create the animation effect, the motion of the tiles must be spatially and temporally consistent. For example, two neighbouring tiles on the right arm of the boy in Figure 1.4(b) should move with similar velocity through all the frames. Otherwise they will move away from each other and generate unpleasant “splitting” artifacts in the boy's right arm. The only exemption for motion consistency is at object boundaries, where neighbouring tiles belong to different objects therefore can have different motions. Moreover, to maintain the classic mosaic style, the tile shape, usually square or rectangle, is not allowed to deform, scale or blend, unlike what is allowed in NPR animations rendered in other styles (i.e. oil painting style).

1.2.3 Motivation and Contribution

We improve our previous work on classic static mosaics [63] by addressing this problem in a more principled global energy minimization framework. Our main contribution to the area of non-photorealistic rendering is our animated mosaic method. Most previous works on generating non-photorealistic animations, such as the work of Klein et al. [50], require the users to provide the motion information for the rendering primitives. This is a very tedious work even for very short video clips. Moreover, rendering primitives are usually deformed during the animation process, for instance, in the work of Litwinowicz [60]. Our goal is to generate animated classic mosaics without the user providing full motion information. The constraints on classic mosaic also require that the mosaic tiles are not deformed in any case. Therefore, the previous methods on rendering NPR animations can not be applied to our animation problem.

Unlike the work of Smith et al. [85], we render animated mosaics from real video sequences. As already mentioned, one of the main challenges is to make sure there is a temporal coherence in the animation. One way to achieve temporal coherency is to displace groups of tiles in a consistent manner. For this purpose we develop a new motion segmentation algorithm with occlusion reasoning. Our algorithm requires minimal help from the user. We pack the tiles into the discovered coherent motion layers, using colour information in all the frames in a global manner. Our tile packing algorithm is based on the one for still mosaic proposed by Liu et al. [64], with several modifications to address video input. We also restate the problem of rendering static mosaics as a global energy optimization frame work in Chapter 3. Occlusions are handled gracefully. We produce colourful, temporally coherent and uniquely appealing mosaic animations. We believe that our method is the first one to animate classic mosaics directly from video.

1.3 Semi-dense Visual Correspondence

1.3.1 Visual Correspondence

In order to create a faithful mosaic animation, the motion of the pixels inside the given video must be estimated accurately. Rendering each frame individually without the motion information leads to unpleasant "flickering" effects which are disturbing to the viewer. Motion estimation is usually performed by finding the visual correspondence between consecutive frames. In the visual correspondence problem, we are given two images of the same real world scene. A pixel in one image is said to correspond to a pixel in the other image, if these two pixels are projections along the lines of sight of the same physical scene element, see Figure 1.6. The problem is to find pairs of such corresponding pixels.

In stereo, the real world is captured by synchronized cameras from distinct view points. The straight line connects the optical centers of these two cameras is called the baseline. Usually it is assumed that this baseline is parallel to the image planes for both cameras after rectification. Corresponding points are found between these two images, see Figure 1.6(a). The difference in the locations of corresponding pixels seen in the left and right images, often referred as "disparity", is used to determine the depth[1] of these pixels. The disparities for stereo problem are only in the horizontal dimension, which means the corresponding pixels are in the same scanline. This

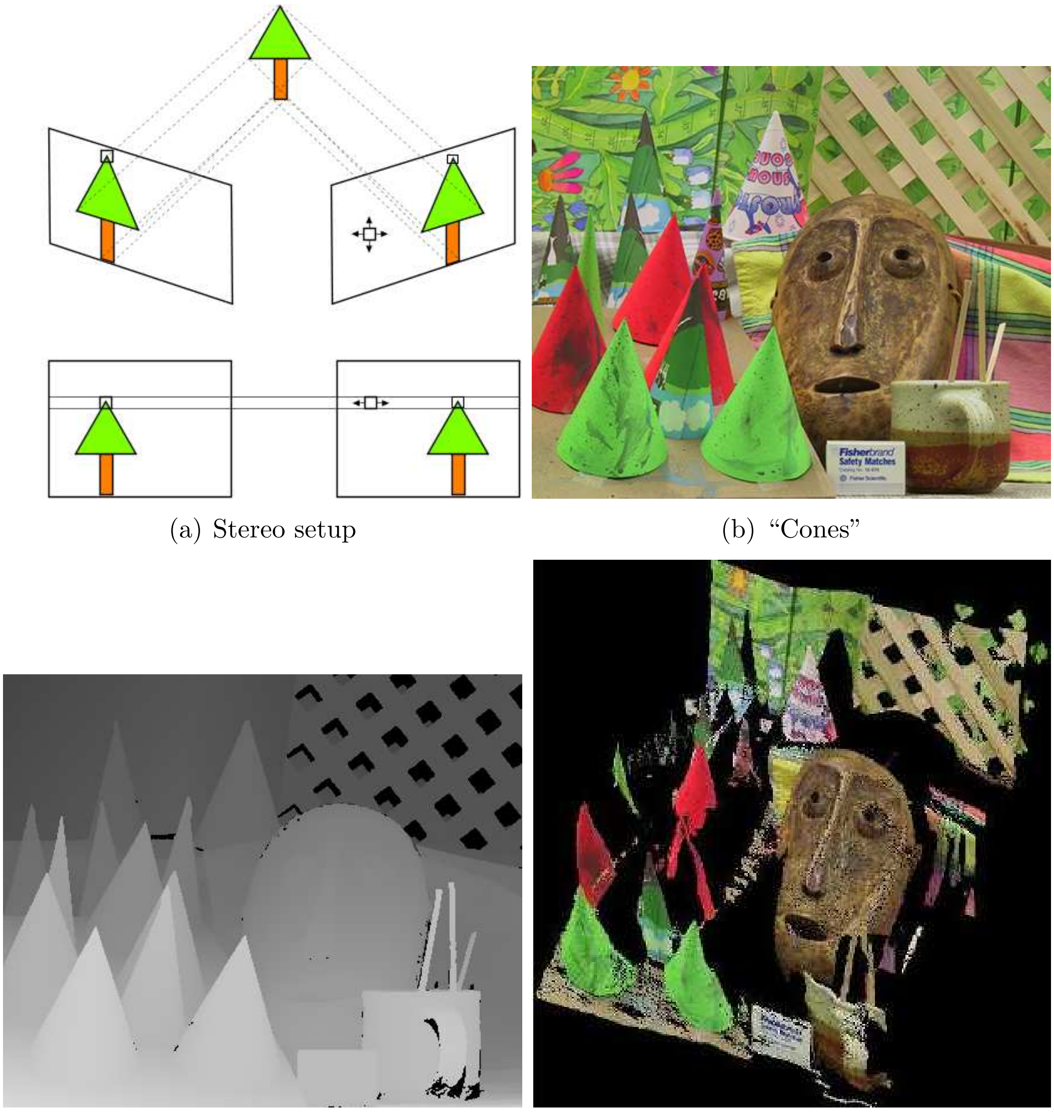
[1] Here "depth" means the distance between the world point corresponding to the pixel and the image plane of the camera.

concept of visual correspondence can be easily adapted to motion. In motion, images are taken from the same view points at consecutive time and correspondence serves as the evidence of motion.

Figure 1.6 shows a synthetic example and a real example of visual correspondence for stereo. Figure 1.6(a) shows a synthetic example of stereo. The images are taken by two synchronized cameras, and are rectified so that the cameras share the same baseline parallel to their image planes. Figure 1.6(b) shows the right image of a stereo image pair. Figure 1.6(c) shows the true disparities of the image given in Figure 1.6(b). And Figure 1.6(d), taken from [103], shows the depth of the pixels in Figure 1.6(b). For each pixel in Figure 1.6(c), the brighter is the intensity, the larger is the disparity, hence the closer is to the camera, as shown in Figure 1.6(d). The problem of finding visual correspondence is still not a well solved problem after many years of investigation. This, together with our need to find more accurate motion for the tiles for mosaic animation, inspires us to extend our research into the field of finding visual correspondence. Most methods intended for stereo correspondence are easily extended to motion correspondence, however stereo correspondence has a simpler setup. Thus although our intent is to develop visual correspondence for motion sequences, we begin by investigating correspondence for stereo.

1.3.2 Visual Correspondence: Main Challenges

Finding the visual correspondence is a very challenging problem. To get the correct visual correspondence, a lot of problems have to be overcome. For instance, image noise will bring errors in when measuring the difference between corresponding pixel pair. Other problems can be caused by sampling artifacts, exposure change, motion blur, etc. Among all these problems, lack of texture and occlusions are two of the hardest problems to solve when computing the visual correspondence. First, in textureless areas or areas with repeating texture, it is difficult to calculate the correct



(a) Stereo setup

(b) “Cones”

(c) Disparity map for “Cones”

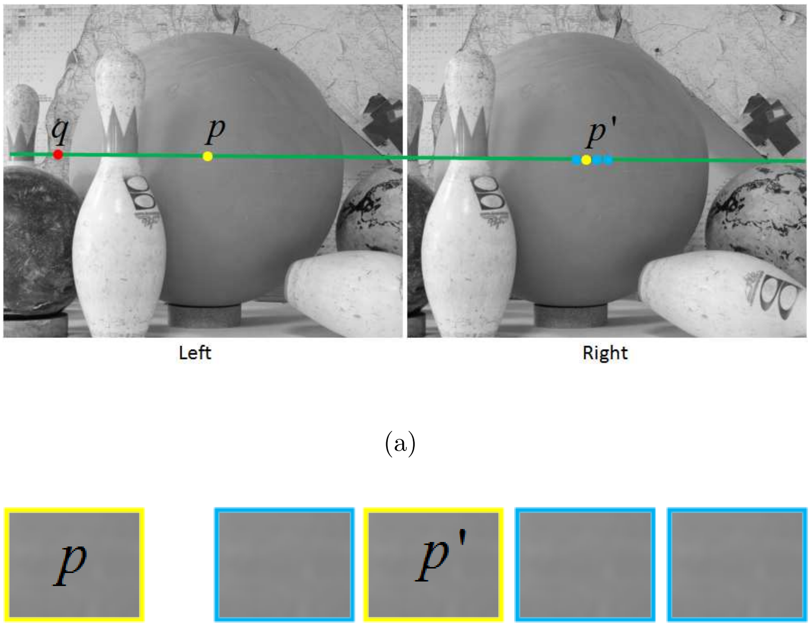
(d) Depth for “Cones”

Figure 1.6: *Examples of stereo: Figure (a) Stereo setup. The images are taken by two synchronized cameras, and are rectified so that the cameras share the same baseline parallel to their image planes. The two pixels marked by the black squares are corresponding pixels, since they are both the projections of the tree tip. By finding the correspondence between the pixels in the left and right images, we can then compute the displacement of the pixels from right to left image, which is also referred as “disparity”. In stereo, the disparities are only in horizontal direction. Once the disparity is known, the depth of the corresponding scene point can be found by triangulation. Figure (b) shows the right image of a stereo image pair. Figure (c) shows the ground truth for the disparities of the image given in Figure (b). And Figure (d) shows the depth of the pixels in Figure (b). For each pixel in Figure (c), the brighter is the intensity, the larger is the disparity, hence the closer is to the camera, as shown in Figure (d).*

matching error for correspondence due to ambiguity. Second, in occluded areas, where part of an object can be seen in one image but missing in other images, it is hard to obtain correct correspondences since information is lost.

Figure 1.7 illustrates an example for stereo image pair with textureless and occluded regions. Figure 1.7(a) shows the input stereo image pair. Pixel p and pixel q are two pixels in the left image, located on the same scanline. Here we use yellow and red dots to show their locations. Pixel p' is the pixel that corresponds to p in the right image, shown with a yellow dot. Pixel q does not have a corresponding pixel in the right image since the wall part where it is located on is occluded by the ball in the right image. The blue dots are the neighbouring pixels of pixel p'. In Figure 1.7(b), we enlarge pixels p, p' and the neighbours of p' to show their intensity. As shown in the Figure 1.7(a), pixel p and p' locate in a region with little texture. The neighbours of p' also have very similar intensity with pixel p, which makes it hard to decide the true disparity of p. Pixel q is actually occluded by the bowling ball in the right image. Therefore its disparity can not be determined without further assumptions about the scene.

A lot of research has been devoted to solve these problems. The existing approaches can be mainly divided into two categories. On one hand, correspondence can be found by using sparse feature points. These methods obtain promising results in regions with texture cues, but their results can be too sparse to be used in some applications such as image-based rendering. Dense correspondence approaches estimate disparity at every pixel but can have gross errors in some parts of textureless areas. Worse still, most dense approaches do not produce confidence maps for their estimates, that is they do not say which parts of the disparity maps they are more confident in. That is the motivation for the proposed method. Our main objective is to find semi-dense visual correspondence in the areas of an image where disparities can be found more reliably.



(a)

(b)

Figure 1.7: *An example for occlusion and textureless regions: Figure (a) shows the input stereo image pair. Pixel p and pixel q are two pixels in the left image, located on the same scanline. Here we use yellow and red dots to show their positions. Pixel p' corresponds to p in the right image, but pixel q does not have correspondence in the right image, it is occluded by the ball. The blue dots are the neighbouring pixels of pixel p'. In Figure (b), we enlarge pixel p, p' and the neighbours of p' to show their intensity. As shown in the Figure (a), pixel p and p' locate in a region with little texture. The neighbours of p' also have very similar intensity with pixel p, which makes it hard to determine the true disparity of p.*

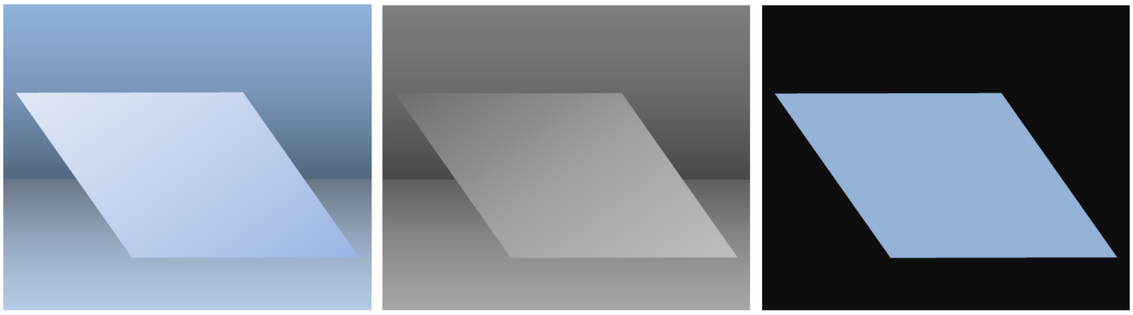
1.3.3 Semi-dense Visual Correspondence

The difficulties in finding visual correspondence are caused by many reasons, including image noise, image sampling artifacts, textureless regions and occlusions. It is well known that a reliable visual correspondence cue can be found more easily in the textured regions. This is because the intensity changes provide more information about the reliability on the matching quality of the corresponding pixel pairs. This principle also applies to the boundaries of the textureless region. Usually, we can establish visual correspondence of the pixels on the boundary of the textureless regions much more reliably than in the interior of the textureless regions. If we can propagate the information provided by the boundaries to the interior of the textureless region, we will have more confidence in detecting visual correspondence. Therefore, in our research, we intend to detect blob-like cues for visual correspondence. These cues are composed of blobs of pixels which have a texture cue on the boundary that is easily matched, and are likely to have smoothly varying disparities inside the blob, as shown in Figure 1.8. Figure 1.8(a) illustrates the right image of an artificial stereo image pair. Notice there is barely any texture on the parallelogram or in the background. Figure 1.8(b) shows the ground truths for the disparity of the input image. The brighter is the pixel in Figure 1.8(b), the larger is the disparity at that pixel. Figure 1.8(c) shows the semi-dense visual correspondence detected by our approach. The pixels in the parallelogram belong to the same semi-dense visual feature. Unlike that of [90] and [91], whose blob-like visual cues only have one disparity for each, our blob-like visual cues have a smoothly varying range of disparities. The edges of this semi-dense visual feature provide more confident information about the disparities of the region they surround. Since there is no visual cues detected on the boundary of the background or in its interior, our approach can not detect any dense visual features in the background.

1.3.4 Motivation and Contribution

Our work of finding semi-dense visual cue was inspired by the work of Veksler [91, 90]. For stereo and motion detection, the main difficulty is the ambiguity brought by texturelessness or repeating texture. No matter using local methods or global methods[2], it is hard to decide the correct disparity for the pixels inside the textureless

[2] Informally, in local methods, one pixel may look at the nearby pixels in a small neighbourhood to determine its own visual correspondence, and use this information only to make the decision. On the contrary, global methods take into account of all the pixels simultaneously.



(a) Right image of the input stereo pair

(b) Ground truth

(c) Blob-like visual cues

Figure 1.8: *A synthetic example of our blob-like visual cues: Figure (a) illustrates the right image of an artificial stereo pair. Notice there is little texture on both the parallelogram and the background. Figure (b) shows the ground truth for the disparity of the input image. The brighter is the pixel in Figure (b), the larger is the disparity at that pixel. Figure (c) shows the blob-like visual cues that our approach aims at. The pixels in the parallelogram belong to the same blob-like visual feature, where the disparities are in a smoothly varying range. The edges of this blob-like visual feature provide cues about the disparities of the region they surround. However, for the background, since we can not detect any visual cues on its boundary, there are no visual cues detected in the background. This is indicated by the black pixels.*

region, since more then one possible disparities will give little or even zero matching error, see Figure 1.7.

However, reliable cues can be found at the border of the textureless regions. This can be done by comparing the matching error with the strength of the intensity edges. If the matching error is less than the edge strength at a particular pixel by some threshold, as mentioned in [91], then this feature can be used as a trustable cue for the stereo and motion detection. The threshold of selecting the visual correspondence cues is picked by hand in the work of [91] and [90]. In our work, we developed a learning framework that select useful features to decide the locations of these visual cues automatically, which avoids the problem of selecting fixed threshold. More useful features other than the difference between matching error and edge strength can be easily included in our learning frame work. We construct a feature pool of 148 different features related to stereo and motion, such as matching cost (absolute and shifted difference between corresponding pixel pairs), the boundary condition feature (difference between matching cost and edge strength). Useful features for deciding visual correspondence are selected in a learning framework. This step results in a classifier that can detect sparse cues for stereo in the textured regions.

To get denser results, Veksler [91] uses graph cut algorithm to propagate the cues of boundary condition to the textureless regions. This is done for each disparity separately. We extend this approach to semi-dense cues for multiple disparities. We propose a grouping method which clusters the sparse cues detected in the previous step into several groups, based on their geometric location and associated disparities. The resulting feature groups contain sparse cues for consecutive disparities. And the sparse visual cues in the same group are also geometrically close to each other. We also use graph cut algorithm to propagate the information brought by the visual cues in the feature groups into textureless regions. This step is done for each feature group individually. And the results are semi-dense feature blobs for groups of consecutive disparities.

The final step is ambiguity resolving. One pixel inside the input image pair can belong to more than one semi-dense feature blobs generated in the previous step. To find the exact boundaries of these semi-dense cues, we apply α-expansion algorithm to resolve the ambiguities between the semi-dense feature blobs. This step produces nice boundaries between regions of the images with different groups of disparities. Textureless regions are also covered by our approach, which produces more evidence to the disparities of these regions.

1.4 Thesis Organization

In Chapter 2 we describe the energy optimization framework, followed by the introduction to the graph cuts algorithm, from the aspects of different constraints of vision problem. In Chapter 3, we restate the problem of rendering classic mosaics in a global energy optimization framework. We first formulate the energy formulation on the static mosaic problem. Then we provide the approach of optimizing the energy function. In Chapter 4, we will go through the details about rendering animated mosaics from real video. We show how we perform motion segmentation with occlusion handling, together with how we modify our static mosaic rendering methods for the animation problem. In Chapter 5, we will describe how we extract semi-dense visual cues from stereo image pair. We propose a method which extracts useful features in the textured regions of the input image for finding visual correspondence, followed by our clustering approach to grouping all the sparse features detected in the previous step. We apply graph cut algorithm to propagate our features into textureless regions and to resolve ambiguities. In Chapter 6, we conclude our work.

Chapter 2

Energy Optimization with Graph Cuts

In this chapter, we will describe the framework of energy optimization method based on graph cuts, which in the last several yeas has become a widely utilized optimization approach in computer vision and graphics. In Section 2.1 we will first present some labeling problems in Computer Vision and Graphics, and the energy formulation for solving them, followed by the common constrains for these problems. Next, we will go through some energy optimization approaches which are widely used, such as dynamic programming, simulated annealing and gradient descent.

In Section 2.2, we will give the overview of Graph Cut algorithm at first. Then, from the aspect of different constraints on the labeling problems to solve, we will go through variations of energy optimization methods based on graph cuts. In the last section, Section 2.3, we briefly describe some applications in vision and graphics which use Graph Cuts as a powerful optimization tool.

2.1 Energy Optimization in Vision and Graphics

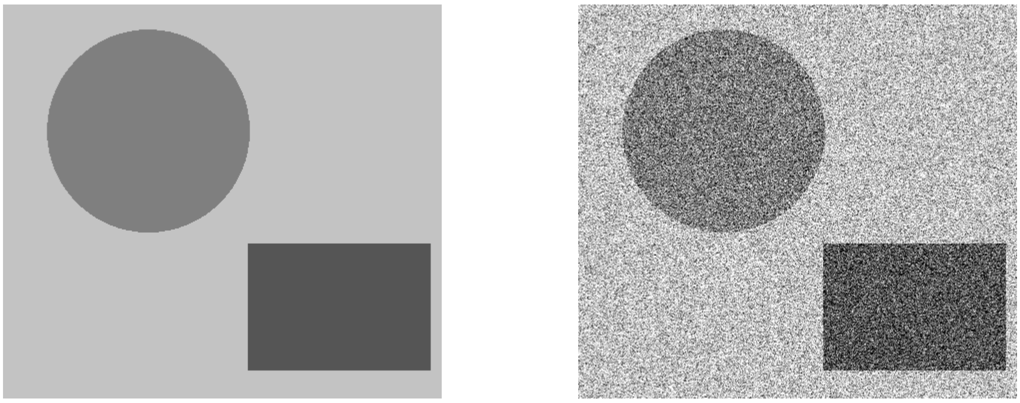
In this section, we will show how certain vision and graphics problems can be stated as labeling problems, and how energy functions can be formulated to evaluate the quality of the labeling. We will also describe the common constraints for these labeling problems, such as everywhere smooth constraint, piecewise smooth constraint and piecewise constant constraint. We will illustrate how these constraints can be encoded in the energy functions associated with the labeling problems. Moreover, we will introduce some commonly used algorithms in optimizing these energy functions.

2.1.1 Labeling Problem and a Common Form of Energy Functions

Many problems in vision and graphics can be naturally represented as labeling problems. For example, if one wishes to segment an object of interest from its background, then each pixel can be labeled as either background or foreground, denoted correspondingly by labels 0 and 1. This type of a problem with two labels is called a binary labeling problem.

Many other problems can be viewed as multi-labeling problems, for instance, the image restoration problem shown in Figure 2.1. Figure 2.1(a) illustrates the original image with a light background and two objects with constant intensity. Figure 2.1(b) is the corrupted image with Gaussian Noise ($\mu = 0$ and $\delta = 0.05$). A label set $\mathcal{L} = \{0, 1, 2, \ldots, 255\}$ represents all possible gray scale intensity levels. To restore the original image, a label $f_p \in \mathcal{L}$ will be assigned to each pixel p in the image. Let $f = \{f_p | p \in \mathcal{P}\}$ denote the collection of all pixel label assignments. If f is close enough to the original image, then the noise is reduced and original image is restored.

Thus the goal of a labeling problem is to assign to each pixel $p \in \mathcal{P}$ a label $f_p \in \mathcal{L}$,



(a) Original Image

(b) Noisy Image

Figure 2.1: *An Example of Image restoration: Figure (a) is the original image, where both the objects and the background are of constant intensity. Figure (b) shows the image corrupted with Gaussian noise ($\mu = 0$ and $\delta = 0.05$). $\mathcal{L} = \{0, 1, 2, \ldots, 255\}$ is the label set. The restoration task is to assign a label $l \in \mathcal{L}$ to pixels in the image so that the noise is removed.*

where $\mathcal{L}$ is a finite label set. A commonly used constraint is that the labels should vary smoothly almost everywhere. Labels are allowed to change drastically in a few places, and this is very important in order to preserve sharp discontinuities that may present.

Energy functions are formulated to evaluate the solutions to these labeling problems, under the constraint of smoothness over the image which comes from our prior knowledge as well as other constraints. We will discuss some common types of the smoothness constraints later. An additional constraint comes from the observed data. Let f be a labeling that assigns each pixel $p \in \mathcal{P}$ a label $f_p \in \mathcal{L}$. Usually the following energy function is formulated to evaluate the quality of f:

$$E(f) = E_{smooth}(f) + E_{data}(f) \tag{2.1}$$

Here, E_{smooth}, which is often called the smoothness term, measures the extent to which f is not smooth. E_{data}, usually called the data term, measures how pixels in $\mathcal{P}$ like the labels that f assigns them.

E_{data} is often formulated as

$$E_{data}(f) = \sum_{p \in \mathcal{P}} D_p(f_p)$$

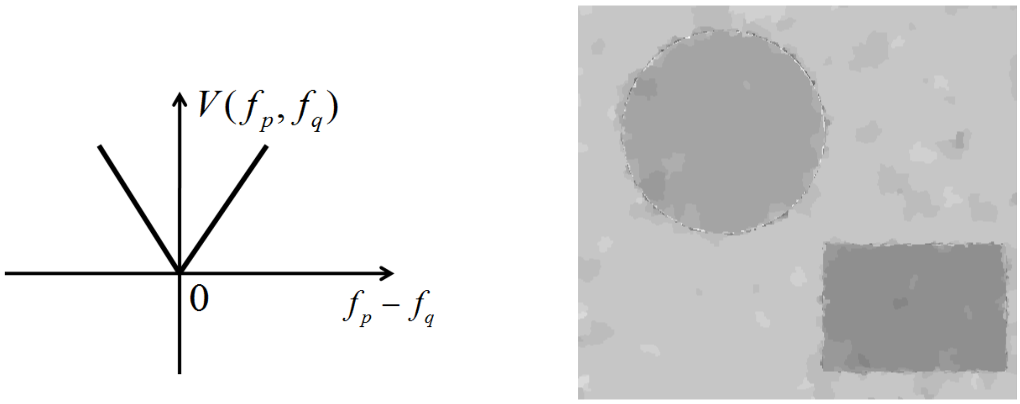
where D_p is the penalty for assigning pixel p the label f_p. For example, for binary segmentation, suppose we expect the object to have intensity of 40 and the background to have intensity of 200. Then we can set $D_p(1) = (I_p - 40)^2$ and $D_p(0) = (I_p - 200)^2$, where I_p is the intensity of pixel p, 1 is the object label, and 0 is the background label.

A typical choice of E_{smooth} is

$$E_{smooth} = \sum_{\{pq\} \in \mathcal{N}} V_{pq}(f_p, f_q) \tag{2.2}$$

Usually, $\mathcal{N}$ consists of pairs of immediately adjacent pixels, that is the interactions are given by the standard 4-connected grid. Longer-range interactions can be also included in $\mathcal{N}$. The choice of V_{pq} is critical. In most cases, it should make f vary smoothly in most places while preserving the discontinuities at object boundaries. How to set up V_{pq} depends on our prior knowledge about the smoothness of the desired labeling. There are several widely applied constraints on V_{pq}. Here we list three major types.

The everywhere smooth prior has a small penalty for labeling that is smooth everywhere. To encode the everywhere smooth prior in the energy function, V_{pq} should assign higher penalties for greater difference between labels f_p and f_q. For example, $V_{pq}(f_p, f_q) = |f_p - f_q|$, see Figure 2.2(a), is a everywhere smooth prior, because for large $|f_p - f_q|$, assigning labels f_p and f_q to neighboring pixels costs too much. If for all neighboring pixels p and q the label difference $|f_p - f_q|$ is small, then



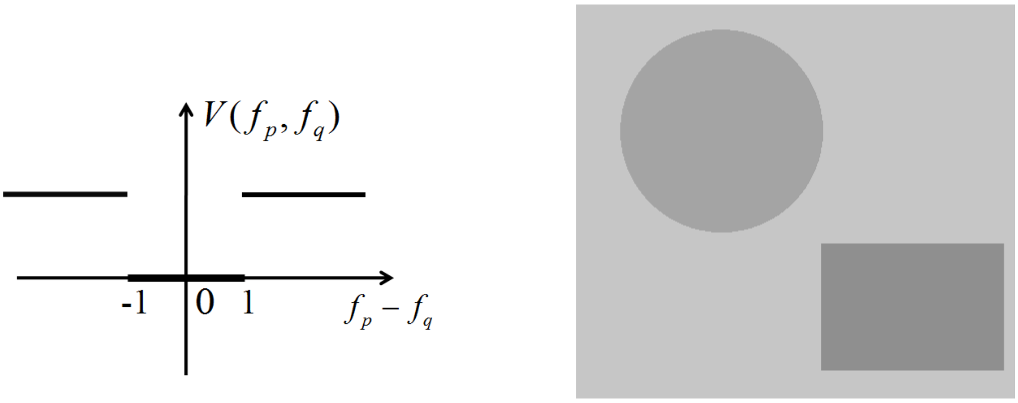
(a) $V_{pq} = |f_p - f_q|$

(b) Restored Image

Figure 2.2: *Everywhere Smooth Prior: Figure (a) shows an example of the everywhere smooth prior, the absolute distance $V_{pq} = |f_p - f_q|$. Figure (b) is the restored image for Figure 2.1(b). It is generated with $V_{pq} = |f_p - f_q|$. Notice the image is over-smoothed at the boundaries of the circle and the rectangle. We histogram corrected the image in Figure (b) to illustrate the over-smoothing effect.*

this labeling will not be penalized too much. Therefore, the optimal labeling is not likely to have drastic changes between any pair of neighbouring pixels. The problem with the everywhere smooth prior is that for most labeling problems in vision and graphics, there should be some tolerance for sharp label changes on the boundaries of the objects, so that they can restore the data around the discontinuities correctly. Otherwise, these discontinuities are over-smoothed, see Figure 2.2.

Piecewise Constant Prior: Some computer vision and graphics problems require different labels for neighouring pixels at object boundaries, but within the object, the pixels should have same labels. Consider the example of the image restoration problem illustrated in Figure 2.1. Both the objects and the background have constant intensities. We need to preserve the discontinuities at the boundaries of these objects, that means we should allow sharp changes at the edges of the regions of background and the objects. However, within each object and the background, the labels should be constant. For instance, inside the the round object, the optimal labeling should



(a) $V_{pq} = w_{pq} \times T(f_p \neq f_q)$

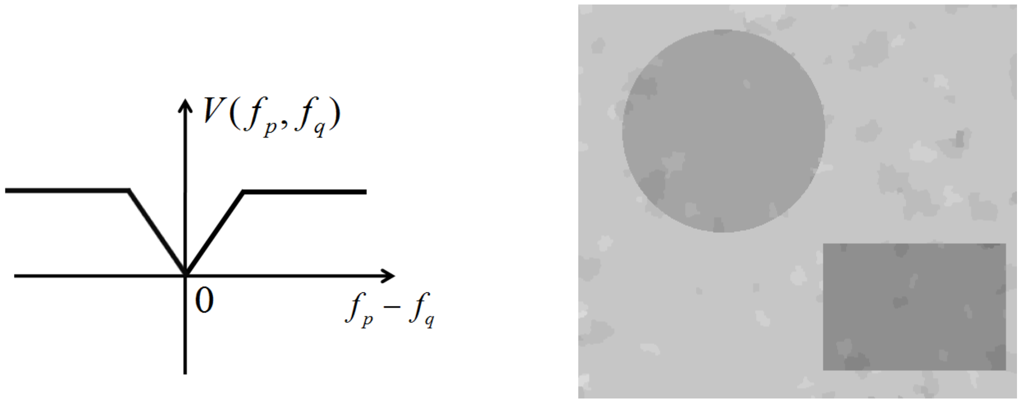
(b) Restored Image

Figure 2.3: *Piecewise Constant Prior: Figure (a) is the Potts smoothness term,* $V_{pq} = w_{pq} \times T(f_p \neq f_q)$*. Figure (b) shows the image restoration result for Figure 2.1(b) produced with Potts smoothness term. It is the best solution for this particular example, comparing with Figure 2.2(b) and Figure 2.4(b).*

assign label 127 to every pixel, which is the true intensity of the circle. The piecewise constant prior assigns a low cost to such labeling.

A good choice for piecewise constant prior is the so called Potts Model, which is $V_{pq}(f_p, f_q) = w_{pq} \cdot T(f_p \neq f_q)$, see Figure 2.3(a). Here $T(f_p \neq f_q)$ is 1 if $f_p \neq f_q$ and 0 otherwise. Potts smoothness term is most appropriate for non-ordered labels or when the number of labels is small. Figure 2.3 shows the image restoration result generated with piecewise constant prior. For this particular example, since both the background and the objects are of constant intensity, piecewise constant prior is a more appropriate choice compared to the everywhere smooth prior.. Therefore, comparing with the results generated with everywhere smooth prior, see Figure 2.2(b) and piecewise smooth prior, see Figure 2.4(b), piecewise constant prior produces much better solution to the image restoration problem.

Piecewise Smooth Prior: There are also computer vision and graphics problems that require different labels for neighouring pixels at object boundaries, and within



(a) $V_{pq}(f_p, f_q) = \min(K, |f_p - f_q|)$

(b) Restored Image

Figure 2.4: *Piecewise Smooth Prior: Figure (a) shows the truncated linear smoothness term,* $V_{pq}(f_p, f_q) = \min(K, |f_p - f_q|)$*. Figure (b) shows the image restoration result for Figure 2.1(b) produced with truncated linear smoothness term. The result in Figure (b) is better than that of Figure 2.2(b), however, for this example, the image restoration result generated with Potts smooth term, see Figure 2.3(b) is the best among them.*

each object or region, the labels should vary smoothly. For instance, for stereo, at the boundaries of the objects, one should alow the disparity labels to change drastically. But within each object, the disparity of the pixels should be able to vary smoothly to form a smooth surface, and also restore the data. To make a discontinuity preserving V_{pq}, one typically "truncates" the function by setting $V_{pq}(f_p, f_q) = \min(K, |f_p - f_q|)$, where K is the truncation constant, see Figure 2.4(a). In this way, the penalty for a discontinuity is never larger than K, and sharp discontinuities can be created, see Figure 2.4.

Many discontinuity preserving energy functions have been proposed in [33, 57, 89]. In addition to specifying smoothness assumptions, the choice for E_{smooth} also dictates the choice of optimization algorithm and as the result the quality of optimization that can be achieved. For example, if $V_{pq}(f_p, f_q) = |f_p - f_q|$, then the energy in Equation 2.1 can be optimized exactly [44]. In general, any convex choice for V_{pq} will lead to an energy that can be optimized exactly with graph cuts [44]. However, the truncated linear and the Potts V_{pq} lead to an energy which is NP-hard to optimize [53], but there are approximation algorithms with different quality guarantees, see [12]. We will describe some of these algorithms in Section 2.2.

To solve these labeling problems related to the vision and graphics applications, energy functions presented in Equation 2.1 must be optimized. Unfortunately, many of these energy functions are not convex and they have many local minima. Worse still, the labeling f usually has dimension $|\mathcal{P}|$ to the power of $|\mathcal{L}|$, which is thousands as $\mathcal{P}$ is the set of pixels in an image. The computational cost of getting global minima of these energies represented in Equation 2.1 is enormous. It forces researchers to find efficient but approximate optimization algorithms.

2.1.2 Optimization algorithms

Over the years, researchers have been working on finding good energy optimization algorithms. In this section, we will briefly describe some commonly used optimization methods.

Gradient descent is an algorithm which can find a local a minimum of a function. The idea behind gradient descent is taking steps in the direction of negative gradient of the function. This makes the function decrease fastest and then reach a local minimum. There are two problems with gradient descent. Firstly, the convergence of this approach is not guaranteed. The choice of step width is the critical issue of gradient descent algorithm. When the steps are too large, the function may not converge. If the step width is too small, it will take too many steps to reach a local minima, which results in an enormous computational cost. Worse still, there is no general guidance to find a good step width and make sure the gradient descent algorithm converges in an acceptable time. The biggest drawback of gradient descent is that it can only reach a local minimum of the energy function. The global minimum, which is usually more important, may be far away from this local minimum obtained with gradient descent. Due to these shortcomings of gradient descent, researchers have developed other energy optimization algorithms.

Simulated annealing was first proposed by Kirkparick et al. in [73]. The name and inspiration of simulated annealing come from annealing in metallurgy. The physical annealing is a process which uses heating and controlled cooling process to find the low energy states of a material. The heating process prevents the atoms from being stuck in a low energy state which is not the optimal state. The slow cooling process enables the atoms to wander through many energy states which have low energy, and find a better solution.

The simulated annealing algorithm is an analogue to the physical annealing process. It is the only general-purpose method to find, or in most cases, to approximate the best solution to the energy optimization problems. It starts from a random initial labeling. At each iteration of simulated annealing, the labeling at one particular pixel is changed locally and randomly. The annealing schedule is controlled by the temperature parameter, T, which is set to be a high value and gradually decreased during the annealing process. If the energy is decreased with the local change, then the new labeling is accepted. In case of energy increasing, the probability of accepting the new labeling is then determined partially by the temperature parameter. The higher the temperature, the larger is the probability of accepting the change. The probability of accepting a new labeling with higher energy than the current labeling prevents the algorithm from being stuck at a local minimum. When the temperature is high, the simulate annealing algorithm performs more like a random walk. As the temperature decreases during the annealing process, the algorithm tends to look for a local minimum. There is a certain order to visit the pixels.

The temperature parameter is decreased according to a cooling schedule. If the cooling schedule is optimal, simulated annealing will obtain a global minimum. However, it is prohibitively expensive to perform simulated annealing under the optimal schedule. Therefore, the sub-optimal schedules are often used in practice to reduce the running time. In this case, only a local minimum of the energy function can be found with simulated annealing.

Dynamic programming can be applied to energy optimization when the energy function has a very restricted form. It solves a complex problem by breaking it down into simpler subproblems. The solution of a given optimization problem must be able to be obtained by the combination of optimal solutions of the subproblems. And any recursive algorithm solving the original problem should solve the subproblems by breaking them down into simpler problems, rather than generating new subproblems of same complexity. Therefore, dynamic programming requires that the problem to be solved must have one-dimensional structure rather then loopy structures. This includes some important cases, such as snakes [48]. The two dimensional energy functions that arise in low level vision can not, in general, be solved efficiently via dynamic programming except some special cases [28].

Graph cuts can be applied to find the global minimum for certain two-dimensional energy functions and find good approximate solutions of certain other energy functions. In the following section, we will describe how graph cut algorithm is used to optimize different energies.

2.2 Graph Cuts

The graph cut algorithm has been widely used in Computer Vision and Graphics for the purpose of energy optimization. It has been proved to be successful in many situations. In this section, we will describe the graph cuts algorithm and how it is used to optimize different energies, together with some examples in vision and graphics which use graph cut as a powerful optimization tool.

2.2.1 Overview of graph cuts

Let $G = \langle V, E \rangle$ be a weighted graph. Here V is a set of vertices and E is the set of edges which connect vertices in V. Every edge $e \in E$ is assigned a non-negative weight w_e. There are two special vertices in V, they are called *terminals* and identified as *source* s and *sink* t. A cut C is a subset of edges $C \subseteq E$ such that when we remove C from G, V is partitioned into two disjoint sets S and T such that $s \in S$ and $t \in T$, where $T = V - S$.

The cost of a cut C is defined as:

$$|C| = \sum_{e \in C} w_e$$

The cut with minimum cost is called the minimum cost cut. There are two main approaches to solving Min-Cut/Max-flow problem for the two-terminal graphs. In Cormen et al. [15], an "augmenting path" strategy is described to compute the minimum cut of a graph. An alternative approach named "push-relabel" is presented in Goldberg and Tarjan [31] to solve the minimum cut problem. Theoretically, the computational cost of minimum cut algorithms is a low order polynomial. Boykov and Kolmogorov present a comparative study of standard minimum cut algorithms in [13]. They also provide a new minimum cut algorithm in [13] and in practice their algorithm is significantly faster than other standard algorithms for graphs arising from computer vision problems. For our implementation, we use the algorithm in [13], and its implementation is provided by the authors.

2.2.2 Energy optimization with graph cuts

Energies which can be optimized through graph cuts usually have the following form:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{pq\} \in \mathcal{N}} V_{pq}(f_p, f_q) \tag{2.3}$$

where $\mathcal{N}$ consists of pairs of immediately adjacent pixels. These interactions are given by either the standard 4-connected grid or longer-range interactions.

For binary labeling f, that is $f_p \in \{0,1\}$, to optimize its energy $E(f)$, a graph $G = \langle V, E \rangle$ whose minimum cut represents the optimal $E(f)$ can be constructed in case when $E(f)$ is regular [53]. Let each non-terminal node $p \in \mathcal{V}$ represent a pixel in $\mathcal{P}$. Therefore,

$$V = \mathcal{P} \cup \{s,t\}$$

where s is the source terminal and t is the sink terminal. Each non-terminal node p is connected to source s and sink t, by terminal links (*t-link*) e_{sp} and e_{pt}. Each pair of interacting pixels (pq) are connected by node links (*n-link*) e_{pq}. Thus

$$E = \{e_{pq} : \{pq\} \in \mathcal{N}\} \cup \{e_{sp}, e_{pt} : p \in \mathcal{P}\}$$

Suppose that $V_{pq}(f_p, f_q)$ is given by the Potts Model, that is $V_{pq} = w_{pq} \cdot T(f_p \neq f_q)$. In this case, the graph construction is particularly simple. Recall that a graph cut partitions all the nodes into two disjoint sets, $\mathcal{S}$ and $\mathcal{T}$, such that the source $s \in \mathcal{S}$ and the sink $t \in \mathcal{T}$. If p is in $\mathcal{S}$, it is associated with label 0, otherwise it is associated with label 1. Thus a graph cut will give a labeling which will assign a label to each pixel. To let the cost of the minimum cut correspond to the minimum energy, the weight assigned to each edge is:

edge	weight	for
e_{sp}	$D_p(1)$	$p \in \mathcal{P}$
e_{pt}	$D_p(0)$	$p \in \mathcal{P}$
e_{pq}	w_{pq}	$\{pq\} \in \mathcal{N}$

Figure 2.5 shows how to construct a graph for binary labeling problems in case of Potts $V_{pq}(f_p, f_q)$. After building such graph, the energy $E(f)$ can be optimized by computing the min cut of this graph. Graph cuts can be used to optimize energy

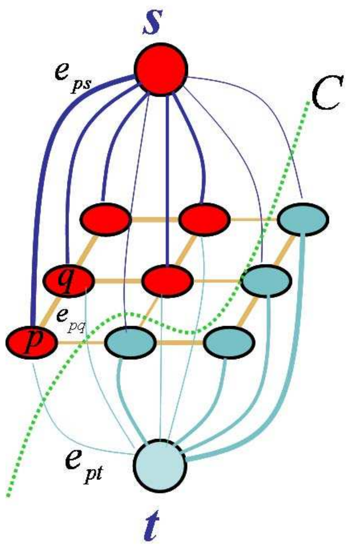


Figure 2.5: *Graph cut demonstration on a 3 × 3 graph. The label set is $\{0, 1\}$. Source terminal s stands for label 0 and sink t stands for label 1. Each pixel in this graph is connected to terminals by t-links. The thickness of the t-links represents how pixels like the corresponding label. For example, pixel p is linked to s by a thick edge e_{sp} and a thin edge e_{pt} connect p to t. Therefore p is more likely to be assigned label 0. Vertices pq form an interacting pair. The weight of the n-link between p and q is large when pq are likely to have the same label. A cut C segments the vertices into two disjoint subsets, represented by different colors of the vertices. The label of each pixel can be obtained by finding the subset which contains the pixel. This constructions was first given by Greig et al. [32]*

functions which are more general than Potts binary energies. Kolmogorov and Zabih [53] describe the conditions on binary energy functions that can be optimized exactly with a graph cut. A function of two variables $E(x_1, x_2)$ is regular on the variable value set $\{0, 1\}$ if it satisfies:

$$E(0,1) + E(1,0) \geq E(0,0) + E(1,1) \tag{2.4}$$

The energy defined in Equation 2.3 has $|\mathcal{P}|$ variables, and it can be viewed as a sum of several two-variable ($V_{pq}(f_p, f_q)$) and single-variable ($D_p(f_p)$) functions. If all of these functions are regular, then $E(f)$ is regular, see [53]. To check the regularity of the energy functions, we only have to examine the interaction penalty V_{pq}. According to [53], all the one variable functions are regular. Therefore, if $V_{pq}(f_p, f_q)$ satisfies

$$V_{pq}(0,0) + V_{pq}(1,1) \leq V_{pq}(1,0) + V_{pq}(0,1)$$

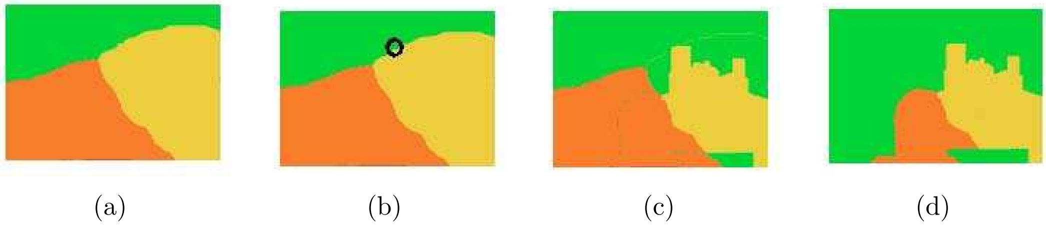
then $E(f)$ is regular, and it can be optimized by graph cuts.

Multi-labeling is more general in vision and graphic problems. However, only a few energy functions can be optimized exactly with graph cuts for multi-labeling problems. Ishikawa [44] proves that when $V_{pq}(f_p, f_q)$ is convex, for example $V_{pq}(f_p, f_q) = |f_p - f_q|$, the energy defined in Equation 2.3 can be optimized exactly with graph cuts. Boykov et al. [12] show that the Potts model ($V_{pq}(f_p, f_q) = w_{pq} \cdot T(f_p \neq f_q)$) and the truncated model ($V_{pq}(f_p, f_q) = \min(K, |f_p - f_q|)$), as defined in Section 2.1, are NP-hard to be optimized. Because convex energy functions, such as $V_{pq}(f_p, f_q) = |f_p - f_q|$, are not discontinuity preserving, the Potts model and truncated enenrgy functions are often used to obtain a better labeling. There are two methods, the expansion move and the swap move algorithms, for approximating the energy which is NP-hard to optimize exactly, such as Potts model and truncated energy function. Both expansion and swap algorithms find a strong local minimum of the energy function. In the following paragraphs, we will go through the swap and expansion algorithms based on graph cuts and analyze their optimality properties from the aspects of different constraints.

The expansion move can be used to optimize energy functions where V_{pq} is a metric [12]. V_{pq} is metric if it satisfies all the metric constraints, which are defined by the followings:

$$V_{pq}(\alpha, \alpha) = 0 \tag{2.5}$$

$$V_{pq}(\alpha, \beta) = V_{pq}(\beta, \alpha) \geq 0 \tag{2.6}$$



(a) (b) (c) (d)

Figure 2.6: *From left to right: (a) original labeling, (b) labeling within one standard move (the changed pixel is highlighted by a black circle), (c) labeling within one green-yellow swap, (d) labeling within one green expansion*

$$V_{pq}(\alpha, \beta) \leq V_{pq}(\alpha, \gamma) + V_{pq}(\gamma, \beta) \tag{2.7}$$

Example of interaction terms which are metric include the Potts Model $V(\alpha, \beta) = K \cdot T(\alpha \neq \beta)$ and the truncated absolute distance $V(\alpha, \beta) = \min(K, |\alpha - \beta|)$, as defined in the previous section.

For each labeling f defined on pixel set $\mathcal{P}$, we can define a set of moves M_f that are allowed to be taken from f. For any move $f' \in M_f$, f' is another labeling on the pixel set $\mathcal{P}$ which can be obtained by changing the labels of a subset pixels $P \subseteq \mathcal{P}$. Let $H(f, f') = |\{p|p \in \mathcal{P}, f_p \neq f'_p\}|$. That is $H(f, f')$ counts the number of pixels for which f is different from f'. A *standard* move $f' \in M_f$ is a move where $H(f, f') \leq 1$. Thus at most only one pixels changes its label from f, see Figure 2.6(b).

Given a labeling f and a label α, an α-expansion move only allows pixels whose current label in f is not α to change to α in the new labeling f'. That is a move f' is called an α-expansion if $f_p = f'_p$ whenever $f'_p \neq \alpha$. For the α-expansion algorithm, M_f is defined as the collection of α-expansions for all labels $\alpha \in \mathcal{L}$. Figure 2.6(d) shows an example of α-expansion.

In the expansion algorithm, optimization starts with an initial labeling f. For each iteration, a label α is chosen from the label set $\mathcal{L}$. Then the α-expansion move which decreases the energy $E(f)$ most will be saved and taken as the input labeling for the next iteration. To find the optimal α-expansion move, a certain graph $G = \langle V, E \rangle$ is constructed and the minimum cut of this graph G is computed.

The graph construction for G is as follows. The vertex set V consists of three parts. First, there are two terminals α and $\overline{\alpha}$ included in V, where $\overline{\alpha}$ represents the current labels in f and α means the new label to be expanded. Second, every pixel in $\mathcal{P}$ also presents in V. At last, for each pair of neighboring pixels $\{p, q\} \in \mathcal{N}$ where $f_p \neq f_q$, an *auxiliary* node $a_{\{p,q\}}$ is created. Notice the auxiliary nodes only present between pixel pairs with different labels in the old labeling f. Thus the set of vertices is

$$V = \{\alpha, \overline{\alpha}\} \cup \mathcal{P} \cup \{a_{\{p,q\}} | \{p, q\} \in \mathcal{N}, f_p \neq f_q\}$$

The edge set E also consists of t-links and n-links. For neighbouring pixels whose labels are the same in f, that is $\{p, q\} \in \mathcal{N}$ and $f_p = f_q$, there are n-links e_{pq} connects them. For pixel pairs where $\{p, q\} \in \mathcal{N}$ and $f_p \neq f_q$, there are three edges $\varepsilon_{p,q} = \{e_{p,a}, e_{a,q}, t_a^{\overline{\alpha}}\}$ created. Here $a = a_{pq}$ is the corresponding auxiliary node between p and q. The edges $e_{p,a}$ and $e_{a,q}$ connect pixels p and q to a_{pq} and t-link $t_a^{\overline{\alpha}}$ connects the auxiliary node a_{pq} to the terminal $\overline{\alpha}$. For each pixel $p \in \mathcal{P}$, there are two t-links t_p^{α} and $t_p^{\overline{\alpha}}$ connect it to the terminals α and $\overline{\alpha}$ respectively. Thus the edge set is:

$$E = \{t_p^{\alpha}, t_p^{\overline{\alpha}} | p \in \mathcal{P}\} \cup \{\varepsilon_{p,q} | \{p, q\} \in \mathcal{N}, f_p \neq f_q\} \cup \{e_{\{p,q\}} | \{p, q\} \in \mathcal{N}, f_p = f_q\}$$

Figure 2.7 shows an illustration of constructing graph for α-expansion.

The weights assigned to the edges are:

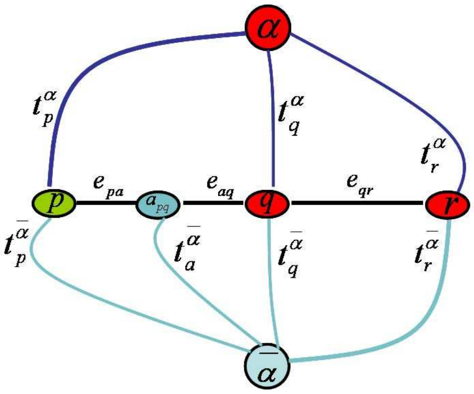


Figure 2.7: *An example of graph for α-expansion. The pixels set is $\mathcal{P} = \{p, q, s\}$. Pixel p and q have different labels in the current labeling, indicated by the different colors on the nodes. An auxiliary node a_{pq} is introduced between p and q.*

edge	weight	for
$t_p^{\overline{\alpha}}$	∞	$p \in \mathcal{P}, f_p = \alpha$
$t_p^{\overline{\alpha}}$	$D_p(f_p)$	$p \in \mathcal{P}, f_p \neq \alpha$
t_p^{α}	$D_p(\alpha)$	$p \in \mathcal{P}$
$e_{p,a}$	$V(f_p, \alpha)$	$\{p, q\} \in \mathcal{N}, f_p \neq f_q$
$e_{a,q}$	$V(f_q, \alpha)$	
$t_a^{\overline{\alpha}}$	$V(f_p, f_q)$	
$e_{p,q}$	$V(f_p, f_q),$	$\{p, q\} \in \mathcal{N}, f_p = f_q$

A minimum cut on graph $G = \langle E, V \rangle$ will find the optimal α-expansion move. A pixel $p \in \mathcal{P}$ is assigned label α if the optimal cut separates it from the terminal node α. If the optimal cut on graph $G = \langle E, V \rangle$ separates pixel p from node $\overline{\alpha}$, then its old label will be preserved. To make the energy of the optimal cut represent the energy of the resulting labeling correctly, it is required that the weight of edge $e_{q\alpha}$ is less than the sum of weight of edges $e_{\alpha p}$ and $t_{\alpha}^{\overline{\alpha}}$. That is:

$$V(f_p, \alpha) + V(f_p, f_q) \geq V(f_q, \alpha)$$

Therefore, α-expansion can only be applied on energy functions which are metric, see

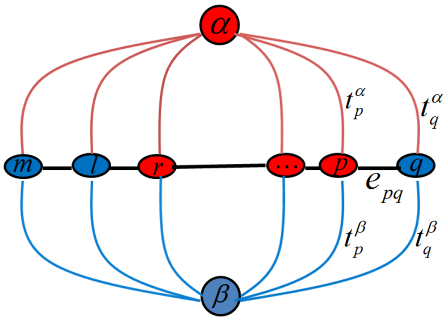


Figure 2.8: *An example of α-β swap. $\mathcal{P}_\alpha = \{r, \ldots, p\}$ is the set of pixels whose current label is α, indicated by red circles. $\mathcal{P}_\beta = \{m, l, q\}$ is the set of pixels whose current label is β, coloured with blue. All the pixels are connected to the two terminal nodes associated with label α and β. Let $\mathcal{P}_{\alpha\beta} = \mathcal{P}_\alpha \bigcup \mathcal{P}_\beta$. For any pixel $p \in \mathcal{P}_{\alpha\beta}$, the weights of the t-links between p and the terminals α and β are: $t_p^\alpha = D_p(\alpha) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\alpha, f_q)$ and $t_p^\beta = D_p(\beta) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\beta, f_q)$. The neighbouring pixels are connected with n-links. The weight of the n-link e_{pq} between any pair of pixels p and q is $V_{pq}(\alpha, \beta)$.*

Equation 2.5, 2.6, 2.7. The algorithm cycles in random order in the label set $\mathcal{L}$ until convergence.

Swap Move is another approximation algorithm for multi-labeling. Given a labeling f and a pair of labels α and β, a move f' is called a $\alpha - \beta$ swap if $f_p = f'_p$ whenever $f_p \neq \alpha$ and $f_p \neq \beta$. This move only allows the differences between f and f' where some pixels that were labeled α in f are now labeled β in f', and some pixels that were labeled β in f are now labeled α in f'. Figure 2.6(c) shows an example of $\alpha - \beta$ swap. In each iteration of the swap move algorithm, a pair of labels α and β are selected randomly from label set $\mathcal{L}$, and the optimal $\alpha - \beta$ swap is found. This step is performed by finding the minimum cut on a certain graph $G = \langle V, E \rangle$.

Let vertices set V contains the source terminal s and the sink terminal t. The source s is associated with label α and sink t is associated with label β. Let $\mathcal{P}_{\alpha\beta}$ be the set of all pixels for which either $f_p = \alpha$ or $f_p = \beta$. Pixels in $\mathcal{P}_{\alpha\beta}$ are set to be non-terminal nodes in V. Each nonterminal node is then connected to the source with a t-link weighted

$$D_p(\alpha) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\alpha, f_q)$$

Each nonterminal node is also connected to the sink with a t-link weighted

$$D_p(\beta) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\beta, f_q)$$

Each pair of neighboring nodes $\{p, q\}$ in $\mathcal{P}_{\alpha\beta}$ are connected by a n-link weighted $V_{pq}(\alpha, \beta)$. The minimum cut $C = (\mathcal{S}, \mathcal{T})$ on graph G corresponds to the optimal $\alpha - \beta$ swap. If $p \in \mathcal{S}$, f'_p is set to be α, otherwise $f'_p = \beta$. For pixels which are not in $\mathcal{P}_{\alpha\beta}$, $f'_p = f_p$. Figure 2.8 illustrates the structure of the graph for $\alpha - \beta$ swap. The swap move algorithm cycles randomly inside the label set $\mathcal{L}$ until convergence.

A restriction for swap move algorithm is that the energy function must be semi-metric [12]. An energy function is semi-metric if it satisfies Equation 2.6 and 2.5. The truncated quadratic $V(\alpha, \beta) = \min(K, |\alpha - \beta|^2)$ is an example of semi-metric energy functions. It encourages the labeling to have several regions in which pixels in the same region have similar labels.

The optimality of α-expansion has been proved by Boykov et al. [12]. Although it is NP-hard to optimize the energy function whose interaction term is not convex, Potts Model $V_{pq}(f_p, f_q) = w_{pq} \times T(f_p \neq f_q)$ for example, a local minimum when expansion moves are allowed is within a know factor of the global minimum. This factor, which can be as small as 2, depends on V. Let

$$c = \frac{\max_{\alpha \neq \beta \in \mathcal{L}} V(\alpha, \beta)}{\min_{\alpha \neq \beta \in \mathcal{L}} V(\alpha, \beta)}$$

be the ratio of the greatest non zero value of V to the least non zero value of V, where L is the set of all labels. Since $V(\alpha, \beta) \neq 0$ for $\alpha \neq \beta$ due to the metric properties defined in Equation 2.5 and 2.6, this ratio c is well defined. Let $\hat{f}$ be a local minimum when the expansion moves are allowed and f^* be the global optimal solution. Then we have $E(\hat{f}) \leq 2cE(f^*)$.

A local minimum given by swap move can be arbitrarily far from the global minimum. However, since it only requires the energy functions to be semi-metric (see Equation 2.5 and 2.6), the swap move can be applied to a wider range of energy functions than that of expansion move.

2.3 Applications of graph cuts optimization

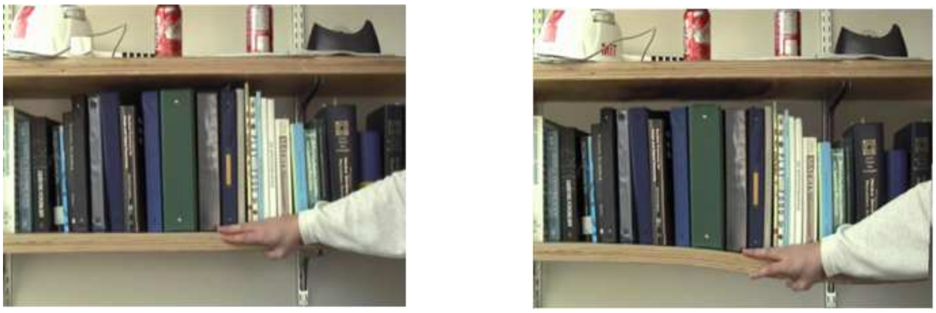
In this section, we describe a specific application of graph cut optimization which is most related to this thesis, motion magnification. We describe the problem, how it is represented as labeling problem and formulate energy functions to solve it. In the last part, we will briefly introduce some other applications which are based on graph cut algorithm.

2.3.1 Motion Magnification

Motion magnification is a virtual microscope which amplifies subtle motions in a video sequence. It visualize the deformation of the regions whose motions are invisible for human eyes. Figure 2.9 illustrates an example of motion magnification. Comparing with the frame from the input video sequence, the deformation of the bookshelf is amplified so that it can be visible for human.

The main difficulty of motion magnification is motion layer segmentation. To amplify the motions of the subtle regions, the first step is to detect these regions as well as their motion vector. However, motion segmentation is not a well solved problem yet. Liu et al. [61] proposed a motion magnification approach based on graph cuts.

Their basic idea is first detecting sparse feature points which translate from one frame to the next. The second step is to estimate the dense motion vector for all the



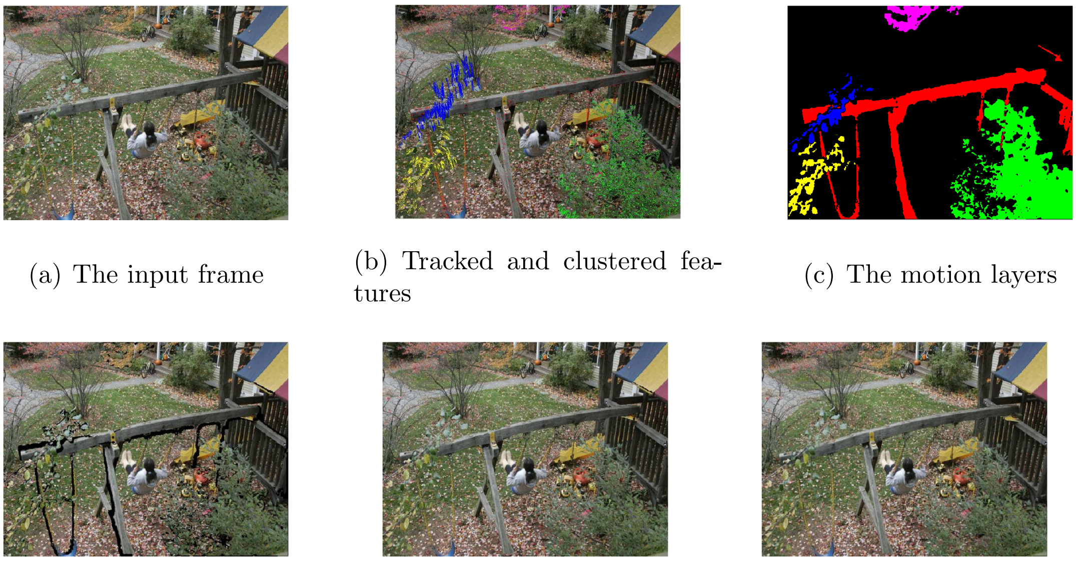
(a) Input Frame (b) Motion Magnification on Bookshelf

Figure 2.9: *An example of motion magnification. Figure (a) is from the input video sequence where the bookshelf is pressed. The deformation of the bookshelf is inevitable to human vision. Figure (b) is produce by the motion magnification approach proposed by Liu et al. [61]. Compared with Figure (a), it clearly reveals the deformation of the bookshelf.*

pixels and segment the pixels into several motion layers. This step is formulated as a multi-labeling problem and solved with graph cuts. Finally, the motion of the user selected layers is magnified. The "hole" revealed by the amplified motions are filled in by texture synthesis. Figure 2.10 shows the main procedure for the work of Liu et al. [61].

The first step of Liu et al. [61] is detecting and tracking the robust feature points inside the video sequence. It starts with detecting corners with Harris corner detector. Then the trajectories of these corner features are computed by mapping each feature point from the reference frame to each other frames, based on the minimum sum of squared differences (SSD) over a small path. They develop a learning approach based on Expectation Maximization (EM) algorithm to compute the weight map that characterizes how pixels should be weighted in the SSD similarity measures. This allows them to track the features more robustly through occluded regions.

The next step is to cluster all the trajectories into several clusters, which forms the basis of motion layer segmentation. The entire motion trajectory of each feature point, rather than motions between two nearby frames, is used in the clustering step.



(a) The input frame

(b) Tracked and clustered features

(c) The motion layers

(d) Holes generated by amplifying the motions

(e) After texture synthesis

(f) After user modification to the segmentation map in Figure (c)

Figure 2.10: *The process of Liu et al. [61]. These images are from Liu et al. [61].*

This is because the goal of the motion clustering is to group together motions with a common cause, even though the motions may be in different directions. Normalized correlation between the trajectories are used to measure the similarity between them. Trajectory clustering is done with spectral clustering. The number of clusters is provided by the user. Figure 2.10(b) shows the result for computing and clustering the trajectories.

In the third step, every pixel of the image is assigned to one of the motion clusters detected in the second step. This is done in a multi-labeling framework. The label set $\mathcal{L}$ is the set of all motion layers. The data term consists two parts, motion likelihood and colour likelihood. The motion likelihood measures the variance of the colour of the pixels along the motion trajectory. If the motion label fits the pixel well, then the colour of the pixel should be stable when moving along with the motion trajectory associated with the label. Colour likelihood uses a Gaussian mixture model to compute the likelihood for each pixel belongs to the same motion layer. The intuition behind the colour likelihood model is that pixels which move with the similar motion usually belong to the same object. Therefore, they have similar colour information. The smoothness term is the Potts model, where the weight of the smoothness term is calculated based on the edge strength between the neighbouring pixels. The stronger the edge is between the neighbouring pixel pair, the less the smoothness term is. Graph cut algorithm [12] is used to optimize the energy function. Figure 2.10(c) shows the result of motion segmentation.

The fourth step is to amplify the small motion in the motion layer selected by the user, and then displace the pixels inside that layer according to the magnified motion. Since motion magnification will reveal occluded regions, as shown in Figure 2.10(d), texture synthesis is used to fill in the gap space generated by motion magnification. Figure 2.10(e) shows the image in-painting result. At last, user interaction is applied to correct the small errors made in motion segmentation, see Figure 2.10(f).

2.3.2 Other applications of graph cuts optimization

Besides motion magnification, graph cuts has also been applied as energy optimization tools in many other vision and graphic problems. Stereo [12, 52, 49] is one of the most important problems in computer vision. The task of stereo algorithm is to construct a 3D structure of a scene, given two or more photos from different views of the same scene. To solve this problem, the typically used approach is to find the correspondence between views and thereby find the disparity of each point in the scene. By viewing the disparities as labels, stereo problems are formulated as labeling problems and the optimal labeling can be computed by graph cuts. The 3D reconstruction problem can be done by viewing the problem as a multi labeling problem and graph cut is used to obtain the optimal labeling.

For motion segmentation [46, 100, 98], the task is to cluster pixels in clouds that undergo similar motions. This problem can be naturally represented by a labeling problem and graph cut can be used as the optimization tool.

Other vision problems, such as image restoration, texture synthesis, and image stitching can also be solved by graph cuts, see [12, 55, 1].

Chapter 3

Global Formulation of Static Mosaics

In this chapter, we formulated the problem of classic mosaic generation in a global optimization framework, and designed an energy function encoding the properties that lead to perceptually pleasing mosaics. Our global optimization framework offers a more principled approach than previous work, which is mostly based on heuristics. Comparing with the preliminary version of our static mosaic rendering method in [63], we formulate our energy function in a new way so that all the constraints for a good classic mosaic (edge alignment, edge avoidance, tight tile packing) are encoded in one energy function. The restating of the mosaic rendering process makes it easier to comprehend the problem, since the relationship between all the mosaic constraints is revealed as a whole energy function. We also sped up our algorithm by 3 to 4 times, by improving the method in which we implement the tile packing algorithm in [63]. Most material in this chapter was published in [64].

3.1 Introduction

Classic mosaic is an ancient art form. It is composed of a large number of small tiles with regular shapes, such as rectangles. For thousands of years, durable mosaics were used for decoration purpose.

Studying the work of mosaic artists, two main properties of a visually appealing mosaic emerge. First, mosaic tiles should be placed at orientations that emphasize perceptually important curves in an image. This is usually done by placing the tile sides parallel to the important curves. Recall the synthetic example presented in Section 1.2.2, the circle in Figure 1.5(a) is strongly emphasized in Figure 1.5(c) by placing the tile sides parallel to the circle boundary. If the tiles are placed at random orientations, the circle is emphasized much less, see Figure 1.5(b). Parallel tile placement is by far the most popular way to emphasize the boundaries, although other techniques for boundary emphasis are also possible.

Deciding which curves are perceptually important and are to be highlighted is frequently done with user interaction [36, 23]. While a human is the ultimate expert, it may be desirable to produce classic mosaics automatically. Since perceptually important curves tend to coincide with strong color edges in an image, some methods take advantage of the information provided by the gradient magnitude to automate the mosaic generation process. Explicit methods [10, 6] label the boundaries returned by an edge detection or an image segmentation algorithm as perceptually important. Such approach is simple but brittle, since edge detection and image segmentation frequently produce unappealing boundaries. Implicit methods [63, 5] use the gradient magnitude only as a soft cue for a possible boundary to emphasize. Whether a pixel with a high gradient magnitude gets emphasized or not depends on other factors, such as preferred tile orientations at other pixels, etc.

The second important mosaic principle is to maximize the number of tiles, while avoiding tile overlap as much as possible. This, combined with the first principle, means that tile orientations should align with important boundaries and vary smoothly in the image, since smoothly varying orientations allow a tighter packing of tiles. In Figure 1.5(c), tile orientations vary much more smoothly compared to that of Figure 1.5(b). Therefore the mosaic in Figure 1.5(c) has less gap space and is visually more appealing than that in Figure 1.5(b).

The following sections of this chapter is organized as follows. Section 3.2 discusses previous work, section 3.3 is a brief overview of our algorithm. Section 3.4 describes our energy function for mosaic generation, and section 3.5 gives a detailed description of our optimization strategy. Section 3.6 presents our experimental results and comparison with previous work. Section 3.7 is a summary.

3.2 Related Work

In this section, we will briefly introduce several significant methods on generating static mosaics. They are divided into two categories: orientation guideline methods and energy optimization methods. For the orientation guideline based methods [36, 23], the tile orientations are computed according to the orientation guidelines which are either provided by the user or detected using edge detection methods. Tile packing is usually done using heuristics. Another approach is to formulate the classic mosaic as a labeling problem and solve it by energy optimization methods [63, 5]. The energy optimization based approaches provide a new way to encode the edge information as a soft cue for computing tile orientations. Therefore, they eliminate both the user interaction of drawing the tile orientation guidelines and edge detection. The constraint of tight tile packing can also be encoded in the energy function, hence it can be approximately optimized and produce better results than the heuristic approaches.

3.2.1 Orientation Guideline Methods

The first successful work on simulating classic mosaics was introduced by Hausner [36]. The main process of Hausner's work can be divided into two stages: computing the tile orientation field and packing the tiles. Each of these two stage addresses one of the main constraints for classic mosaics (the edge alignment constraint and tight tile packing constraint), as we mentioned in section 1.2.2.

In the beginning, the user is required to input a set of curves, which are usually the edges of the objects inside the input image. A vector field of tile orientations is then computed based on the curves drawn by the user. When a square tile is placed at pixel (x, y), its side should be parallel to the orientation vector located at (x, y). To generate such an orientation vector field, Hausner first computes the distance transform of the input curves, which is also known as distance field. Let $E(x, y)$ be the input curve map, that is $E(x, y) = 0$ if (x, y) is located right on the curves marked by the user and $E(x, y) = 1$ otherwise. If $D(x, y)$ is the distance transform of $E(x, y)$, then for any pixel (x, y), $D(x, y)$ is the distance from (x, y) to its nearest curve point in $E(x, y)$. The gradient field of the distance map $D(x, y)$ for the input curve map $E(x, y)$ is then computed and used as the tile orientation field. Figure 3.1(b) shows the result of computed tile orientations.

The orientation field computed in this way will ensure that, at the points close to the object edges, the tile orientation vectors will be perpendicular to the nearest edge. Aligning the tiles with these guiding vectors will make the resulting mosaic highlight the shape of the objects inside the input image.

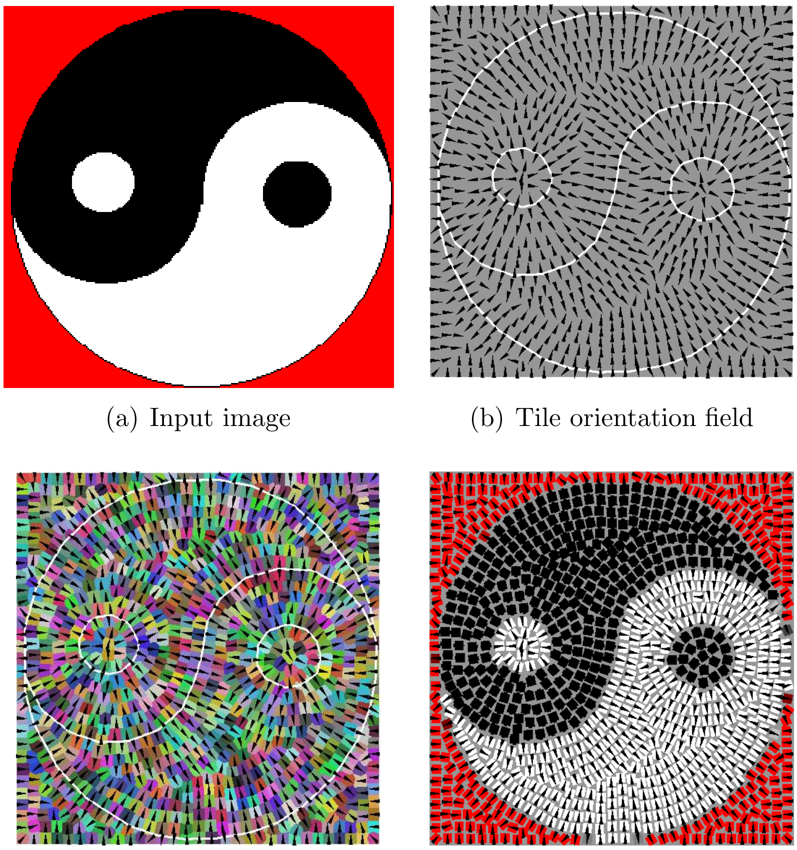
After computing the tile orientation, the next step is to pack the tiles as dense as possible and avoid putting the tiles over the strong intensity edges, which will blur the final mosaic. In Hausner's work, tile packing is done by using Centroidal Voronoi Diagram (CVD). Let N be the number of sites which are randomly distributed on the 2D space. These sites become the centers of the tiles.

A Voronoi Diagram (VD) segments the 2D space into N non-overlapping regions so that each region contains exactly one site. In addition, points within the same region are closer to the site which is contained in that region than to any other site. If the sites are the centroid (mass center) of their associated region, then this Voronoi Diagram decided by these sites is also a Centroid Voronoi diagram. In order to pack the tiles to form a classic mosaic, Centroidal Voronoi Diagram, other than regular Voronoi Diagram, is used since the regions it segments cover the 2D space more equally.

For each region generated by the CVD, a tile will be placed centered at the site it contains. The tile orientation is then decided by the orientation vector computed in the first step. Euclidean distance based CVD segments the image plane into hexagonal grids, which will result in a large gap space between square tiles. To adapt the CVD algorithm to the square tile shape, Centroidal Voronoi Diagram based on Manhattan distance is used to minimize the gap space between the neighboring tiles. When measuring the Manhattan distance from each site, the axes are rotated to be parallel to the orientation vector at the tile site, see Figure 3.1(c).

In the second stage of tile packing, an iterative algorithm is developed in Hausner's work based on Centroidal Voronoi Diagrams. At first, N tile sites, which can be also viewed as N tile centers, are distributed randomly inside the 2D image space. The number of tiles, N, is also provided by the user. In each iteration, to compute the Centroidal Voronoi Diagram, a regular Voronoi Diagram is first computed and then the sites are moved to the centroid of their associated region. To incorporate the tile orientation field computed in the first step, when computing the Manhattan distance from the sites, the axes are rotated to be parallel to the tile orientation vector at the sites. The problem with rotating the axes is that the iterative algorithm for computing CVD may not converge in some cases.

The advantage of using Centroidal Voronoi Diagram is that it is fairly easy to encode edge avoidance constraint for a good mosaic. When rendering the tiles, the tile colour



(a) Input image

(b) Tile orientation field

(c) CVD aligned to the orientation field

(d) The mosaic image

Figure 3.1: *Hausner's method. Figure (a) shows the input image. Figure (b) illustrates the tile orientation field. Figure (c) shows the Centroidal Voronoi Diagram computed with Manhattan distance and aligned with the tile orientation field in Figure (b). Notice the CVD cells are pushed away from the curves marked by the user, which are shown in white lines. Figure (d) is the final result after putting the tiles centered at each cell of CVD.*

is usually taken as the average colour of all the pixels of the original picture covered by the tile. This helps to preserve the color scheme between the original image and the mosaic rendering. If a tile is located at a point where the intensity variance is too sharp, taking the average colour of the covered pixels will create a blurring effect. By encoding edge avoidance into the algorithm, Hausner's work can reduce the blurring effect caused by putting the tiles over the intensity edges. This is done by adding a pushing force on the curves of the input curves, so it will push the tile sites to move away from the edges. After the image space is segmented into N regions by CVD, one tile is placed centered at each region, and the tile orientation should be parallel to the tile orientation vector at the site associated with the region, as shown in Figure 3.1(d).

Hausner's work produces good results in rendering classic mosaics, see Figures 3.4(d), 3.5(d) and 3.6(d). However, it requires extensive user interaction to draw object boundaries. Also, the number of tiles, N, is provided by the user, therefore, the final mosaic may have large gap space between the tiles if N is too small, or there may be a great number of tiles overlapping each other if N is too large.

An alternative approach for generating classic mosaics was developed by Elber and Wolberg's [23]. The main idea of this method is also packing the tiles along the guiding curves drawn by the user. At the beginning, a set of closed curves are marked by the user. Similar to Hausner's work, this set of curves is usually also the outline of the objects one wants to highlight in the input image. These curves are referred to as feature curves. A set of curves which are parallel to the feature curves are computed, and used as the tile orientation guidelines. The distance between nearby curves is the same as the size of the tiles in the final mosaic. This is basically the set of level lines to the user-drawn curve. Tiles are then packed along these guiding curves, with their sides parallel to the nearest curves, under the constraint that tiles do no overlap.

The main drawback of Elber and Wolberg's work is that their method creates artifacts at the pixels which are far away from the feature curves input by the user. The input feature curves usually highlight the shape of the important objects in the image. Therefore, the details in the background are often ignored. After laying the tiles along with the guiding curves which are parallel to the input feature curves, the tiles located in the background are also aligned to the object feature curve. This creates a strange halo effect in the background, with tile orientations emphasizing distant foreground objects instead of background details, see Figure 3.2. Elber and Wolberg's work also creates discontinuities of the tile orientation at the skeleton of

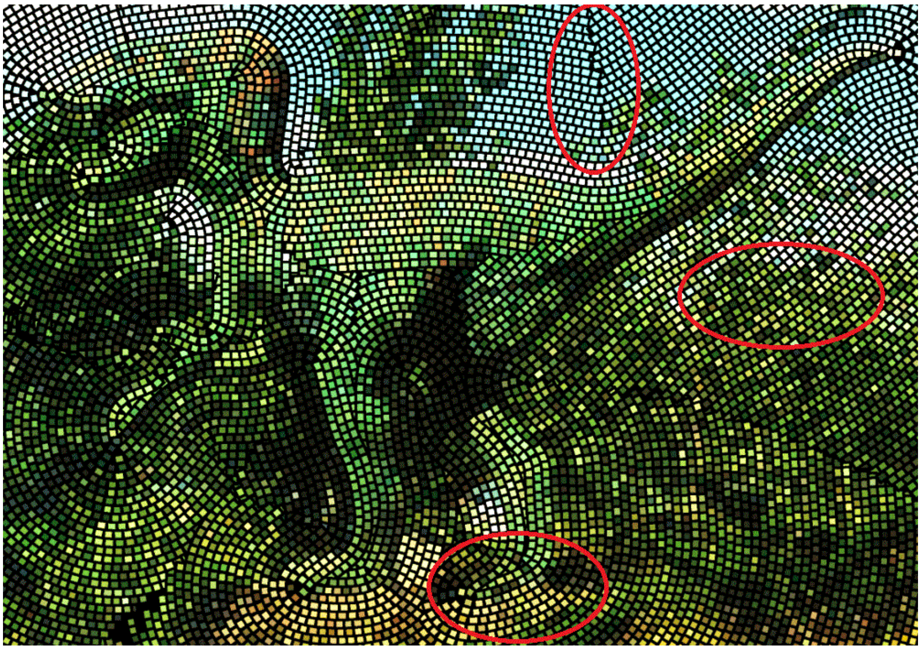


Figure 3.2: *Elber and Wolberg's [23] result: notice inside the red ovals, the tiles in the background are following the shape of the dinosaur, which creates either unnecessary discontinuities or strange halo effect that ignores the background details.*

the input curves, which results in breaking tiles when packing.

3.2.2 Methods Based on Energy Optimization

A method which is most close to our work is Battiato et al. [5]. In this paper, the authors present a classic mosaic rendering method which is also based on energy optimization framework, like our work. They also compute the tile orientation field in the beginning. In the first step, Battiato et al. [5] adopt the algorithm from Xu et al. [101] to compute the Gradient Vector Flow (GVF) of the input image. The Gradient Vector Flow is a vector field of $\mathbf{v} = [v, u]$ which minimize the following energy function:

$$E = \int \int \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla f|^2 \times |\mathbf{v} - \nabla f|^2 dxdy \qquad (3.1)$$

Here $|\nabla f|$ is the gradient magnitude of the input image and μ is the regularization parameter. By looking at Equation (3.2.2), it is easy to see that at the homogeneous regions of the input image, where $|\nabla f|$ is very small, the first term of this equation dominates the integrand. Therefore, to minimize this energy, the resulting vector field should vary smoothly so that the sum of the partial derivatives of the vector field is small. When $|\nabla f|$ is large, then the second term of Equation (3.2.2) is the dominant part. Then setting $\mathbf{v} = \nabla f$ will minimize the energy. Thus the resulting gradient vector flow will have smooth variance where the image is homogeneous, and equal to the gradient vector where $|\nabla f|$ is large. Aligning tiles with the GVF will ensure that the tiles follow the shape of the objects in the input image.

Heuristics are used to pack the tiles in Battiato et al. [5]. To pack the tiles, Battiato et al. start with points whose gradient vector magnitude is greater than a threshold T. First, tiles are placed at points with large gradient vector magnitude in a descending order. Tile overlap is prohibited in this step. For regions with small gradient vector magnitude, tiles are simply placed one by one from left to right. Tile overlap is also prohibited in this stage. The intuition behind this process is that, by starting with points of large gradient vector magnitude, areas close to the important edges are processed first.

The work of Battiato et al. [5] produces nice mosaic, see Figure 3.4(c), 3.5(c) and 3.5(c). However, their energy optimization algorithm is local and not as powerful as the global methods. Furthermore, the heuristic tile packing procedure creates uneven gap space over the mosaic image, which is an unpleasant artifact.

3.3 Overview of Our Method

Our goal is to develop a classic mosaic algorithm based on global optimization. Having a global objective function has a number of advantages. We can model the desired mosaic properties, such as tile orientations parallel to the important edges in the image, directly by including the appropriate terms into the energy function. Successful optimization of this energy guarantees that the resulting mosaic satisfies those properties that we think are desirable. If the results are unsatisfactory, we can rethink and redesign those terms in the energy functions that are likely to be the cause of a failure. By modifying the energy function, new mosaic effects can be introduced. We can also use the value of the energy as a metric for measuring and comparing the quality of the mosaics.

The energy function that we design incorporates the properties illustrated in Figure 1.5, such as tile alignment to significant edges, etc. We also avoid both user interaction and explicit edge detection that are currently required by most mosaic algorithms. Furthermore, unlike many existing approaches, we completely prohibit overlap between tiles.

Our objective function is too hard to optimize in all the variables simultaneously, and therefore we optimize different sets of variables sequentially. First we optimize for tile orientation variables. The constraints are similar to those in previous work: we require that tile orientations vary smoothly and align with the strong intensity edges. This step is performed with the α-expansion algorithm [12] which is based on graph cuts. Graph cuts proved to be a powerful optimization tool [87]. The biggest benefit of using global optimization is that our approach preserves the smoothness of tile orientations as a global property. We do not have discontinuities in tile orientations like those in [36, 23] because we optimize tile orientations directly and globally. None of the existing approaches enforce the smoothness of tile orientations in a global optimization framework. Furthermore, we eliminate user interaction because explicit edge information is not needed.

In the second step, we optimize over tile visibility variables of our energy function. Intuitively, this step can be seen as stitching together multiple candidate mosaic layers. First we generate multiple mosaic layers obeying the pre-computed tile orientations. A candidate mosaic layer is heuristically generated and therefore is not a good mosaic overall. The gap space between tiles is not optimized and many tiles may be placed over sharp intensity edges, which creates blur, like in Figure 1.5(b). However, some parts of a candidate layer are good, i.e. the tiles are packed tightly and avoid overlapping the sharp intensity edges. By optimizing over visibility variables of the energy function, we select the good parts from all the candidate layers. This step can be seen intuitively as stitching together candidate mosaic layers. Optimizing over visibility variables is also done in the energy optimization framework with graph cuts [12].

3.4 Energy Formulation for Static Mosaics

In this section, we give a detailed motivation and description of the energy function that we use for mosaic generation. We start by explaining the label set. Any tile can be identified by its center and orientation. Therefore we use two labels per pixel. Let I be the colour image from which we wish to generate the mosaic, and let $\mathcal{P}$ be the collection of all pixels inside I. For each pixel $p \in \mathcal{P}$ we wish to assign a label which is an ordered pair: (v_p, φ_p). Here $v_p \in \{0, 1\}$ is the binary "visibility" variable. If $v_p = 1$, then we place a tile centered at pixel p in the mosaic. If $v_p = 0$ then the mosaic does not have any tiles centered at p. We assume that all tiles are square with the side of size $tSize$.

The second part of the label, φ_p, specifies the orientation of the tile centered at pixel p, if there is such a tile in the mosaic. If $v_p = 1$ then φ_p has a meaning (i.e. tile orientation), if $v_p = 0$, the value of φ_p is not used. Our discrete optimization framework requires that the set of orientations is finite. Here we discretize the orientations into

n angles, at equal intervals. Since the square tiles have several angles of symmetry, we need angles only in the range of $[0, \frac{\pi}{2})$. The set of all possible orientations is:

$$\Phi = \{\frac{\pi}{2n} \times (i-1) | \;\; i = 1, 2, \ldots, n\}.$$

We set n to 32 for all the experiments.

Occasionally we need to refer to the set of all pixels covered by a tile centered at pixel p and with orientation φ_p. We will denote this set as $\mathcal{T}(p, \varphi_p)$. The color of the tile is an average of colors over the pixels in I that this tile covers.

Let $\varphi = \{\varphi_p | p \in \mathcal{P}\}$ and $v = \{v_p | p \in \mathcal{P}\}$. A mosaic then is an ordered pair of variables (v, φ) s.t. $v \in \{0, 1\}^n$ and $\varphi \in \Phi^n$, where n is the size of $\mathcal{P}$.

We are now ready to formulate the energy function for a mosaic (v, φ). Our energy function encodes the following principles for generating a visually pleasing mosaic: tiles should align with strong edges in the image I, nearby tiles should have similar orientations, tiles should avoid crossing strong edges in image I, and, finally, the gap space in the mosaic should be minimal. Our energy function is as follows:

$$\begin{array}{rcl} E(v, \varphi) & = & \sum_{p \in \mathcal{P}} (1 - v_p) + \sum_{p \in \mathcal{P}} v_p \cdot D_p(\varphi_p) + \\ & + & \sum_{\{p,q\} \in \mathcal{N}} V_{pq}(v_p, v_q, \varphi_p, \varphi_q). \end{array} \quad (3.2)$$

The first sum in Equation (3.2) ensures that the gap space is minimized. The more tiles are placed in the mosaic, the less gap space there is. In addition, this terms ensures that the optimal solution is not the trivial one: $v_p = 0$ for all pixels p. The second sum in Equation (3.2) is the data term, which measures how well the tiles that we place in the mosaic align to the edges and avoid crossing the edges. The last sum is the smoothness term that encourages nearby tiles to have similar orientations and also prohibits tile overlap. We now discuss the data and the smoothness terms in greater detail.

3.4.1 Data Term

For each pixel p, the data term is $v_p \cdot D_p(\varphi_p)$. The term $D_p(\varphi_p)$ measures the quality of a tile with center at pixel p and with orientation φ_p. Multiplying by v_p ensures that we consider only the quality of the tiles that are actually present in the mosaic. The D_p has the following form:

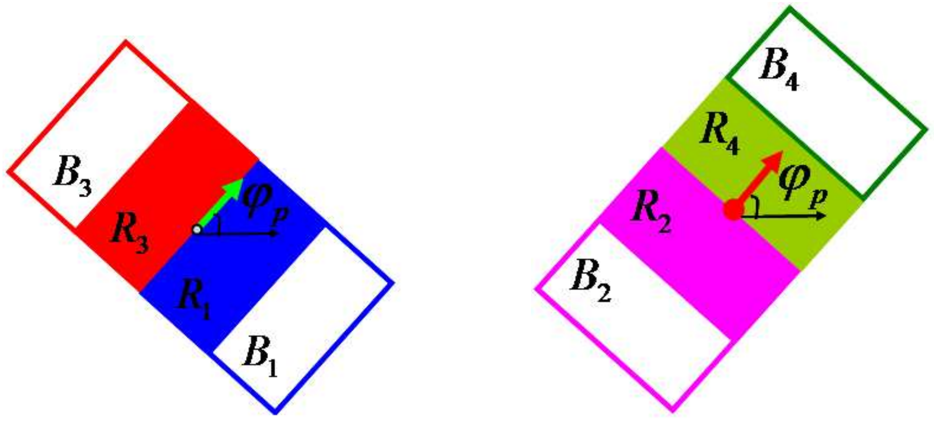
$$D_p(\varphi_p) = D_p^{align}(\varphi_p) + D_p^{avoid}(\varphi_p), \tag{3.3}$$

where $D_p^{align}(\varphi_p)$ encodes edge alignment, encouraging tile sides to be parallel to the intensity edges of the underlying image I, and $D_p^{avoid}(\varphi_p)$ encodes edge avoidance, pushing the tiles away from intensity edges to prevent blurring.

We first explain the edge alignment term D_p^{align}. Assuming that a pixel p is a tile center, and knowing the tile size, it is fairly easy to estimate how well a particular orientation aligns a tile to any intensity edge in the neighborhood. Since the tiles have four sides, we check for the evidence of a strong edge for each one of them, and then choose the side with the strongest evidence. To check for an edge presence, we use the color difference between boxes around a tile side. We split a tile in half horizontally into regions R_1 and R_3, and then, separately, we split a tile vertically into regions R_2 and R_4, as illustrated with colored solid rectangles in Figure 3.3. Regions B_1, B_2, B_3, B_4, illustrated in Figure 3.3, are located outside the tile, and adjacent to R_1, R_2, R_3, R_4, respectively. To measure the evidence for an edge on the right hand side of the tile, we take the difference in color between regions R_1 and B_1. The other sides are handled similarly. Thus D_p^{align} is:

$$D_p^{align}(\varphi_p) = w_e \cdot \max_{i=1\ldots4} \| \sum_{p \in R_i} I(p) - \sum_{p \in B_i} I(p) \|, \tag{3.4}$$

where $I(p)$ stands for the color vector at pixel p. The weight w_e is negative. Thus when there is a high response on the color difference between the pixels inside the



(a) (b)

Figure 3.3: Shows R and B regions used in of $D_p^{align}(\varphi_p)$.

tile and that outside the tile, the term $D_p^{align}(\varphi_p)$ is negative, making tile orientations with a higher contrast to incur less cost.

The edge avoidance term $D_p^{avoid}(\varphi_p)$ is defined as:

$$D_p^{avoid}(\varphi_p) = w_v \cdot \sum_{q \in \mathcal{T}(p,\varphi_p)} \|g(q)\|, \tag{3.5}$$

where $\|g(p)\|$ is the magnitude of the gradient at pixel p, and $\mathcal{T}(p, \varphi_p)$ is the set of pixels covered by the tile with center at p and orientation φ_p. We approximate gradient by the standard Sobel operator. This term measures the intensity variance inside the tile, therefore we call it the variance term. If the label φ_p makes the tile overlap a strong intensity edge, then the variance term will penalize the overlap between the edge and the tile. This term is particularly important for pixels close to the edges of an object. The weight of w_v is set to be positive, since we want high gradient to be penalized.

Notice that the $D_p(\varphi_p)$ term involves summation over a potentially large group of pixels if tile size is large. To compute $D_p(\varphi_p)$ efficiently, we use the summed-area table technique [17]. Summed-area tables allows computing $D_p(\varphi_p)$ in constant time, independent of the tile size $tSize$.

3.4.2 Smoothness Term

The last term in Equation (3.2) is the smoothness term. First we define the neighborhood system as: $\mathcal{N} = \big\{\{p,q\}|dist(p,q) \le \sqrt{2}\cdot tSize\big\}$, where $dist(p,q)$ is the Euclidian distance between the coordinates of pixels p and q. This neighborhood system is large enough to contain all pairs of pixels s.t. if tiles centered at these pixels are placed in the mosaic, then these tiles are adjacent or overlapping.

We define the interaction term $V_{pq}(\varphi_p, \varphi_q, v_p, v_q)$ as:

$$V_{pq}(\varphi_p, \varphi_q, v_p, v_q) =$$

$$= \begin{cases} 0 & \text{if } v_p = 0 \text{ or } v_q = 0 \\ w_s \cdot |\varphi_p - \varphi_q|_{(\frac{\pi}{2})} & \text{if } v_p = v_q = 1 \text{ and} \\ & \mathcal{T}(p,\varphi_p) \cap \mathcal{T}(q,\varphi_q) = \emptyset \\ \infty & \text{if } v_p = v_q = 1 \text{ and} \\ & \mathcal{T}(p,\varphi_p) \cap \mathcal{T}(q,\varphi_q) \neq \emptyset \end{cases}, \tag{3.6}$$

where

$$|\varphi_p - \varphi_q|_{(\frac{\pi}{2})} = \begin{cases} |\varphi_p - \varphi_q| & \text{if } |\varphi_p - \varphi_q| \le \frac{\pi}{4} \\ \frac{\pi}{2} + \ \varphi_p - \varphi_q| & \text{otherwise} \end{cases}. \tag{3.7}$$

The smoothness term serves two purposes. First, any finite energy labeling has no overlapping tiles. Second, it encourages orientations of adjacent tiles to have similar orientations. Notice that we only consider the orientations of neighboring tiles that are actually placed in the mosaic. The modulo arithmetic in Equation (3.7) reflects the fact that rotation by angle φ_p gives the same result as rotation by angle $\varphi_p + \frac{\pi}{2}$, due to the symmetry of a square. Thus the penalty for two neighboring pixels to have different orientation labels is an absolute difference of their labels, modulo $\frac{\pi}{2}$ arithmetic.

A mosaic (v', φ') that has a low value of energy in Equation (3.2) is expected to be visually pleasing. Any other desired mosaic properties can also be included. However, successful optimization depends on the particular form of the energy function. There may be properties that one wishes to include that make the energy very hard to optimize. For example, we may wish to include terms that make the gap space evenly distributed throughout the mosaic. However, such terms would require higher order interactions, which are much harder to optimize. We found that the energy in Equation (3.2) offers a nice balance between containing the most important terms for a pleasing mosaic, and being reasonable to optimize.

3.5 Energy Optimization for Static Mosaics

In this section we describe our optimization approach. The energy in Equation (3.2) is too difficult to optimize in all variables simultaneously. We devise a stepwise approach for approximation. First, we ignore the tile visibility labels v_p, and optimize tile orientation variables φ_p, section 3.5.1. Keeping tile orientation variables φ_p fixed, we then optimize for the visibility variables v_p, section 3.5.2.

3.5.1 Optimizing in Orientation Variables

We now explain how to optimize the orientation variables φ while ignoring the visibility variables v. Intuitively, optimizing only the orientation φ generates a smooth tile orientation field, which is usually the first step in most mosaic algorithms [36, 6]. However, unlike most previous algorithms (with a notable exception of [5]), our orientation field is generated in a principled manner using a well-understood objective function. Our advantage over [5], who also use optimization to get the orientation field, is that our objective function is optimized globally with graph cuts, not locally as in [5]. Global optimization of non-convex functions produces better results, as shown in [87]. Our energy function is non-convex, and, in fact, NP-hard to optimize, as shown below.

Ignoring the visibility labels v_p's, our energy in Equation (3.2) becomes a function of orientation labels:

$$E^o(\varphi) = \sum_{p \in \mathcal{P}} D_p(\varphi_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{pq}(\varphi_p, \varphi_q), \tag{3.8}$$

where $V_{pq}(\varphi_p, \varphi_q) = w_s \cdot |\varphi_p - \varphi_q|_{mod(\frac{\pi}{2})}$.

Another way of looking at decoupling of variables φ and v is as follows. In the energy in Equation (3.8), D_p terms are optimized for all pixels p, and V_{pq} terms are optimized for all neighboring pixel pairs $\{p, q\}$. Therefore if φ^* optimizes the energy in Equation (3.8), all $D_p(\varphi_p^*)$ and $V_{pq}(\varphi_p^*, \varphi_q^*)$ terms are expected to be small. In the complete energy in Equation (3.2), only the D_p and V_{pq} terms for pixels p, q with nonzero v_p, v_q matter. Assigning v_p's while keeping φ_p's fixed to the previously optimized values of φ_p^*'s corresponds to picking out a subset of previously optimized $D_p(\varphi_p^*)$ and $V_{pq}(\varphi_p^*, \varphi_q^*)$ terms. Since all $D_p(\varphi_p^*)$ and $V_{pq}(\varphi_p^*, \varphi_q^*)$ terms were small, their subset is also expected to be small.

To minimize the energy in Equation 3.8, we use the α-expansion algorithm [12]. According to [11], to optimize our energy with α-expansion algorithm, for all $\alpha, \beta, \gamma \in \mathcal{L}$, the smoothness term V_{pq} should satisfy:

$$V_{pq}(\beta, \gamma) \leq V_{pq}(\alpha, \gamma) + V_{pq}(\beta, \alpha) \tag{3.9}$$

We now prove that the energy in Equation (3.8) satisfies Inequality (2.7).

Proof: To simplify notation, V denotes V_{pq}. Recall that orientations are in the range of $[0, \frac{\pi}{2})$, where $\frac{\pi}{2}$ is identified with 0. By definition in Equation (3.7), for any orientation labels α, β,

$$V_{pq}(\alpha, \beta) = min\left\{|\alpha - \beta|, \frac{\pi}{2} + \ \alpha - \beta|\right\} \leq \frac{\pi}{4}.$$

Therefore

$$\begin{aligned} V(\alpha,\gamma) + V(\beta,\alpha) &= \min(|\alpha - \gamma|, \tfrac{\pi}{2} + \ \alpha - \gamma|) \\ &+ \min(|\beta - \alpha|, \tfrac{\pi}{2} + \ \beta - \alpha|) \end{aligned}$$

There are four possible cases:

Case 1:

$$V(\alpha,\gamma) + V(\beta,\alpha) = |\alpha - \gamma| + |\beta - \alpha| \geq |\gamma - \beta| \geq V(\gamma, \beta).$$

Case 2:

$$V(\alpha,\gamma) + V(\beta,\alpha) = (\frac{\pi}{2} + \ \alpha - \gamma|) + |\beta - \alpha|.$$

Because $|\beta - \alpha| + \ \alpha - \gamma| \geq \vdash \gamma - \beta|$, we have that

$$V(\alpha,\gamma) + V(\beta,\alpha) \geq \frac{\pi}{2} + \ \gamma - \beta| \geq V(\beta, \gamma).$$

Case 3:

$$V(\alpha,\gamma) + V(\beta,\alpha) = |\alpha - \gamma| + (\frac{\pi}{2} + \ \beta - \alpha|),$$

the proof is identical to Case 2.

Case 4:

$$V(\alpha,\gamma) + V(\beta,\alpha) = (\frac{\pi}{2} + \ \alpha - \gamma|) + (\frac{\pi}{2} + \ \beta - \alpha|).$$

Since $\alpha, \beta, \gamma \in [0, \frac{\pi}{2})$, and $|\alpha - \gamma| \geq \frac{\pi}{4}$ and $|\beta - \alpha| \geq \frac{\pi}{4}$, we have that either α is larger than both γ and β or α is smaller than both γ and β. In the first case, $V(\alpha,\gamma) + V(\beta,\alpha) = \pi - 2\alpha + \gamma + \beta \geq \gamma + \beta \geq |\gamma - \beta| \geq V(\gamma, \beta)$. In the second case, $V(\alpha,\gamma) + V(\beta,\alpha) = \pi + 2\alpha - \gamma - \beta \geq \pi + max\{\gamma,\beta\} - max\{\gamma,\beta\} - \gamma - \beta \geq$

$|\gamma - \beta| \geq V(\gamma, \beta)$. □

It is interesting to note that the energy in Equation (3.8) is NP-hard to optimize. Suppose $\Phi = \{0, \frac{\pi}{6}, \frac{\pi}{3}\}$. Let $\alpha, \beta \in \Phi$. Then there are three cases for $V_{pq}(\varphi_p, \varphi_q)$: $V(0, \frac{\pi}{6}) = V(\frac{\pi}{6}, \frac{\pi}{3}) = V(0, \frac{\pi}{3}) = \frac{\pi}{6}$. Thus this V_{pq} is the so called Potts model, which was shown to be NP-hard to optimize in [12].

3.5.2 Optimizing in Visibility Variables

Let φ^* be the tile orientation field computed in section 3.5.1. We now must optimize Equation (3.2) over the visibility variable v with φ fixed to φ^*. Unfortunately, optimizing v, even if φ is kept fixed, is an NP-hard bin packing problem. Our approach approximates this problem in a two-step manner. First for $i = 1, ..., m$ we generate in a heuristic manner a number of labelings v^i s.t. $E(v^i, \varphi^*) < \infty$ for all i. Therefore each (v^i, φ^*) corresponds to a mosaic with no overlapping tiles. Since v^i is generated heuristically, (v^i, φ^*) may not be a good mosaic overall, but might contain a few regions that are good candidates for the final mosaic. We call each (v^i, φ^*) a *candidate mosaic layer*. Orientations of the mosaic corresponding to each candidate mosaic layer are given by φ^*. The final step is to stitch all (v^i, φ^*) into a final mosaic (v^*, φ^*) in such a way that the energy in Equation (3.2) is minimized.

3.5.2.1 Generating Candidate Mosaic Layers:

Intuitively, building each layer candidate mosaic layer (v^i, φ^*) given a tile orientation field corresponds to using heuristics for tile placement, as done in most other mosaic generation algorithms [36, 5], etc. However, instead of generating just one final mosaic, we generate m layers, that are stitched together to form a better final mosaic according to our global energy function in Equation (3.2).

We build each v^i heuristically. Our goal is for a candidate mosaic layer (v^i, φ^*) to have no tile overlap and at the same time contain as many tiles as possible (i.e. for as many pixels as possible, variables v^i_p should be set to 1). We start with $v^i_p = 0$ for all pixels p. Next we select a starting pixel s at random. The starting pixel gets assigned $v^i_s = 1$. Then we put all the other pixels in $\mathcal{P}$ on an ordered list O of pixels to be processed. Pixels in list O are ordered by their distance to the starting pixel s, with pixels closer to s placed closer to the beginning of O. Let q be the next pixel to be processed. We set $v^i_q = 1$ if placing tile centered at q with orientation φ^*_q does not cause tile overlap among the tiles previously placed, i.e. if $\mathcal{T}(p, \varphi^*_p) \cap \mathcal{T}(q, \varphi^*_q) = \emptyset$ for all pixels p with v^i_p already set to 1. The process of "growing" v^i stops when the list O is empty. By placing pixels on the list in order of their distance to the start pixel, we are trying to ensure that the next tile placed in v^i is as close as possible to the already placed tiles, thus making the gap space in v^i minimal.

To build a candidate mosaic layer efficiently, when assigning $v^i_q = 1$ for some pixel q, we remove all the pixels in $\mathcal{T}(q, \varphi^*_q)$ from the list O. We also check for tile overlap efficiently. Whether two tiles overlap or not depends only on their relative orientations and the relative distance between the tile centers. We compute a small two-dimensional lookup table, which, based on the relative orientation and center differences, tells us whether two tiles overlap or not. Thus the overlap checking can be done in constant time.

We build enough candidate mosaic layers to ensure that for all pixels p, there is an i s.t. $v^i_p = 1$. This gives each pixel p a chance to have a tile centered at p appear in the final stitching. The candidate mosaic layers built in this heuristic manner are far from an optimal mosaic. While we try to pack the tiles tightly in each candidate layer, any layer will have regions where the packing is not as tight as possible. In addition, since the quality of tiles (i.e the data term D_p) is not checked when building the layers, many tiles will be placed on the high intensity edges which causes blurring of the mosaic. Therefore, we need the third step of stitching all layers (v^i, φ^*) together to find a better solution.

3.5.2.2 Stitching Candidate Mosaic Layers

After generating a set of candidate mosaic layers (v^i, φ^*), $i = 1, ..., m$, the last step is to stitch them together to form the final mosaic (v^*, φ^*) s.t. the energy $E(v^*, \varphi^*)$ is minimized.

The stitching is performed in a pairwise manner. Let (v^i, φ), (v^j, φ) be two mosaics with equal orientations field. Then their stitching is another mosaic (v', φ) s.t. for all $p \in \mathcal{P}$, $v'_p \in \{v^i_p, v^j_p, 0\}$. This implies that the stitching of two mosaics cannot have any tiles that were not present in either the first or the second mosaic, thus the name "stitching".

Our stitching algorithm is iterative. We always have the current mosaic (v^c, φ^*), and stitch it with one of the candidate mosaic layers (v^i, φ), chosen at random. The stitching is performed in such a way as to minimize the energy of the resulting mosaic. We update the current mosaic to the result of the stitching, and repeat. To initialize, v^c is set to a randomly chosen v^i. The process stops when there is no layer s.t. stitching this layer improves the current mosaic (v^c, φ), or when the maximum number of iterations is reached.

We now explain how to find the optimal stitching of the current mosaic (v^c, φ) and the candidate mosaic layer (v^i, φ). Let $\mathcal{P}_c = \{p \in \mathcal{P} | v^c = 1\}$, $\mathcal{P}_i = \{p \in \mathcal{P} | v^i = 1\}$, and let $\mathcal{S} = \mathcal{P}_i \cup \mathcal{P}_c$. Notice that only pixels $p \in \mathcal{S}$ can have their visibility variable v_p change as a result of a stitching. Therefore optimization is performed only over the variables v_p s.t. $p \in \mathcal{S}$. With the variables φ fixed to φ^* and optimization performed only over pixels in $\mathcal{S}$, the energy in Equation (3.2) reduces to the energy below:

$$\begin{aligned} E^v(v) = \ & \textstyle\sum_{p \in \mathcal{S}} (1 - v_p) + \sum_{p \in \mathcal{S}} v_p \cdot D_p(\varphi^*_p) + \\ & \textstyle\sum_{\substack{\{p,q\} \in \mathcal{N} \\ \{p,q\} \subset \mathcal{S}}} V_{pq}(v_p, v_q, \varphi^*_p, \varphi^*_q). \end{aligned} \tag{3.10}$$

The energy that we can actually optimize exactly is:

$$\begin{aligned}\tilde{E}^v(v) = \; & \textstyle\sum_{p\in\mathcal{S}}(1-v_p) + \sum_{p\in\mathcal{S}} v_p \cdot D_p(\varphi_p^*) + \\ & \textstyle\sum_{\substack{\{p,q\}\in\mathcal{N} \\ p\in\mathcal{P}_c, q\in\mathcal{P}_i}} V_{pq}(v_p, v_q, \varphi_p^*, \varphi_q^*).\end{aligned} \tag{3.11}$$

The difference between the energies E^v in Equation (3.10) and $\tilde{E}^v$ in Equation (3.11) is that only the pairwise terms between pixels $p \in \mathcal{P}_c$ and $q \in \mathcal{P}_i$ are present in Equation (3.11). Pairwise terms between pixels inside $\mathcal{P}_i$ and inside $\mathcal{P}_c$ are missing in $\tilde{E}^v$. We omit this terms to make optimization tractable. However, the absence of these terms is not as important as may seem at first. First of all, since there is no tile overlap in either v^c or v^i, $V_{pq}(v_p, v_q, \varphi_p^*, \varphi_q^*)$ is finite when $p, q \in \mathcal{P}_c$ and when $p, q \in \mathcal{P}_i$. Furthermore, tile orientations φ_p were optimized in the first step of our algorithm. Therefore we may assume that $V_{pq}(v_p, v_q, \varphi_p^*, \varphi_q^*)$ have low values for most neighboring pixel pairs. Therefore, it is relatively safe to exclude the pairwise terms V_{pq} when either $p, q \in \mathcal{P}_c$ or $p, q \in \mathcal{P}_i$. However, ignoring V_{pq} when $p \in \mathcal{P}_c$ and $q \in \mathcal{P}_i$ is not safe, since for such p, q the term $V_{pq}(v_p, v_q, \varphi_p^*, \varphi_q^*)$ could be infinite due to overlap of tiles centered at p and q. We do include such “unsafe” terms in the energy $\tilde{E}^v$. Therefore, the energy $\tilde{E}^v$ is a good approximation to the energy E^v.

Another explanation for $\tilde{E}^v$ is that it finds a stitching of the current mosaic (v^c, φ^*) and a candidate layer (v^i, φ^*) in such a way that the gap space is optimized and the “good” tiles (tiles with small D_p) are selected . Orientations at the seam of the stitching are accounted for, but orientations outside of the stitching seam are disregarded, since those are already assumed to be satisfactory because an optimized φ^* is used.

We now explain how to optimize the energy in Equation (3.11). We use the idea of roof duality from [34]. Let us introduce a new variable t_p for each pixel p, with the following dependence on the visibility variables v_p. If $p \in \mathcal{P}_c$, then $t_p = v_p$. If $p \in \mathcal{P}_i$, then $t_p = 1 - v_p$. In words, for the pixels in $\mathcal{P}_c$, the meaning of variable t_p is the same as the meaning of variable v_p, and the meaning of variable t_p is reversed for $p \in \mathcal{P}_i$. Let $t = \{t_p | p \in \mathcal{P}\}$. We can rewrite the energy in Equation (3.11) in terms of the new variables:

$$E^v(t) = \sum_{p \in \mathcal{S}} D'_p(t_p) + \sum_{\{p,q\} \in \mathcal{N}_{ci}} V'_{pq}(t_p, t_q), \tag{3.12}$$

where $\mathcal{N}_{ci} = \{\{p,q\} | \{p,q\} \in \mathcal{N} \text{ and } p \in \mathcal{P}_c, q \in \mathcal{P}_i\}$, and $D'_p(t_p)$, $V'_{pq}(t_p, t_q)$ are defined below:

$$D'_p(t_p) = \begin{cases} 1 + t_p(D_p(\varphi^*_p) - 1) & \text{if } p \in \mathcal{P}_c \\ t_p + (1 - t_p) D_p(\varphi^*_p) & \text{if } p \in \mathcal{P}_i \end{cases} \tag{3.13}$$

and

$$V'_{pq}(t_p, t_q) = \begin{cases} 0 & \text{if } t_p = 0, t_q = 0 \\ 0 & \text{if } t_p = 0, t_q = 1 \\ V_{pq}(\varphi^*_p, \varphi^*_q, 1, 1) & \text{if } t_p = 1, t_q = 0 \\ 0 & \text{if } t_p = 1, t_q = 1 \end{cases}. \tag{3.14}$$

As shown in [53], a binary energy can be optimized exactly with a graph cut if it is submodular, that is if the pairwise terms satisfy: $V_{sr}(0,1) + V_{sr}(1,0) \geq V_{sr}(0,0) + V_{sr}(1,1)$. Clearly the energy in Equation (3.12) is submodular, since $V'_{pq}(1,0)$ is either a positive constant or infinite, and all other V'_{pq} are 0. Therefore the energy in Equation (3.11) can be optimized exactly with a graph cut. For implementation, we use the max-flow/min-cut algorithm in [13].

The last detail of the stitching algorithm is as follows. Let $(\tilde{v}, \varphi^*)$ be the mosaic that is the result of stitching the current mosaic (v^c, φ^*) with a candidate mosaic layer (v^i, φ^*), i.e. $\tilde{v}$ optimizes the energy in Equation 3.11. Since the energy in Equation 3.11 is only an approximation to the energy in Equation 3.10, we check if $E^v(\tilde{v}) < E^v(v^c)$, that is if the stitching using approximate energy is better according to the exact energy. If yes, then we update v^c to $\tilde{v}$. If not, we discard the results of stitching.

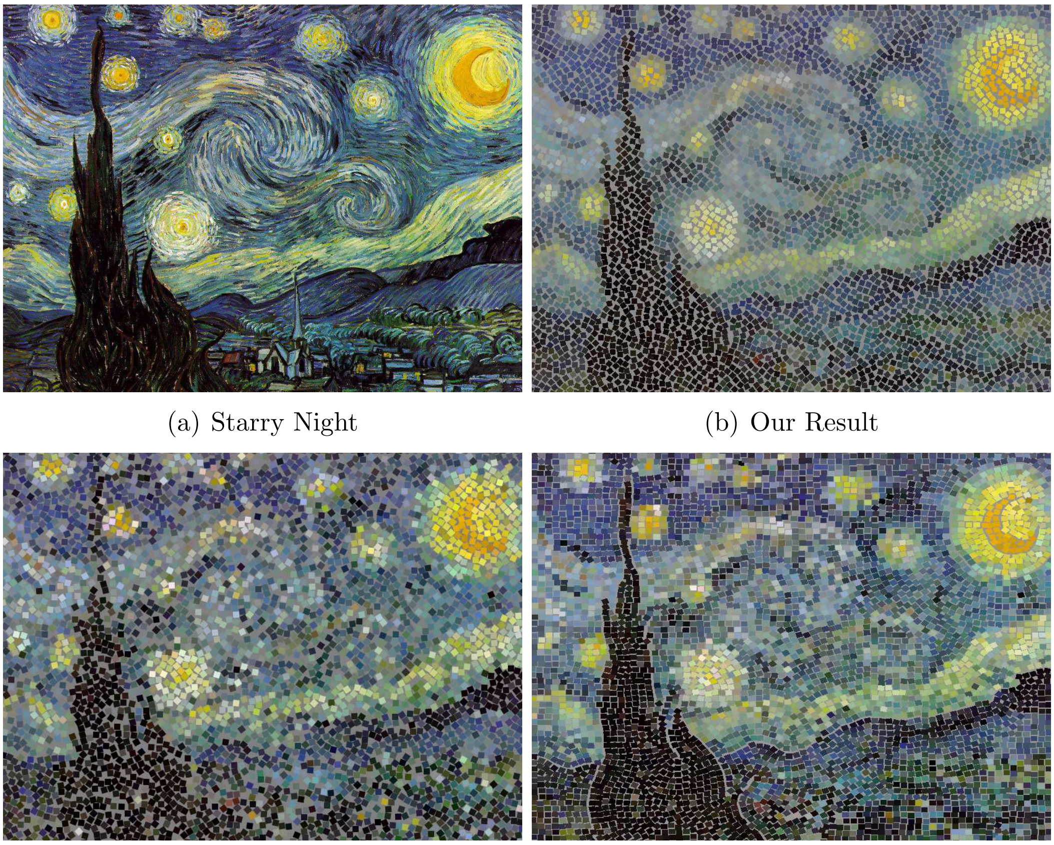
Note that [58] use an optimization procedure similar to ours for computing the optical flow from video. They compute many flow fields and “fuse” or stitch them together.

3.6 Experimental Results

We now present our experimental results. We perform comparison to Hausner [36] (using code available on the author’s web site), and we also compare to Battiato et.al. [5](using the results provided by the authors). For all the experiments, the parameters were fixed to the following values: $w_e = -50$, $w_v = 20$, and $w_s = 20$.

Figure 3.4(a), 3.5(a) and 3.6(a) show the images used for mosaic generation. Figures 3.4, 3.5, and 3.6 show the mosaics obtained with our method, Battiato et.al., and Hausner. Compared to Battiato et.al., our mosaic is more spatially coherent. Figure 3.4(c) and 3.6(c) have many tiles that “pop out”, that is their color is incoherent compared to the nearby tiles. This happens because Battiato et.al. contains heuristic steps that do not discourage tiles to cross strong intensity edges. When a tile crosses a strong intensity edge, its color is blended and it stands out from the surrounding tiles, as illustrated in Figure 1.5(b). We perform global optimization and discourage strong edge crossing as a part of the energy function. In addition, compared to that of Battiato et.al., the tiles in our mosaics are better aligned to object edges, and therefore they outline the object shapes more accurately. For example, in Figure 3.4(b), all the stars are clearly delineated, where as in Figure 3.4(c) only the larger stars have reasonable outlines and the other stars are blended with the background. The tiger face that we produce (Figure 3.6(b)) is more readily recognized as a tiger compared to that of Battiato et.al. (Figure 3.6(c)). This is, again, due to our mosaics delineating objects more clearly and have less tiles that are incoherent with the surrounding tiles.

The method of Battiato et.al. can be viewed as a simplified version of our algorithm. They also generate tile orientation field (similar to what we do in Section 3.5.1, but using local optimization). Then they generate the final mosaic heuristically. Thus their



(a) Starry Night

(b) Our Result

(c) Battiato et.al. [5]

(d) Hausner [36]

Figure 3.4: Starry Night Results.

final mosaic is equivalent to our single mosaic layer, and the steps in Section 3.5.2.1 and 3.5.2.2 are not performed. To understand the importance of stitching multiple layers together, consider Figure 3.7. Figure 3.7(a) shows our starting layer. This would be similar to the output of Battiato et.al. The mosaic in Figure 3.7(a) is clearly inferior, many tiles overlap intensity edges and therefore their color is blurred. The Chinese character can be barely recognized in this mosaic. Figure 3.7(b) shows our result after stitching 64 layers together. The results are significantly improved compared to the starting layer in Figure 3.7(a). The result of stitching 98 layers (Figure 3.7(c)) still shows a mild improvement over Figure 3.7(b).

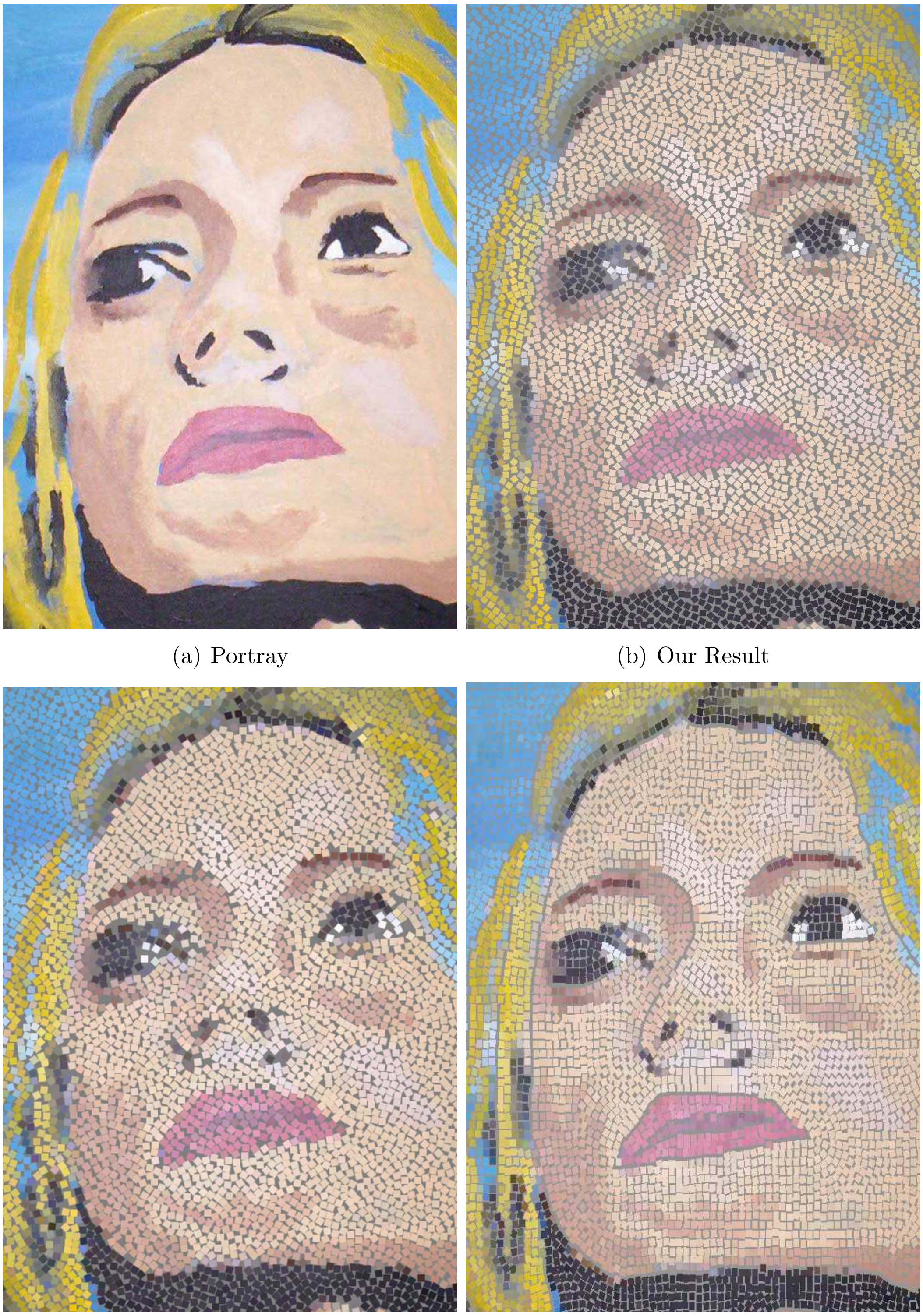
Compared to Hausner [36], our mosaics also have fewer tiles that "pop out", due to our use of global optimization. Since Hausner needs user interaction, the extent to which the objects are outlined depends solely on the user. For example, in Figure 3.4(d), the user marked the boundaries around the tree, the largest star, and the border between the sky and the ground regions. These are exactly the objects that are outlined very well in the resulting mosaic. The boundaries of other objects are not emphasized. For example, the medium size star on the right of the castle has tiles with orientations following the tree boundary, not the star boundary. In addition, tile orientations of Hausner's mosaics is visibly more discontinuous compared to our mosaics.

Having a global energy function allows us to control the mosaic appearance by tuning parameter values. For example, parameter w_v in Equation (3.5) determines the importance of the edge avoidance constraint. When w_v is set to 0, tiles are free to cross any edges and the mosaic is blurred, see Figure 3.8(b). When w_v is very large, some tiles which are close to the edges will be removed and a large gap space is created in the final mosaic, see Figure 3.8(a). A good choice of w_v is in the middle between the two extremes in Figure 3.8.

On the images in Figure 3.4(a), 3.5(a) and 3.6(a), of sizes 867 by 691, 940 by 1233, 1200 by 827, the running times were, respectively, 8, 20, and 18 minutes. We sped up our program by pre-computing a table storing the area of overlap for a pair of tiles. This table was indexed by the relative positions and orientations between tiles. Using this table reduced the time spent on building mosaic layers and stitching the layers from tens of minutes to less then ten minutes. However, since more then 40% of the running time was spent on generating the tile orientations, it remains our future work to improve our time complexity.

3.7 Summary

We formulated the problem of classic mosaic generation in a global optimization framework, and designed an energy function encoding the properties that lead to



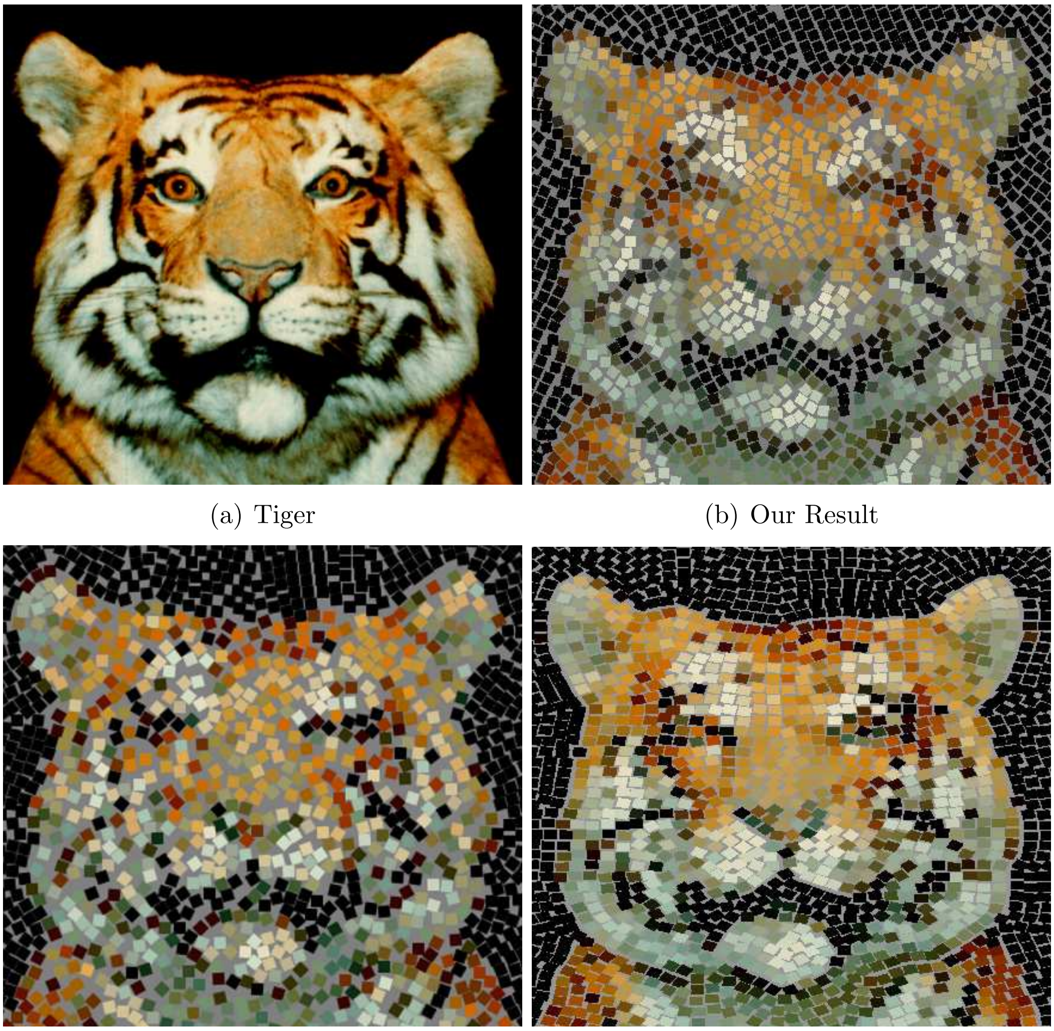
(a) Portray

(b) Our Result

(c) Battiato et.al. [5]

(d) Hausner [36]

Figure 3.5: Portray Results



(a) Tiger

(b) Our Result

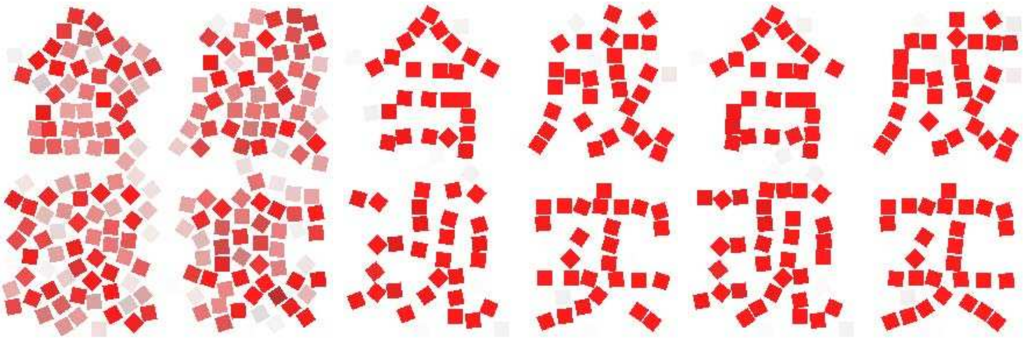
(c) Battiato et.al. [5]

(d) Hausner [36]

Figure 3.6: Tiger Results.

perceptually pleasing mosaics. Our global optimization framework offers a more principled approach than previous work, which is mostly based on heuristics. The desired mosaic properties can be directly modeled into an energy function, instead of devising a sequence of heuristics steps that may possibly lead to the desired result.

If the results are unsatisfactory, it is clear that either the energy function itself or the optimization procedure for the energy function is at fault. Therefore one should either redesign the objective function or exploit a different optimization method. Another



(a) Starting Layer (b) After 64 iterations (c) After 198 iterations

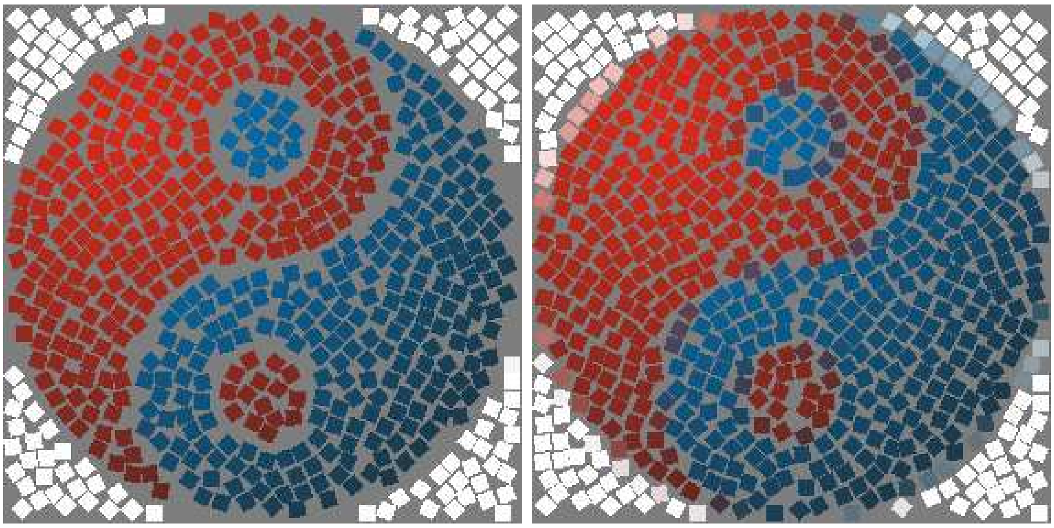
Figure 3.7: Progression of mosaic stitching described in Section 3.5.2.2 .

advantage is that the value of the energy function itself can be an effective measure to assess the quality of a mosaic.

Note that the artificial mosaics produced by our algorithm are different in appearance from the classic mosaics produced by artists. In these methods, the tiles are placed at successive bands around the feature edges. This creates discontinuities at the skeleton of the feature curves, as explained in the introduction. The tiles have smoothly varying orientations.

While removing user interaction is an advantage for a naive user, an artist may want more control on the edges to emphasize. If desired, it is easy to include user interaction in our framework, by fixing orientations of some tiles to user specified values. Then the algorithm can proceed as before, but optimizing only over the free variables.

In the future, we intend to extend our approach to rendering mosaics with tiles of different size and shapes. Smaller tiles are needed in image regions which have fine scale details, and larger tiles are sufficient in areas of the image which have coarse features. Therefore we need to vary the tile size for different regions of the image. It is relatively trivial to include tile size as an additional (third) variable in our optimization framework, favoring the areas with higher spacial frequencies to have smaller tile size. In addition, we generate time-coherent video mosaics, which is introduced in Chapter 4.



(a) Mosaic with large w_v

(b) Mosaic with small w_v

Figure 3.8: Mosaics with different w_v

Chapter 4

Rendering Animated Mosaics with Graph Cuts

In this chapter, we propose a method which renders animations with classic mosaic effects from real world video. There has been a great interest in generating classic mosaics in the recent years, see [36, 23, 10, 79, 18, 63, 64]. However, little work has been reported on generating mosaic animations automatically, especially for animated mosaics from real video. The animations generated by our approach are composed of hundreds of colourful square tiles, which are arranged to present the shape and colour of the objects in the video, moving in a timely coherent manner. Each frame of the resulting animation is a classic mosaic image composed of a large number of square tiles, which are aligned to the strong edges in the input scene. Between the consecutive frames, the tiles are moved according to the motion of their center pixels. Therefore, the whole animation has a consistent motion effect. Our method estimates the motion of the pixels in the video, renders the frames with mosaic effect based on both the colour and motion information from the input video. We aim at a tool that requires minimal help from the user to finish the task of generating animated mosaics. We hope with the help of our animation tool, more people will be able to create their own mosaic animation without professional training. Most material in this chapter was published in [62].

This chapter is organized as follows. Section 4.1 is a brief introduction to animated mosaics. In Section 4.2 we discuss the related work. In Section 4.3 we give an overview and in Section 4.4 a detailed description of animated mosaic generation from video. We present experimental results in Section 4.5, and we conclude with a discussion in Section 4.6.

4.1 Introduction

An image stylized with a NPR technique can have a stronger effect on the user than the original. This is perhaps even more true of NPR animation [68, 60, 38, 50, 37, 94, 56], since time adds a new dimension for expressiveness. A sequence of mosaic images creates a uniquely appealing animation style. While it is possible to create animated mosaics manually, with stop-motion animation, the process is extremely labor intensive. However, there has been very little work on creating mosaic animations automatically or interactively [85, 18].

In this chapter, we develop a system for creating animated mosaics directly from video sequences. In addition to all the challenges of a static mosaic, see Chapter 3, as in any other NPR animation, we have to ensure inter-frame coherency. Our approach is inspired by [85], who were the first to realize the unique set of challenges for mosaic animation. In many NPR animation methods, in order to facilitate temporal coherence, the primitives are allowed to deform, scale, blend, etc. However, to stay faithful to the classic mosaic style, the tile primitives cannot undergo any such transformations. Each individual frame must be a convincing mosaic, and at the same time the whole sequence must exhibit a convincing motion.

One way to achieve temporal coherency is to displace groups of tiles in a manner consistent with the true motion in the scene. Therefore accurate motion segmentation is critical for creating appealing moving mosaics. However the state-of-the art in motion segmentation is not accurate enough for our application. Therefore we need to involve a user to help correct any inaccuracies in motion segmentation. We develop a new algorithm for motion segmentation with occlusion reasoning that requires only a minimal help from the user. The idea is to present the user with a subsampled sequence of frames, and let the user point out, with a single click, the reliable motion segments. The information from the reliable motion segments is propagated through the neighboring frames to the rest of the sequence. To make the problem more tractable, we make a simplifying assumption that the motion is piecewise-rigid. This can create an interesting "puppet" like effect, that can be considered a part of animation style, since our goal is not precise motion transfer. Motion segmentation and user correction are addressed in a global optimization framework with graph cuts [12].

After motion segmentation is of satisfactory quality, we pack the tiles into the discovered coherent motion layers, using colour information in all the frames in a global manner. Our tile packing algorithm is based on the still mosaic algorithm of [64], with several modifications to address the unique challenges presented by video input, including handling of occlusions. Our method relies extensively on optimization with graph cuts [12], which are used for background subtraction, mosaic packing, and motion segmentation. We produce colourful, temporally coherent and uniquely appealing mosaic animations. We believe that our method is the first one to animate classic mosaics directly from video.

4.2 Related Work

The goal of our work is to animate classic mosaics directly from video input, and thus the main challenge is ensuring temporal consistency. Stylizing each frame individually produces disturbing artifacts, such as tiles "poping out" and "disappearing". Artifacts may be more tolerable in the moving parts of the scene and could possibly be regarded as a special effect. However flickering artifacts are especially pronounced in the static regions of the frame, with the apparent motion due to differences in rendering of the same underlying static scene. Therefore most NPR methods seeking to stylize a video have to deal with temporal consistency.

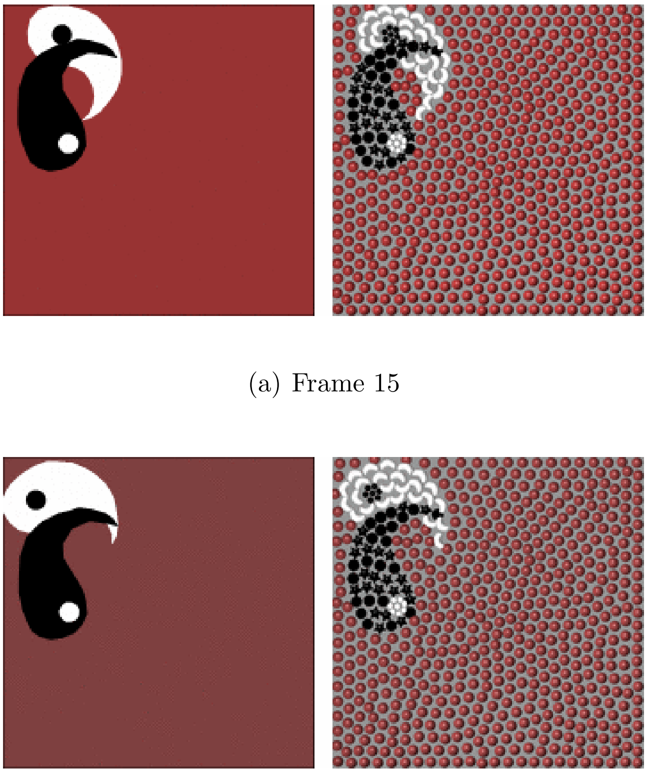
There are two main approaches to ensuring temporal consistency for NPR rendering of video sequences. The first group of methods is based on explicit computation of motion, typically based on optical flow [59, 38, 37]. The idea is to let the rendering primitive (brush strokes, etc.) follow the motion field so that the primitives appear attached to the scene objects. Without correcting for motion, instead of perceiving an object with consistent rendering move throughout the scene, the scene appears to be repeatedly painted over. Our work falls into this first group, and therefore we need to compute explicit motion information from video.

The second group of methods treats a video as a space-time 3D volume [50, 18, 94]. In the work of Klein et al. [50], a video sequence is viewed as a space-time 3D volume. Cutting this 3D volume into small "video solids" and stylizing it with different rendering primitives can generate a different animation effect. The cutting can be done in two different ways. First, the user can input several cutting lines for each key frame of the input video. The cutting lines will be interpolated between key frames to form swept surfaces over time, dividing the volume into video shards, or the so called "video solids". The second approach is to decompose the 3D space-time video volume into small video solids by KD-tree. By cutting the 3D volume in these two ways, the approach prosed by Klein et al. [50] provide the user with either an automatic (by KD-tree decomposition) or an interactive tool (by using user input cutting lines) to guide the motion of the rendering primitives, and avoid the problem of motion estimation. 3D rendering primitives, such as paint strokes, voronoi cells and video shards, are arranged to fill in the 3D video volume, following the motion trajectory created by cutting the video volume. The advantage of this approach is that motion, a notoriously hard computer vision problem, does not need to be computed. This approach, however, is not suitable for rendering classic mosaic animation, since the rendering primitives will be scaled, blended or distorted. Therefore, it can not satisfy the constraints for mosaic rendering.

There has been almost no work on classic mosaic animation. We are aware of only two methods [85, 18]. In [18], a moving mosaic is created by packing 3D volumes with temporally repeating animated shapes. This work is very interesting and produces appealing animations, however, it is far from our goal of rendering a real video in a classic mosaic style.

Our work was inspired by [85]. They make two important observations. The first is that many devices for temporal coherence in NPR animation are based on letting the primitive rendering units change (i.e. scale, blend, etc.) This is not appropriate for classic mosaic animation. They argue that for classic mosaics specifically, one should target coherent motion of group of primitives, i.e. tiles. However in their work they assume that motion information is provided by the user. The input to their algorithm is an animated scene represented as a collection of 2D "containers". The correspondences between containers in adjacent frames are known, as well as the motion correspondence on the boundary of the container. The correspondence between the vertices of these containers is also provided by the user.

There are two ways to move the tiles to achieve temporal coherency. The first approach is to move the tiles in groups according to the motion of the anchor point, which is either one of the vertices of the container or the centroid of the container. The user also has the freedom to choose any point in the container as the anchor for motion. Moving the tiles with the anchor point will make tiles appear and disappear only at the edge of the container. Therefore, such approach preserves the interior of the mosaic very well.



(a) Frame 15

(b) Frame 16

Figure 4.1: *Two consecutive frames from the animation generated by Smith et al. [85].*

The second choice is to move a group of tiles according to the closest edge. The motion of the edge can be interpolated from the motion of its two vertices. This will result in tile insertion and deletion only in the center of the container. Therefore, the edges of the mosaic image are preserved.

The results of Simith et al. [85] are visually appealing, see Figure 4.1. However, our goal is to generate mosaic animations from real video sequence. Moreover, we want to avoid the user interaction of drawing the containers and their correspondence. Thus we must estimate the "containers" and their correspondence. We extend the work of [85] to real video sequences. We develop a new motion segmentation algorithm to perform the correspondence step.

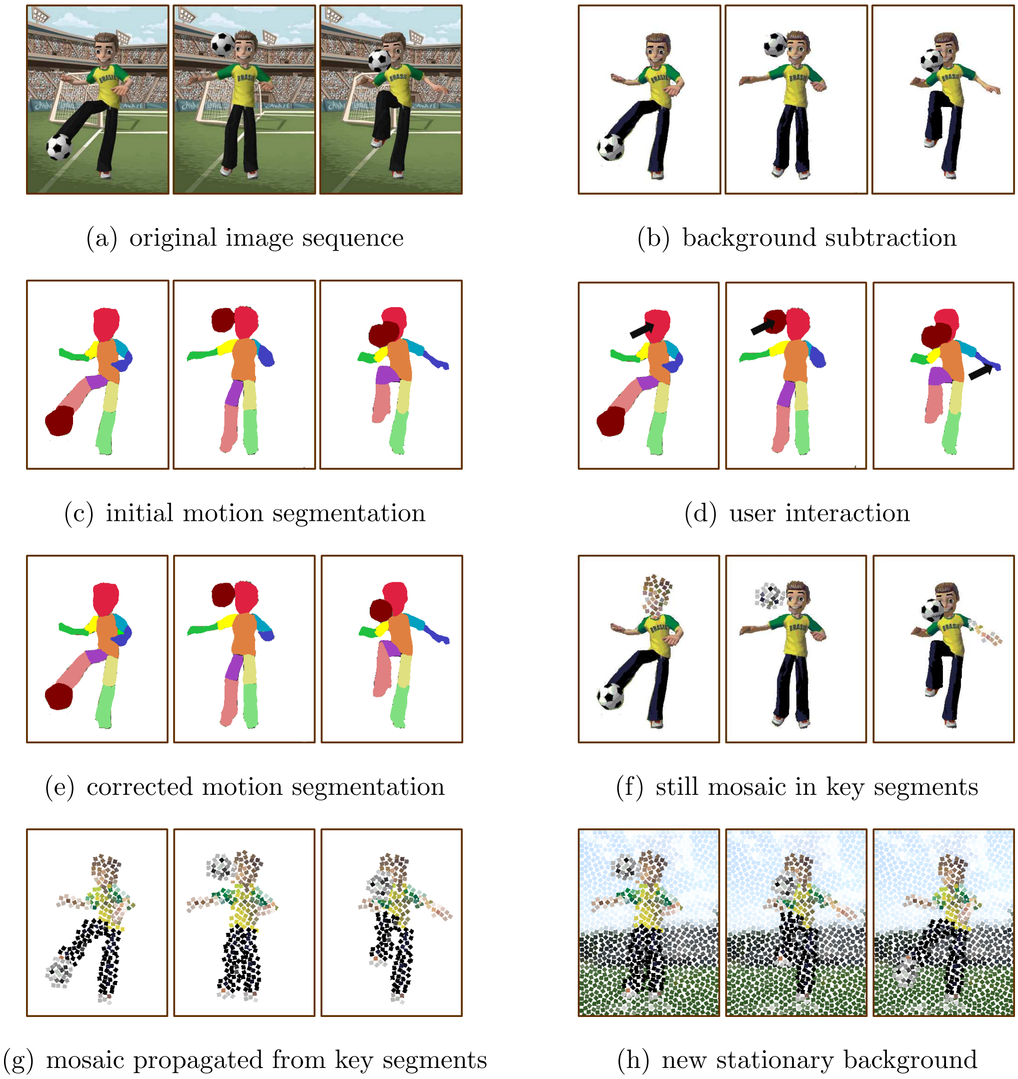
For completeness, we should mention a related but distinct work on video mosaics [51], who pack complete video segments to form a "video" mosaic (different from a classic mosaic).

4.3 Overview of the Algorithm

An overview of our approach to generating classic mosaic animations from real video sequences is illustrated in Figure 4.2. We start with a sequence of several frames, see Figure 4.2(a). We assume that the scene was imaged in front of a stationary background. The assumption of stationary background is easy to remove. However, we are interested in background replacement, since many video sequences are taken against dull backgrounds that do not produce appealing mosaics. Thus our first step is background subtraction in each frame, see Figure 4.2(b).

To ensure temporal coherence, we need to determine groups of pixels that have a common motion. This is the well studied problem of motion segmentation [95, 19, 97]. The goal of motion segmentation is to find groups of pixels in two or more frames that move together and to estimate the motion field associated with each group. In the terminology of motion segmentation, such a group of pixels is typically called a *layer*. Unfortunately, the state of the art in motion segmentation rarely produces results accurate enough for our application. Thus we need to involve the user in the loop for corrections.

We develop a motion segmentation algorithm for the whole sequence in a global optimization framework. Let L be a layer of pixels with common motion throughout the whole sequence. If general motions are allowed, the "containers" corresponding to L in two different frames may undergo drastic changes in scale, shear, etc. One has to come up with non-trivial strategies for filling these corresponding "containers" with tiles such that each container is a valid mosaic and the apparent motion between the frames is acceptable. In [85], they explore two such strategies with different visual


(a) original image sequence

(b) background subtraction

(c) initial motion segmentation

(d) user interaction

(e) corrected motion segmentation

(f) still mosaic in key segments

(g) mosaic propagated from key segments

(h) new stationary background

Figure 4.2: *Summary of the approach.*

effects. Unlike [85], we are already facing a formidable task of motion segmentation of a real video, so instead we chose to make a restriction on admissible motion models.

We assume that the motion of a layer L between neighboring frames is well approximated by rotation and translation, that is a rigid motion. Notice that between each individual pair of frames, the translation and rotation parameters of L can be different. With this restriction, the "containers" corresponding to layer L in neighboring frames have identical shape, except if there is an occlusion or out of frame motion.

Therefore, we also need to include occlusion detection as a part of our motion segmentation algorithm. Under restriction to a rigid layer motion, packing two "containers" between corresponding frames is almost equivalent to moving tiles from one container to another, following the computed motion, except that parts of a container may become occluded by another layer. Limiting the range of motions to translation and rotation is the most restrictive assumption of our algorithm.

Automatic motion segmentation rarely produces error-free results. Figure 4.2(c) illustrates initial motion segmentation. Notice that the motion segmentation of the ball in the first and third frames is not accurate. Therefore we ask the user for corrections, as illustrated in Figure 4.2(d). We sample a portion of frames and present them to the user. Our user interaction is particularly simple. In case some part of a moving object was not segmented correctly, we ask the user to find a nearby frame where the same part was correctly segmented, and click on that well segmented part. These correct segments indicated by the user are then propagated to the neighboring frames and the rest of the sequence. For example, in Figure 4.2(d), the user notices incorrect segmentation of the ball in the first and the third frames, and the correct segmentation in the second frame. The user clicks on the correct ball segment in the second frame and this correct segmentation is propagated to the next and previous frames. In figure 4.2(d), the user clicks are shown with black arrows. Figure 4.2(e) gives the results after user-assisted motion segmentation correction. During correction propagation, we also handle occlusions between the motion layers.

Finally it is time to pack the tiles. The packing algorithm is as in Chapter 3, with some adjustments to the data term to take advantage of the data available in the whole sequence. First the tiles are placed into the "key" segments indicated by user interaction, see Figure 4.2(f). This is done because these segments are more likely to correspond to regions with higher image quality, since they have been recovered correctly by the initial motion segmentation algorithm. Next the mosaics of the key segments are propagated to the rest of the sequence, taking occlusions into consideration. Then we place mosaic in any remaining segments that have not been tiled yet, and render the tiles with the corresponding image colours, see Figure 4.2(g). Lastly we place the recovered moving mosaics in front of a chosen stationary background, where the stationary background is rendered as a classic mosaic using the algorithm in [64], see Figure 4.2(h).

4.4 Detailed Description of the Algorithm

We now give a detailed description of our algorithm outlined in Section 4.3. We start with a sequence of m frames, denoted by $I_1, I_2, ..., I_m$.

4.4.1 Background Subtraction

Many video sequences are taken in front of visually uninteresting scenes, resulting in unimpressive mosaic backgrounds. Our solution is to remove the background and render the moving object in front of a more lively scene, rendered as a classic mosaic using the method in [64]. Background subtraction is a well studied area in computer vision [26]. For a high quality animation, it is important to perform accurate segmentation of the foreground. Segmentation based on global optimization tends to produce more accurate results, and therefore we formulate background subtraction as a binary segmentation problem.

We could perform background segmentation for all the frames simultaneously. However, the backgrounds in our test sequences tend to be stationary and simple in appearance. We found that there is almost no additional accuracy to be gained by incorporating information between the frames. The additional memory requirement for joint background segmentation in all sequences is massive, however, and therefore we segment each frame separately.

Our approach to background subtraction is similar to that of [86]. The task is formulated as a binary labeling problem (Section 2.2), with one label corresponding to the background and another to the foreground. The energy function is as in Equation (2.3). The data terms for the background label are modeled as a Gaussian with the mean and covariance estimated from the background samples taken in the absence of the object. A single Gaussian is enough because the background is relatively uniform in appearance. The data terms for the object are modeled by a uniform distribution. The smoothness terms are the Potts model. The coefficients w_{pq} are set inversely proportional to the magnitude of image gradient.

4.4.2 Initial Motion Segmentation

Motion segmentation is the most important step of our algorithm. Temporal coherency of the final animation depends most of all on the accuracy of segmentation. Motion segmentation is a widely studied problem in computer vision [97]. Methods based on global optimization [98, 99, 81] produce more accurate results, especially around the segmentation boundary. We develop our own motion segmentation algorithm, particularly suitable for our application. Out of the approaches mentioned above, it is most closely related to that of [98].

In [98], motion segmentation is performed on pairs of frames at a time. At first, a sparse set of feature points is matched across two frames by using the feature tracking algorithm proposed in [83]. Then using a modified version of RANSAC [29], several potential motion models are fitted to the matched points. The next step is to perform dense assignment of image pixels to motion layers, this is done with graph cut optimization [12]. The algorithm can be further iterated, refining motion models from the dense motion layers and then reassigning pixels to motion layers again.

For our application, we need to find coherent motion layers for the whole sequence, not just a pair of frames. One solution is to track feature points throughout the whole sequence, as in [99], but the drawback is that number of feature points that appear in all frames is very limited. Feature detection is brittle, even a fairly reliable feature will disappear for a few frames in a sequence.

Our solution is as follows. Initially we estimate only the pairwise (or *local*) motion models between two adjacent frames. Afterwards we find correspondences (possibly one to many) between the motion models in the adjacent frames. In this way, we recover *global* motion models, that is models that describe motion from the first frame to the second frame and eventually to the last frame. After we have estimated these global motion models we use them to perform global motion segmentation, that is segmentation for all the frames at the same time.

Let $I_1, I_2, ..., I_m$ be the m input frames. We match feature points between pairs of frames I_d and I_{d+1}, for $d = 1, ..m - 1$. Next we fit k motion models using RANSAC between each pair of adjacent frames. Since we only allow translation and rotation, each motion model has three parameters, one for rotation and two for translation. Let $\mathcal{M}_d$ be the set of motion models estimated between frames I_d and I_{d+1}, for $d = 1, ...m - 1$. The initial number of models contained in each $\mathcal{M}_d$ is k, that is we ask RANSAC to return the k best models. Let $\mathcal{M}_d^i$ stand for the ith motion model in $\mathcal{M}_d$, i.e. $\mathcal{M}_d^i$ is the ith estimated motion model between frame d and $d+1$. Figure 4.3 is an oversimplified illustration for $m = 3$ and $k = 3$.

We first perform dense motion segmentation between each adjacent pair of frames independently, using the estimated motion models $\mathcal{M}_d$, much in the manner of [98]. To be more specific, given a pair of frames I_d and I_{d+1}, the label set $\mathcal{L} = \{1, ..., k\}$, where each label j corresponds to the jth estimated motion model in $\mathcal{M}_d$. To densely assign labels to pixels in frame I_d, we perform optimization of the energy as in Equation (2.3) with the expansion algorithm of [12]. The data terms for pixel p and label $l \in \mathcal{L}$ measure how likely is pixel p to have motion $\mathcal{M}_d^l$ from frame d to $d + 1$. This data term is based on the colour difference between pixel p in I_d and the same pixel shifted according to the motion model $\mathcal{M}_d^l$ in frame I_{d+1}. We use the Potts smoothness term V_{pq} in Equation (2.3). Let $S^1, S^2, ..., S^{m-1}$ be the resulting segmentations. Here S^d corresponds to segmentation in the frame number d, and $S^d_p = l$ is the motion label assigned to pixel p in frame d. That is if $S^d_p = l$, then pixel p has motion $\mathcal{M}^l_d$ between frames d and $d+1$. Figure 4.3 illustrates a hypothetical result of pairwise motion segmentation.

This initial pairwise motion segmentation gives us information about groups of pixels with consistent motion between pairs of frames, but we need pixel groups with consistent motion across the whole sequence. To find those, we need to perform global optimization across the whole sequence. Therefore we need to find plausible global motion labels that describe motion through the whole sequence. Let $1, 2, ...c$ be the c hypothetical global motion labels. Each individual global motion label, say motion label l, describes how pixels obeying this global motion l move from the first frame to the second, from second frame to the third, and so on until the last frame. We have pairwise motion models $\mathcal{M}_d$ that describe how pixels move from frame d to $d+1$, but we do not know how these same pixels move from frame $d+1$ to $d+2$. That is, given a motion model from $\mathcal{M}_d$, we do not know the "corresponding" motion model in $\mathcal{M}_{d+1}$. One possibility is to take the set of global labels as the product set $\mathcal{M}_1 \times \mathcal{M}_2 ... \mathcal{M}_{m-1}$. This would be very inefficient, however, and most labels in this product set are not plausible.

Instead of taking the product set of $\mathcal{M}_d$'s, we use the following heuristic but simple procedure for determining, given a motion model for frame d, to which motion model for the frame $d+1$ it could correspond to. Consider the motion segmentations results $S^1, ...S^{m-1}$, performed between pairs of frames individually. Let $R^d_i = \{p \in \mathcal{P} | S^d_p = i\}$. That is R^d_i is the set of pixels that are labeled with motion i, or, equivalently, have motion $\mathcal{M}^i_d$ in frame d. Let us warp pixels in R^d_i to frame $d+1$ using motion model M^i_d, and let $W(R^d_i)$ be the set of warped pixels. If at least 80% of pixels in $W(R^d_i)$ are assigned to the same motion model, say model M^j_{d+1}, and if the size of $W(R^d_i)$ is equal to at least 80% of all pixels in S^{d+1} that are assigned motion model

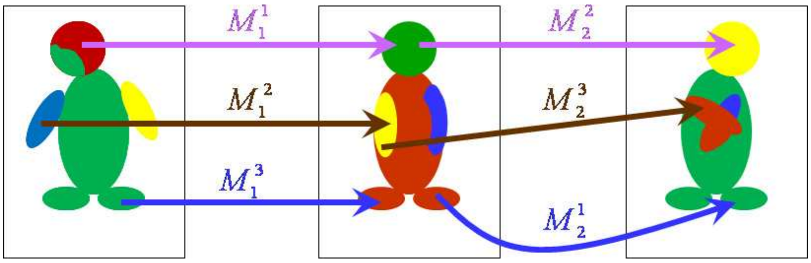


Figure 4.3: *Illustrates global label construction. Three frames that result from pairwise motion segmentation are shown. Three models are extracted between each pair of frames, i.e. $k = 3$. Different labels are illustrated by different colours. Notice that after pairwise motion segmentation, we do not know that the "red" model in frame 1 should correspond to the "green" model to in frame 2 and to the "yellow" model in frame 3. This should be discovered automatically. In practice, motion correspondences are not as easy to resolve as in this picture. Three global motion models extracted: purple (combines M_1^1 and M_2^2) brown (combines M_1^2 and M_2^3), and blue (combines M_1^3 and M_2^1).*

M_{d+1}^j, then we say that motion model M_d^i corresponds to motion model M_{d+1}^j. In Figure 4.3 the corresponding motion models are indicated by the arrows with the same colour. The illustration in this figure is oversimplified, for clarity. Occasionally we need to combine two or three motion models in frame d to satisfy this condition, i.e. we need to take several models in frame d so that pixels assigned to either of these models make 80% of pixels assigned to the same motion in frame $d+1$. This happens because motion segmentation occurs at different level of precision in each pair of frames. For example, between frames d and $d+1$, an arm could be fitted with two motions, but between the next pair of frames, $d+1$ and $d+2$ the whole arm is fitted with one motion. In such cases, we add new motion models to sets M^d and M^{d+1}, the model allowing for no arm splitting in M^d (the added model is simply a combination computed from the two motion models allowing the split), and the motion model with arm split to M^{d+1} (the new model is based on warping the two "split" models from the previous pair of frames).

The procedure described in the previous paragraph creates many global motion labels by linking labels between pairs of frames into a single chain, see Figure 4.3. Notice that chains can start after the first frame and end before the last frame, allowing for appearance of new layers and disappearance of old layers, which usually happens due to occlusion or out of frame motion.

With global motion labels, we are ready to proceed to global layer segmentation. However, if performed on the pixel level, the whole sequence does not fit into the memory on 32-bit architecture. Therefore, we undersegment each frame into "superpixels"[1] using the segmentation algorithm of [27]. Optimization is performed by assigning labels to superpixels, not pixels, resulting in huge memory savings. The neighborhood system is now three-dimensional, with superpixels between the frames also connected. The neighborhood system in the spatial dimension consists of superpixels that have common boundary. In time dimension it has superpixels that have significant overlap in spatial coordinates. Specifically, we connect a superpixel p in frame d to superpixel p' in frame $d+1$, for $d = 1, ..., m-1$ if the (x, y) coordinates of superpixels p and p' overlap by at least 70%. This is justified because we expect the motions to be relatively slow, so most pixels between adjacent frames do not change their global motion labels. Data terms are still based on colour similarity. For a superpixel, the data term is computed as the average of data terms for all the pixels that are contained in it.

4.4.3 User Interaction

The initial results of motion segmentation are not likely to be accurate for all frames. The segmentation algorithm is sensitive to the choice of parameters in optimization, to failures in feature detection and spurious motion model detection by RANSAC. About half of the errors are caused by occlusions between motion layers. Therefore we ask the user to provide guidance.

We sample about one fifth of the frames and show their motion segmentation to the

[1] A superpixel is simply a small image patch returned by a segmentation algorithm.

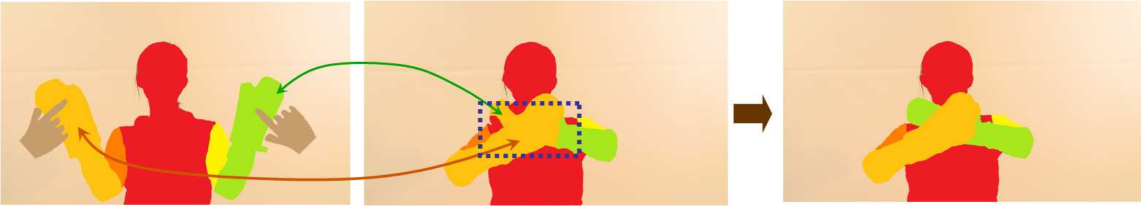


Figure 4.4: *User interaction and motion segmentation correction.*

user. To correct segmentation, starting with the first frame that is not accurately segmented, the user has to point out its correct segmentation in a nearby frame. Consider Figure 4.4. The middle pictures shows segmentation results with gross errors due to occlusion, highlighted with a rectangle. The hands are correctly segmented in the frame on the left. The user selects the correctly segmented parts by clicking on them. Notice that only a single click per layer is required, since segmentation is assumed to be already accurate. This information is used for correcting motion segmentation.

4.4.4 Correction of Motion Segmentation

Let $F^1, F^2, ..., F^{m-1}$ be the motion segmentation with global labels. Suppose the user clicks on a group of pixels assigned a global label l in frame i. Let G_l^i be this group of pixels, i.e. G_l^i is spatially contiguous, contains the pixel the user clicked on and $G_l^i = \{p \in \mathcal{P} | F_p^i = l\}$. We fix the labels of pixels in G_l^i to strongly prefer label l in the ith frame. That is we set the data penalties to be infinite for all labels other than l for pixels in G_l^i in the ith frame. Furthermore, we warp pixels in G_l^i to the $(i+1)$th frame according to the motion label l. Let $W(G_l^i)$ be the set of warped pixels in frame $i+1$. We set neighborhood links w_{pq} (see Section 2.2) between pixels in G_l^i and $W(G_l^i)$ to a large number. Here p is a pixel in frame i and q is the pixel in frame $i+1$ that p gets warped to by the global motion model l.

Here we implicitly assume that motion model l is actually a good fit for pixels in G_l^i, since the user has no way to measure the fitness of the model metrically, he just pays attention to motion boundaries. However, in our experience, regions with accurate boundary segmentations do get assigned a correct motion model because the correct set of pixels was used to compute the motion model. Most errors in motion estimation arise when a wrong set of pixels is fitted with a single model.

Now we are ready to describe occlusion handling. The coefficient w_{pq} is also set in proportion to the colour similarity between pixels p and q. The more similar are the colours, the higher is w_{pq}. Weighting w_{pq} in direct proportion to colour similarity helps us to handle occlusions automatically. Consider Figure 4.4 again. Let O be the group of pixels in the area where the left hand occludes the right hand. Both the left hand pixels and the right hand pixels in the first frame get connected by strong links to pixels in O. However, the links from the left hand are stronger, since the left hand pixels are actually visible in the second frame and their colour similarity, on average, is stronger than that between the right hand and pixels in O. Therefore pixels in O get assigned the correct label, as shown in the leftmost image in Figure 4.4.

After the data terms and the neighborhood weights w_{pq} are updated, the motion segmentation is recomputed again, propagating user corrections throughout the whole sequence and resolving occlusions.

Notice that our occlusion reasoning is based on colour similarity (accumulated over large groups of pixels, not decided for each pixel individually). This requires two layers that come in occlusion to have sufficiently distinct colour or texture, given the resolution of the video camera. If two occluding layers are not distinguished enough through texture, our occlusion reasoning may fail. The mosaic appearance may still be reasonable, however, due to the similarity of colours in the confused layers.

4.4.5 Mosaic Rendering

At this point, we have computed motion segmentation with user help and we are ready to pack mosaic tiles. We start with the “key” segments pointed out by the user, since these segments are likely to correspond to image data of high quality.

For still mosaic, given a pixel p and orientation label φ, we need to decide on the penalty of placing tile with center at p and orientation φ. This penalty is modeled from the data around pixel p, see Chapter 3. For a video sequence, the penalty should depend not just on the current frame, but on all the other frames in the sequence. Let K be a "key" segment in frame I^d that the user clicked on. If we place a tile centered at p under orientation φ, this tile will be propagated by the global motion model (the motion model that pixel p got assigned in frame I^d) throughout the whole sequence. Therefore, to model the data penalty, we propagate the tile throughout the whole sequence (notice its orientation will change in different frames) and compute the data penalty in each frame of the sequence, using the same procedure in each frame as for the still mosaic. The final data term for pixel p to have a tile centered at it with orientation φ in frame I^d is the average of all the data terms from all the sequence frames.

After packing the "key" segments and propagating them throughout the video sequence, we pack the "empty" regions. We start with the first frame, pack any unprocessed regions and propagate them throughout the whole sequence using the same algorithm as for the "key" segments. If there are any unprocessed regions in the second frame (for example, because a new global motion label appears, due to a new object part coming into view), we repeat the same procedure as for the first frame. This process is repeated from the first frame to the last until the whole sequence is packed with tiles. The final step is to paint the tiles with the colours of the underlying image and to insert the chosen mosaic background.

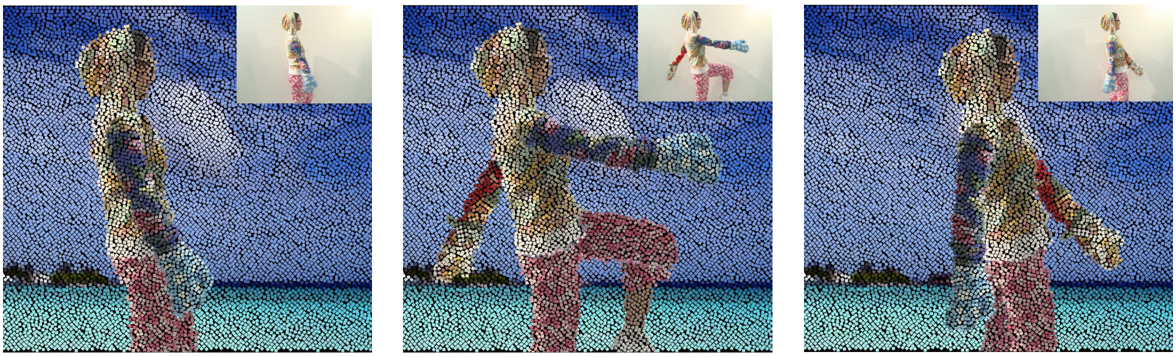


Figure 4.5: *Several frames from a* Walking *sequence and the corresponding classic mosaic.*

4.5 Experimental Results

Our results are best viewed through video animations. [2]. Figure 4.5 shows three frames of a “Waking” sequence. This sequence contains significant occlusions between the torso, upper limbs and lower limbs. In addition, parts of the leg appear and disappear from the scene. Our system produces a nice time coherent animation, with correctly handled occlusions. Due to our restricted motion assumption, the animated figure has a distinctive “puppet”-like effect. This can be regarded as part of our animation style. To create an animation that is closer to the true motion in the scene, we need to implement more general motion models.

Figs. 4.6 and 4.7 show results on other video sequences. Observe how each individual frame of animation is a pleasing classic mosaic. The “Waving arms” sequence is relatively simple, with no significant occlusions. The motion of the torso is modeled with two layers, creating an interesting visual effect. The “Occluding arms” sequence has significant overlap between the two arms, which is handled gracefully. The torso and the head parts have motion very close to stationary. We decided to fix the head

[2] see *http://www.csd.uwo.ca/faculty/olga/VideoMosaic/results.html*

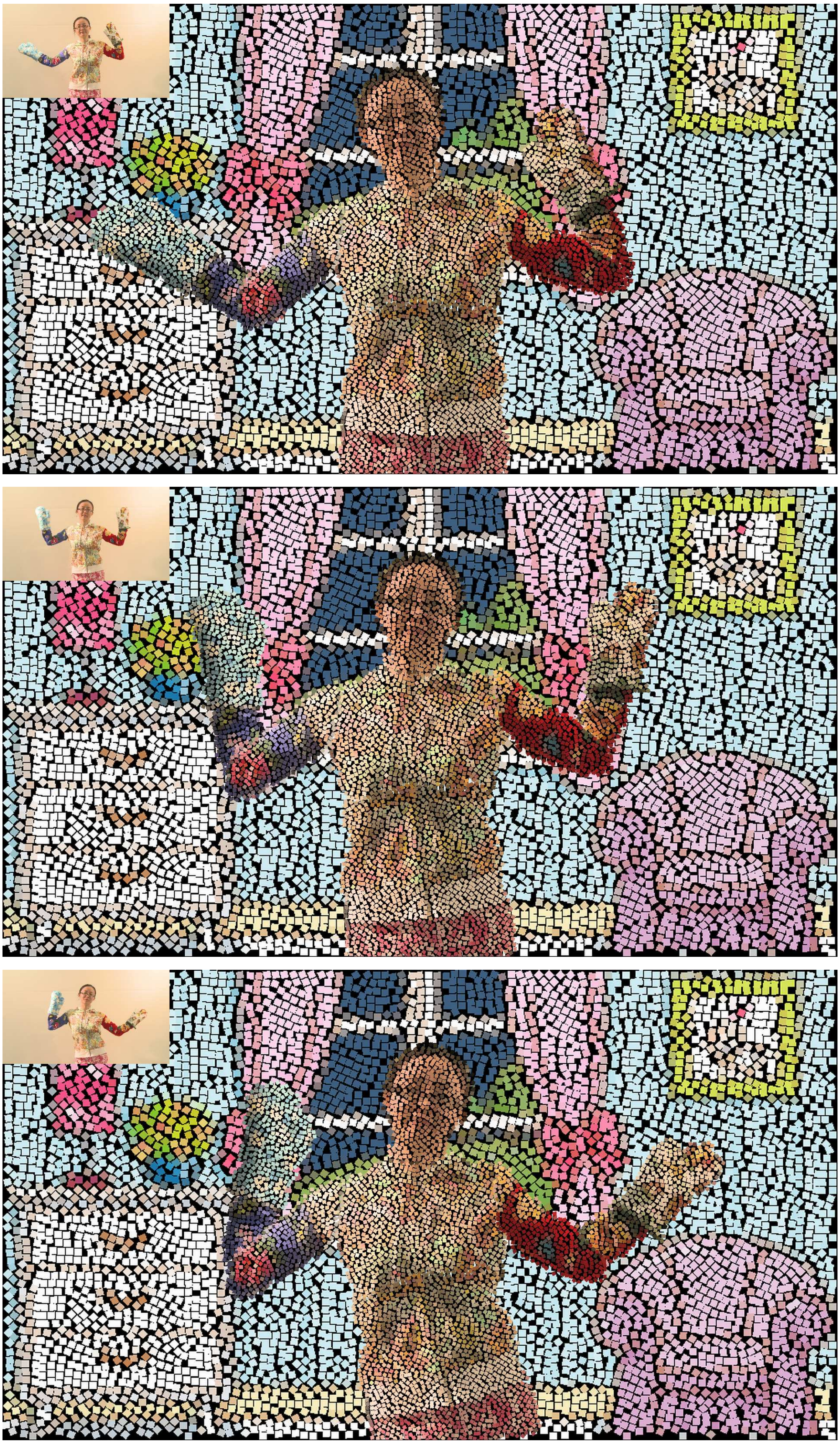


Figure 4.6: Results on “Waving arms” sequence.

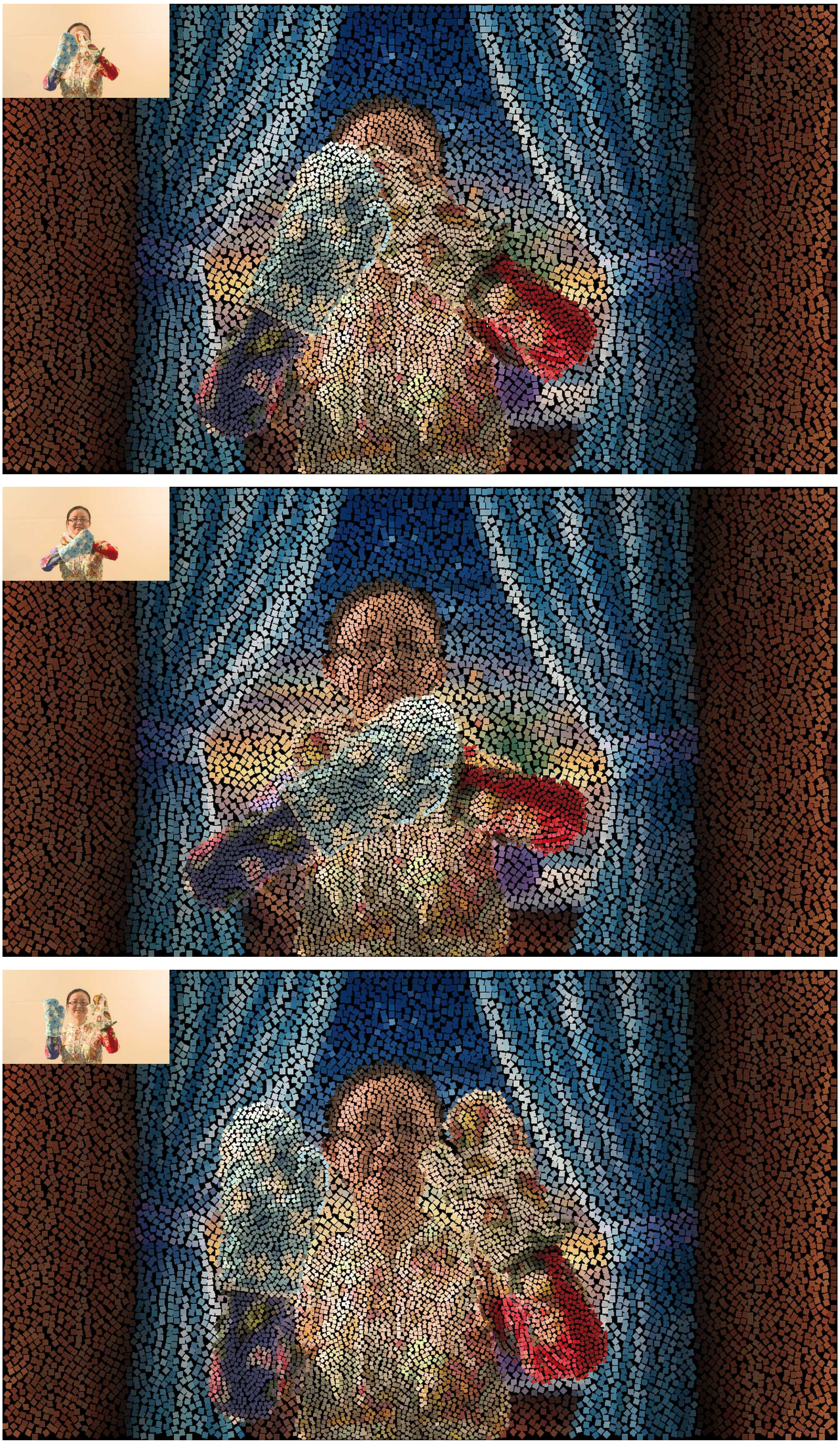


Figure 4.7: Results on an “Overlapping arms” sequence.

and the body to be stationary which visually blends them into the background, to create a somewhat sinister “arms sticking out of the wall” effect.

The processing time of the current implementation needs significant improvement. The most time consuming step is motion segmentation. To speed this step up, faster implementations of the max-flow algorithm is needed. For the sequences presented in this section (frame size is 1920 by 1080), it took a few hours on a personal computer to produce the final mosaic animation, including user interaction.

4.6 Summary

We presented an approach for rendering video sequences in a classic mosaic style. This style is uniquely expressive, part of its appeal comes from the fact that mosaics is an ancient art form. The algorithm relies extensively on our novel motion segmentation algorithm. The state of the art in motion segmentation is such that user interaction is still required to get convincing results. We produce temporally coherent visually appealing animations.

Our biggest current limitation is that the motion layers can only be assigned rigid motion. This can create an interesting “puppet” like effect, that can be considered a part of animation style, since our goal is not precise motion transfer. If a more faithful motion is desired, more general motion models need to be implemented.

Chapter 5

Finding Semi-dense Visual Correspondence

To improve the quality of our animated mosaics (see Chapter 4), we need to detect the motion between the frames of a given video sequence more accurately. The main limitation of our animated mosaic work is that our motion model is restricted to rigid motion (rotation and translation). To simulate the deformation of the objects in the input video, we need to extend our motion models to be more general. Motion estimation is usually performed by establishing visual correspondence between consecutive frames. In this chapter, we propose an approach to detecting the semi-dense visual correspondence between an input stereo pair. Although our original goal was to find a more robust way to detect motion in a video, stereo problems have a similar setting with motion but are easier to start with. Therefore, in this thesis, we focus on visual correspondence in stereo problems. Our main objective is to find semi-dense visual correspondences for which we can estimate the range of disparities with a high confidence. The visual cues that are reliable for establishing correspondence are usually located at pixels with high texture. If there is a region in the image that is surrounded by texture cues, its range of disparities can be estimated more reliably. Even if the region is textureless inside, we can propagate the visual cues from the texture cues at the boundaries.

This chapter is organized as follows: Section 5.1 gives a brief introduction to the problem of finding semi-dense visual correspondence for stereo. In section 5.2, we summarize some related works in stereo problem. In section 5.3, we give an overview of our method. Section 5.4 is the detailed description of our algorithm. In section 5.5, we present our results on finding semi-dense visual correspondence, with quantitative analysis on their accuracy. In section 5.6, we summarize our work.

5.1 Introduction

The input to the visual correspondence problem is usually a pair of images of the same real world scene. A pixel in one image is said to correspond to a pixel in the other image, if these two pixels are projections along the lines of sight of the same physical scene element, see Figure 1.6 in Chapter 1. The problem is to find pairs of such corresponding pixels. For stereo, these two images are taken by two synchronized cameras, which are rectified so that the baseline of the cameras is parallel to their image plane. Corresponding pixels are found between these two images, see Figure 1.6. The difference in the locations of corresponding pixels in the left and right images, often referred to as "disparity", is used to determine the 3D depth of these pixels. The disparities for the stereo problem are only in the horizontal dimension, which means that the corresponding pixels are on the same scanline.

Determining the disparities in the stereo problem is very challenging. The basic step is measuring the matching error of two corresponding pixels. There are different matching errors used in stereo [41]. In the simplest case, the matching error is just the absolute difference of the intensity difference of two pixels. Ideally, the matching error between corresponding pixel pairs should be very small if the disparity is correct. However, the accuracy of this measurement can be affected by many factors. For instance, image noise brings in a lot of errors when measuring the matching error. Other problems can be caused by sampling artifacts, exposure changes, etc.

Among all of these difficulties, two of the hardest problems we need to overcome are lack of texture (or repeated texture in some case) and occlusions. Recall the stereo example given in Figure 1.7. The disparities of the occluded pixels are hard to determine since their corresponding pixels in one of the images are hidden behind some objects. This makes it impossible to measure the matching error. For textureless regions, there are ambiguities when measuring the matching cost. Compared with the matching error of the true disparity, the matching error between one pixel and the neighbours of its corresponding pixel may also be very small. Therefore, it is hard to determine which pixel is the real corresponding pixel.

There exists roughly two categories of approaches to resolving the ambiguities due to the lack of texture. First, it is well known that reliable visual correspondence can be obtained in the regions which are highly textured. Therefore, sparse feature points can be detected in the textured regions. And their visual correspondence can be determined confidently. However, these sparse features are not dense enough to be useful for most applications such as our animated mosaic problem. Dense correspondence approaches estimate disparity at every pixel but can have gross errors in some parts of textureless areas. Worse still, most dense approaches do not produce confidence maps for their estimates, that is they do not say which parts of the disparity maps they are more confident in.

Our work was inspired by Veksler [91, 90]. In these papers, the author maintains that reliable visual correspondence can be established in the textured regions by comparing the matching error with the edge strength at the point of interest. At a particular pixel p, if its matching error of some disparity d is lower than the edge strength at the same point by some threshold t, then it can be considered as a reliable feature that supports the disparity d for pixel p. Vise versa, for disparity d, if the matching error is higher than the edge strength at a pixel, then this pixel can be viewed as a negative feature which strongly disagrees with disparity d. In [91, 90], the threshold t for the difference between matching error and edge strength is picked up by hand. In this work, we seek for a mechanism which extracts useful features for finding visual correspondence automatically. Thus we avoid the problem of determining the hard threshold. And we can also discover more features for stereo beyond the difference between matching error and the edge strength. The result of this step is a classifier that can detect sparse visual cues for stereo in the textured regions.

The second step of our semi-dense visual correspondence approach is to propagate the sparse cues detected at the textured regions to the textureless regions. Instead of propagating the visual cues separately for each disparity, we propose a grouping method which clusters the sparse cues detected in the previous step into several groups, based on their geometric locations and associated disparities. This results in several feature groups containing sparse cues from consecutive disparities. And the sparse visual cues in the same group are also geometrically close to each other. We use the graph cut algorithm to propagate the information brought by the groups of the visual cues into textureless regions. This step is done for each feature group individually. And the results are blob-like semi-dense features which support groups of consecutive disparities.

At last, we need to resolve the ambiguities. One pixel inside the input image pair can belong to more than one semi-dense feature blobs generated in the previous step. To find the exact boundaries of these semi-dense cues, we apply α-expansion algorithm to resolve the ambiguities between the semi-dense feature blobs. This step produces nice boundaries between regions of the images with different groups of disparities. The disparity in the textureless regions are often recovered by our approach with a high confidence.

5.2 Related Work

There has been a long lasting interest in stereo and motion. A complete and thorough survey in the literature of finding visual correspondence is impossible because the large number of new publications coming out each year. Furthermore, most researchers only report their own qualitative results, which makes it harder to do a fair evaluation for these algorithms. However, there are good surveys on stereo and motion algorithms. The study conducted by Barron et al. [4] provided a thorough survey about commonly cited optical flow approaches. Scharstein et al. [76] published an in-depth taxonomy and evaluation of two-frame stereo correspondence algorithms. Brown et al. [14] gave a deep investigation into computational stereo methods from the viewpoint of matching cost computation and optimization. Hirschmuller [40, 41] and Scharstein [41] compared the performance of different matching cost functions. Here, we give a brief summary on the visual correspondence algorithms from the point of matching constraints.

Visual correspondence can be determined under many different constraints. For instance, in optical flow based methods [42], the brightness of the same scene point in the images taken from different view angles should be consistent. Other assumptions are made on the smoothness of the disparity field. Local methods [3] [104] usually emphasize the constraints on a small number of neighboring pixels surrounding the pixel of interest. Global methods [12] [91] take into account a whole scan-line or the entire image by optimizing certain energy functions. Compared with global methods, local methods are in many cases more computationally efficient. Speed is essential for real-time applications such as [47]. The shortcoming of this approach is that it is more sensitive to image noise and ambiguous regions. Global methods are more robust in dealing with local ambiguity, since global constraints allow propagation of information from textured regions to regions with low texture. However, these methods often suffer from the expensive computational cost, and, furthermore, do not give a confidence rating in the produced disparity estimates. With the progress of energy optimization algorithms, for instance, graph cuts [13] [12], the performance of global methods have been greatly improved. In this section, we list several important methods emphasizing either global or local constraints. At last, we will give a more detailed description of [91] which is the most close work to our semi-dense visual correspondence approach.

5.2.1 Local Methods

Local methods emphasize constraints that only rely on a small number of neighboring pixels close to the point of interest. For a certain pixel, it is usually assumed that its surrounding pixels in a small neighbourhood give supportive cues for visual correspondence of the point of interest. Because of the dependence on only a small number of pixels, local methods are often computationally efficient. There are many different ways of applying the local constrains. Here we list three widely used representations of local constraints: window matching, gradient-based method and feature matching.

Window Matching

In window matching methods, to estimate the disparity at a point in one image, a small region around that point, namely, a template, is extracted from the reference image. Small regions from the other image with different disparities are then extracted and compared with the template. The region with least matching error, sometimes referred as matching cost, is then selected and its associate disparity is assigned to the point of interest.

Intuitively, the matching error between a template and its corresponding regions can be represented by the intensity differences between them. The two most commonly used matching cost functions are: the sum of squared intensity difference (SSD)[3] [35] [67] [84] and the sum of absolute intensity difference (SAD) [47]. Noise and disparity discontinuities inside the stereo image pairs cause high matching cost if SSD and

SAD are used. Normalized cross-correlation [35] and binary matching cost [66] are relatively insensitive to sudden changes of image intensity. Therefore, they are more robust for dealing with noises but still fail if there are intensity discontinuities within a matching window.

Since cameras are sensitive in radiometric gain or bias, this commonly results in gross errors in the computation of the matching cost. Approaches using non-parametric measures, such as rank and census transformation [104], work well for eliminating this kind of error. The rank transform for a small region around a pixel is defined as the number of pixels in that region for which the intensity is less than that of the center pixel. A variation of the rank transform, the census transform, is also proposed by Zabih and Woodfill [104] to increase the discriminatory power, since information is lost during the rank transform. These measures are insensitive to differences in camera sensitivity, therefore they are more robust in bad conditions. Brichfield and Tomasi [8] propose shifted absolute difference, which reduces the inaccuracy cased by sampling errors. All these methods can still fail if there are disparity discontinuities inside the matching window. A comprehensive comparison among different matching cost functions can be found in Hirschmuller [40].

Gradient Methods

A common assumption, valid under Lambertian surface reflectance model, is that the same world point projected to different image planes, both in stereo and motion, should have constant brightness. This is the foundation for gradient-based methods. Most optical flow methods [42] formulate the brightness constraint into a set of differential equations. For instance, in stereo, the disparity of a point from the reference image to the other can be determined by solving the following equation:

$$(\nabla_x E)v + E_t = 0. \tag{5.1}$$

Here $\nabla_x E$ is the horizontal component of the image gradient, E_t is the intensity differences between left and right stereo images, and v is the horizontal disparity between left and right images. Within the local optimization framework [65], it can be assumed that the disparity is smooth over a small region. Therefore, more constraint can be added into Equation (5.1). Let $p_1, p_2, \ldots, p_n$ be the n pixels surrounding our point of interest. The disparity of this point can be estimated by the following equation system:

$$v = (A^T A)^{-1} A^T b, \tag{5.2}$$

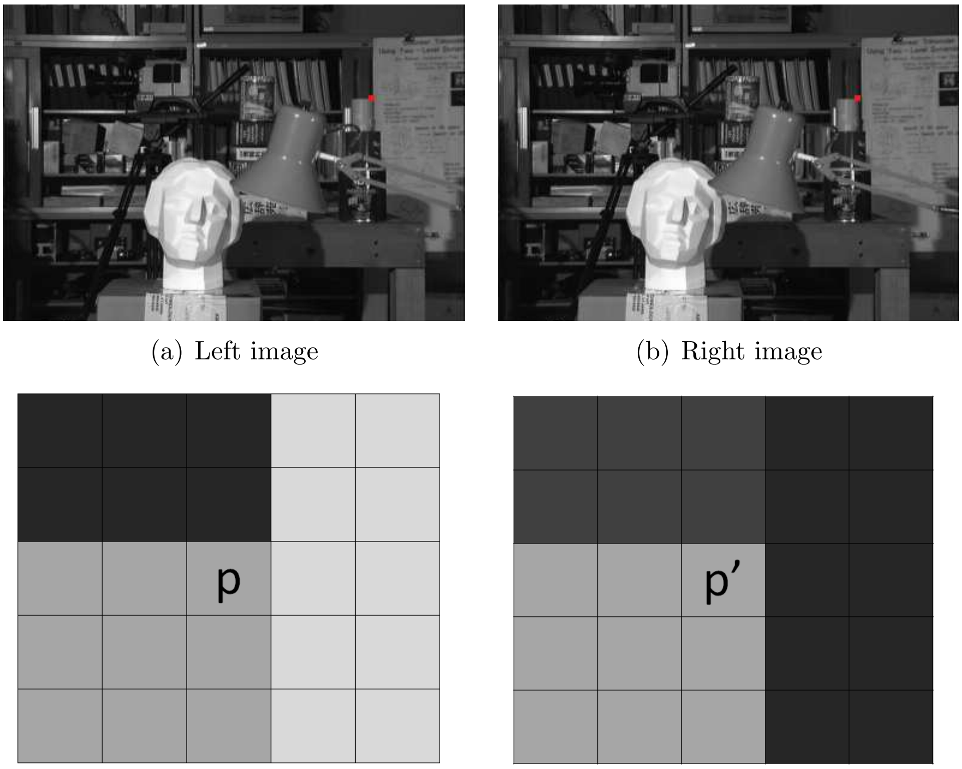
where

$$A = \begin{bmatrix} \nabla_x E(p_1) \\ \nabla_x E(p_2) \\ \vdots \\ \nabla_x E(p_n) \end{bmatrix} \quad \text{and} \quad B = - \begin{bmatrix} E_t(p_1) \\ E_t(p_3) \\ \vdots \\ E_t(p_n) \end{bmatrix}$$

Gradient methods, or optical flow, are quite efficient while well-known to be sensitive to local ambiguities and discontinuities, since only local information is taken into account.

Sparse Feature Matching

The regions around disparity discontinuities have pixels with very different depths, and, therefore, different disparities. Hence window matching methods are not reliable around discontinuities. Figure 5.1 illustrates an example of discontinuities in stereo. Pixel p is located on the can in front of the background. There is a depth (disparity) discontinuity between the can and the background. Therefore, the windows around pixel p and its corresponding pixel p' in the other image have a large matching cost for the correct disparity. This is because a large portion of pixels inside the windows are not in correspondence, making the matching cost high. That is all pixels in the left window that are part of the background do not actually match the overlapping pixels in the right window. Moreover, in textureless regions, window matching and optical



(a) Left image

(b) Right image

(c) Enlarged pixel p and its neighbours

(d) Enlarged pixel p' and its neighbours

Figure 5.1: *The discontinuities in stereo: Figure (a) and (b) are the input stereo images. The red squares show the 5 by 5 windows around a pixel p and its corresponding pixel p' in the right image. These 5 by 5 windows are enlarged in Figure (c) and (d). It shows that these two windows have a large matching cost since pixels above and to the right of pixel p are at depth different from p. Therefore in the left and right windows, the pixels above and to the right of p show two different not corresponding parts of the background, contributing to a large matching cost between the windows. Here the matching cost is the sum of absolute differences.*

flow methods are also easily misguided, because there may be multiple windows of very similar low cost.

Another approach to stereo correspondence is feature matching. Feature matching methods only find disparities of reliable feature points, such as edges [7] [92], curves [80], and textures [83] [90]. These methods produce reliable results where good features are detected. However, results obtained by feature-based methods are usually very sparse since there are no features in regions with uniform texture.

5.2.2 Global Methods

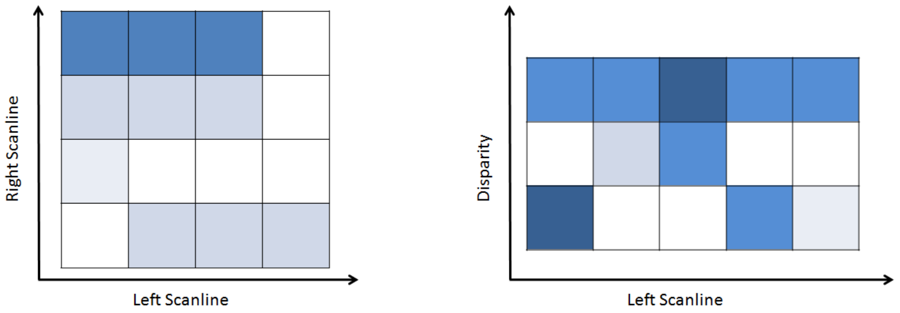
Unlike local correspondence methods mentioned in Section 5.2.1, global correspondence algorithm seek to eliminate ambiguities brought about by discontinuities, occlusion and uniform texture by adding additional constraints that effect the whole image, not only a local region. Like feature-based methods, good features for correspondence can be found in regions which have abundant texture cues. With local constraints, the influence of good features is limited to a small region, which usually can be enlarged if larger neighborhoods are considered. This can be done by applying smoothness constraint over a scan-line or the whole image. Here we list two of the most widely used global approaches in finding visual correspondence.

Dynamic Programming

Dynamic programming is widely used in energy optimization framework in the field of artificial intelligence and computer vision. It reduces the cost of an optimization problem by dividing it into many small sub-problems, each of which can be easily solved. The solution to the sub-problems is then composed to find the optimal solution to the original question.

Stereo can be naturally formulated in a dynamic programming framework. Figure 3.1(a) is an illustration. The horizontal and vertical axis are the corresponding left and right scan-lines of a stereo image pair. In this case, dynamic programming is used to find the minimum cost path from bottom-left corner to top-right corner, as is done by Ohta and Kanade [69] and Cox et al. [16]. Each node on the path, as shown by the white squares in 5.2(a), indicates a matching pixel pair.

Another way of applying dynamic programming in stereo matching is shown in Figure 5.2(b). The axes are defined as the left scan-line and the disparities. The minimum cost path from the first column to the last last column is found by dynamic programming, as is done by Intille and Bobick [43]. The pixel-disparity pairs on the minimum-cost path, as shown by the white squares in Figure 5.2(b) show the result



(a) (b)

Figure 5.2: *Stereo with dynamic programming. In (a), the axes are left and right scanlines. The brightness of the squares shows the matching cost of corresponding pixel pairs. The darker the square is, the greater the matching error is. A path with the minimum matching cost is found from lower left corner, shown by white squares. In (b), the axes are the left scanline and disparities. A path from the fist column to the last column with minimum cost is shown by white squares.*

of the algorithm.

As stated in the previous paragraph, dynamic programming can always find the exact solution to a 2D optimization problem. However, the main problem with dynamic programming methods is that they can only enforce intra-scanline constraints. The inter-scanline constraints are ignored. Two-level dynamic programming is proposed by Cox et al. [16] to solve this problem. Nevertheless it is NP-hard to optimize a matching cost function with dynamic programming if the vertical constraints over the entire image are taken into account.

Graph Cuts

To consider the inter-scanline dependence, one can formulate the stereo correspondence problem in the energy optimization framework. The label set $\mathcal{L} = \{l_0, l_1, l_2, \ldots, l_m\}$, denotes the set of all possible disparities, where m is the maximum disparity. Let $\mathcal{P}$ be the set of all image pixels, and for each $p \in \mathcal{P}$ we wish to assign some label $f_p \in \mathcal{L}$. Thus the stereo problem is now a multi-labeling problem. Let f be the collection of all pixel-label assignments. The following energy function is formulated to measure the quality of f:

$$E(f) = E_{smooth}(f) + E_{data}(f), \tag{5.3}$$

The definition of the data term E_{data} and smoothness term E_{smooth} is the same as Equation 2.3 in section 2.2. That is:

$$E_{data}(f) = \sum_{p \in \mathcal{P}} D_p(f_p), \tag{5.4}$$

where D_p is the penalty for assigning pixel p the disparity f_p. And the smoothness term is:

$$E_{smooth} = \sum_{\{p,q\} \in \mathcal{N}} V_{pq}(f_p, f_q). \tag{5.5}$$

where V_{pq} is the penalty for assigning f_p and f_q to neighbouring pixels p and q. Graph cuts has proved to be very successful in stereo correspondence [12] [91].

Boykov et al. [12] proposed a stereo matching algorithm based on graph cuts. Their method is quite straightforward. Suppose there is a pair of stereo images I and I'. Let $\mathcal{P}$ denotes the set of all the pixels in I. The forward matching cost of pixel $p \in \mathcal{P}$ with disparity d is measured as:

$$C_{fwd}(p, d) = \min_{d-\frac{1}{2} \leq x \leq d+\frac{1}{2}} |I_p - I^{'}_{p+x}|,$$

The backward matching cost C_{rev} is defined symmetrically:

$$C_{rev}(p, d) = \min_{p-\frac{1}{2} \leq x \leq p+\frac{1}{2}} |I_x - I^{'}_{p+d}|,$$

The minimum of C_{fwd}, C_{rev} and a constant is defined as the matching cost $C(p, d)$ for pixel p with disparity d:

$$C(p, d) = \min(C_{fwd}(p, d), C_{rev}(p, d), constant),$$

The data term, E_{data}, as defined in Equation (5.4) is the summation of $C(p, f_p)$ where f_p is a disparity of pixel p. This particular definition of the data cost helps to alleviate image sampling artifacts. It was first proposed by Birchfield and Tomasi [8].

For smoothness constraint, the Potts model is used. Two neighboring pixels are likely to have the same disparity if they have similar intensities. An α-expansion algorithm is applied to get an approximate solution to this energy optimization problem. The results of this algorithm helped to move state-of the art in stereo correspondence forward by showing that global methods can be efficient and produce results by far superior to the window matching methods that were common at the time. However this approach still does not produce confidence in its estimates and can fail for scenes with very low texture.

5.2.3 Semi-dense Visual Correspondence

Our work was inspired by Veksler [91]. As noted in Section 5.2.1, most matching cost functions based on intensity difference can fail in occluded and uniform-textured regions. Moreover, in regions with disparity discontinuities, although they cause ambiguities for local methods, there are useful cues for visual correspondence. The intuition behind this is that in the regions with discontinuities, the matching error should be weaker than the intensity edges. Veklser refers to these cues as "boundary conditions" in [91].

The process of finding visual correspondence is formulated as a labeling problem in [91]. At the first stage the labeling is performed for each disparity separately. For a pixel p and disparity d, the labeling $f_p = 1$ means assigning disparity d to pixel p and $f_p = 0$ means p is of other disparity. In the next step, an energy function in the form of 5.3 is formulated to evaluate the labeling. The data terms $D_p(f_p)$ encodes how pixel p likes disparity d. Since label 1 means pixel p likes the disparity d, $D_p(1)$ should corresponds to the penalty of assigning disparity d to pixel p. The smaller is $D_p(1)$, the more likely label 1 is for pixel p. First the edge strength between p and its neighbor in the left direction, p_l, is measured. That is $\delta_l = |L(p) - L(p_l)|$, where L is the left image of the input stereo pair. Similarly, the edge strength in the right image is measured with $\delta_r = |R(p+d) - R(p_l + d)|$. The the edge strength is $\delta = \min(\delta_l, \delta_r)$. Then the matching error for p and p_l is $e(p) = |L(p) - R(p+d)|$ and $e(p_l) = |L(p_l) - R(p_l + d)|$. The texture cue is $t\_cue = 10 - h(\delta - e(p)) - h(\delta - e(p_l))$, where

$$h(x) = \begin{cases} 10 & \text{if } x < 10 \\ 10 - \frac{x^2}{2.5} & \text{if } 0 \leq x \leq 5 \\ 0 & \text{if } x > 5 \end{cases}$$

The matching error itself is also considered in the data term. First, the matching error is defined as $m\_cue = g(e(p)) + g(e(p_l))$, where $g(x) = 10 - x^2/160$. Finally, $D_p(1)$ is defined as $D_p(1) = \max\{0, \min\{10, (10 - t\_cue) + (10 - m\_cue)\}\}$. The negative cue $D_p(0)$ is the penalty for assigning label 0 to pixel p. In [91] it is defined as:

$$D_p(0) = \max\{0, 10 - \frac{\min\{e^2(p), e^2(p_l)\}}{30}\}$$

For pixel p with disparity d the positive cue $D_p(1)$ measures the boundary condition at the pixel, and the negative cue $D_p(0)$ measures the truncated square difference between p and its corresponding pixel under disparity d.

The distance transform of the boundary feature map is used to calculate the smoothness term v_{pq}. Let δ, $e(p)$ and $h(x)$ have the same definition as that of the $D_p(1)$, then

$$B_l(p) = \begin{cases} \infty & \text{if } \delta < e(p) \\ h(\delta - e(p)) & \text{otherwise} \end{cases}$$

Thus $B_l(p)$ is small if there is likely a boundary condition feature going between p and p_l. The distance map of $B_l(p)$ is then defined as $T_l(p) = \min_{q \in \mathcal{P}}\{B_l(q) + dist(p,q)\}$, where $dist$ is the standard Manhattan distance. Then the smoothness term v_{pq} is defined as:

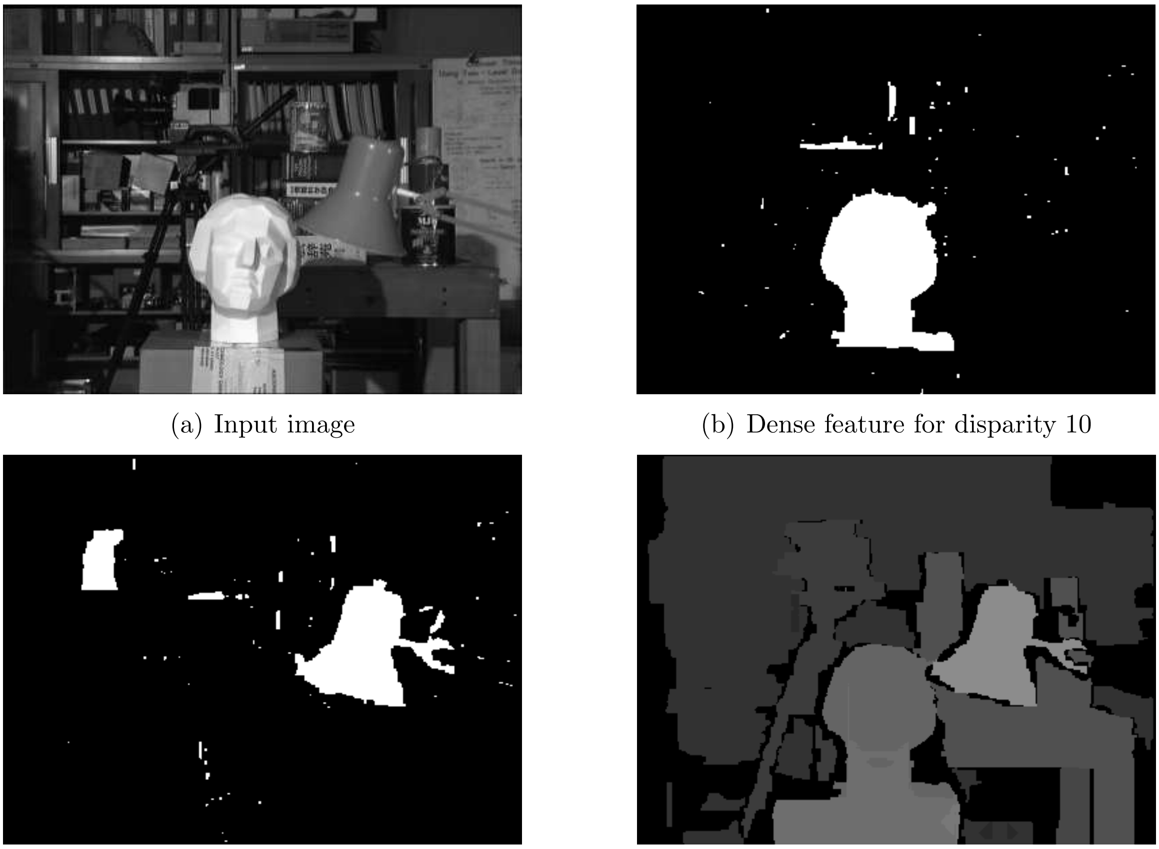
$$v_{pq} = \begin{cases} 1 + B_l(p) & \text{if } B_l(p) \neq \infty \\ 1 + (T_l(p))^2 & \text{otherwise} \end{cases}$$

This allows the boundary cues to pass their support for disparity d to neighbouring pixels. The graph cut algorithm is used by [91] to optimize the cost function. Since the resulting labeling is much denser then the boundary condition feature, it is referred to as dense visual feature.

Any pixel $p \in \mathcal{P}$ can belong to more than one dense visual features. Therefore, the last step of [91] is to resolve the ambiguities. It is done with heuristics. If a pixel p is assigned label 1 for more than one disparity, then p will be assigned to the disparity which has most pixels in the immediate surrounding of p. Figure 5.3 shows the results for [91]. Figure 5.3(a) is the left image of the input stereo pair. Figure 5.3(b) shows the dense feature detected at disparity 10. That is mostly the disparity of the sculpture. In Figure 5.3(c), there are the dense features detected at disparity 14. Figure 5.3(d) is the final result after resolving the ambiguity. The lighter is the pixel, the larger is the disparity.

The main drawback of [91] is that it only allows the pixels with exactly the same disparity to be assigned to the same blob-like visual cue. If there is a large textureless region that straddles several disparities, method in [91] will fail to segment it as a dense feature. Figure 5.4 illustrates an example where the visual correspondences for the large textureless surface with smoothly varying disparities can not be established. Our approach to solve this problem is to group together cues that straddle several disparities, provided they are smoothly varying disparities and are adjacent in the image. The intuition is that an object that we might be able to segment as a dense visual cue (like the ball in Figure 5.3), if it is not at single disparity, then it has a range of smoothly varying disparities.

Another limitation of [91] is that the threshold for the boundary condition is set up by hand. Moreover, the last step of resolving the ambiguities is done with a heuristic.



(a) Input image

(b) Dense feature for disparity 10

(c) Dense feature for disparity 14

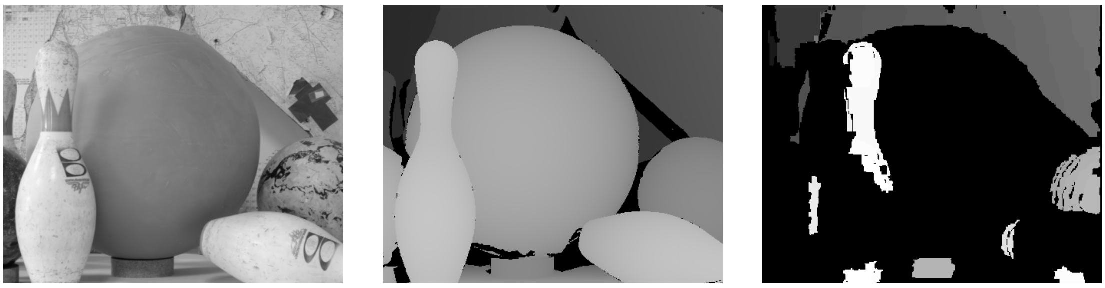
(d) Semi-dense visual correspondence

Figure 5.3: The results of Veksler [91]: Figure (a) is the left input image. Figure (b) shows the dense feature detected at disparity 10. Figure (c) is the dense feature for disparity 14. Figure (d) is the disparity assignment after resolving the ambiguity.

Therefore, we propose a method which selects the useful features for detecting visual correspondence automatically. Thus we can avoid the problem of picking the threshold by hand. Furthermore, we resolve ambiguities between different dense features in a more robust way.

5.3 Overview of Our Method

Our main objective is to find regions in the image for which disparities can be determined more reliably. Like the previous work [90, 91], we require that such regions



(a) The right image (b) Ground truth (c) Result by [91]

Figure 5.4: *The drawback of [91]: Figure (a) is the right image of the input stereo pair. The big bowling ball in the middle of this image has little texture on it and its disparities vary smoothly. Figure (b) is the ground truth. Figure (c) is the result generated by the approach of [91]. Pixels with black colour mean there is no disparity assigned to them. This is because these pixels do not belong to any dense feature. We can see that no visual correspondence is found inside the big bowling ball in the middle because [91] only finds dense features that belong to a single disparity.*

have texture cues on the boundary. In addition they may have texture cues in the interior. Unlike the previous work [90, 91], we allow the disparity range within each such feature blob to vary smoothly, instead of being constant. This produces blob-like semi-dense visual features for which we have a high confidence in their estimated ranges of disparities. To achieve our goal, we develop a three-stage approach for the semi-dense visual correspondence problem.

We start with selecting useful features for visual correspondence at the textured regions. We use the machine learning approach for this purpose [22]. We train a classifier that selects a useful set of features from a large feature pool. This feature pool consists of 148 different useful features which are commonly used for visual correspondence detection, such as matching error, edge strength, difference between the matching error and the edge strength (the so called "boundary condition").

For the training stage, we selected several stereo image pairs with the ground truth from [78, 77, 40] as our training set. The features included in our feature pool are computed in the textured regions of the training image pairs. Ada-boost algorithm is then used to select the informative features that perform well in determining the

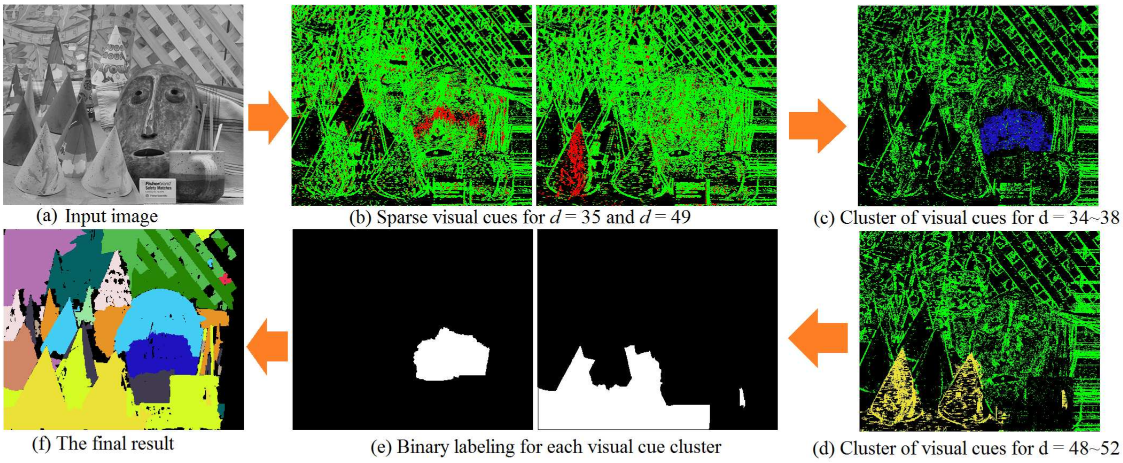

(a) Input image
(b) Sparse visual cues for $d = 35$ and $d = 49$
(c) Cluster of visual cues for d = 34~38
(d) Cluster of visual cues for d = 48~52
(e) Binary labeling for each visual cue cluster
(f) The final result

Figure 5.5: *The main steps of our semi-dense visual correspondence method. Figure (a) is the input image. Figure (b) shows the sparse visual cues detected by our classifier at disparity 35 and 49. The red pixels are the positive cues which support disparity 35 and 49, and the green pixels are the negative cues which do not support disparity 35 or 49. The black pixels neither like nor dislike disparities 35 or 49. Figure (c) and (d) are two of our visual cue groups. Here the blue pixels in Figure (c) show the cluster of positive visual cues which support the range of disparities from 34 to 38. The yellow pixels in Figure (d) are the visual cues which support disparity range from 48 to 52. The green pixels are the negative cues which do not support these disparity ranges. Figure(e) shows the binary labeling results for the visual cue groups in Figure (c) and (d). Figure (f) is the final result after ambiguity resolving.*

disparities of the pixels in the training images. This results in a binary classifier which determines if one pixel p agrees or disagrees with a disparity d in the textured regions. With this classifier, we can then detect sparse visual cues in the textured regions of the input image for each disparity separately. The visual cues detected by our classifier either support or do not support the associated disparity. We refer to the visual cues which support a certain disparity as *positive* cues. For the visual cues that disagree with the associated disparity, we call them *negative* cues.

Figure 5.5(b) shows the results of our binary classifier at disparity 35 and 49 for the input image in Figure 5.5(a). The red pixels in Figure 5.5(b) are the positive visual cues which support disparity 35 or 49. The green pixels are the negative visual cues which do not support these disparities.

In the second step of our approach, we apply linkage clustering on the positive cues detected in the previous step. Thus we cluster these sparse positive cues into denser groups $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$, where m is the number of groups and is a parameter set by the user. In each group $g \in \mathcal{G}$, the positive visual cues are spatially near each other, and their associated disparities are adjacent. We group the positive cues across different disparities in order to be able to handle dense visual features that strand several disparities. As mentioned before, we do expect that a dense feature will have a smoothly varying range of disparities. Figure 5.5(c) and 5.5 (d) illustrate two of the visual cue groups for the image in Figure 5.5(a). The blue pixels form the visual cue group which supports disparity range from 34 to 38. Pixels in this group form the middle part of the mask. The yellow pixels vote for disparity range from 48 to 52. This is the disparity range for the two cones in the front row. The green pixels do not support any disparity in these ranges.

Notice that the visual cues obtained for each disparity range are rather sparse. Our next step is to obtain denser cues (or "features"). This step is performed as a binary labeling problem, separately for each group of the positive cues. For group g, if a pixel p is labeled 1, then it means this pixel has a disparity within the range associated with group g. Label 0 means that p does not belong to group g. This binary labeling process produces a set of blob-like visual cues for each group, while pixels in these blob-like cues support the disparity range associated with the group. See Figure 5.5(e) for the results of a binary labeling of the visual cue groups in Figure 5.5(c) and (d).

After the previous step, each pixel can belong to zero, one, or more of the visual groups. Thus we need to resolve the ambiguities to find the exact boundaries between the regions of different disparity groups. This is the next step of our method, and we formulate it as a multi-labeling problem. The label set $\mathcal{L} = \{l_1, l_2, \ldots, l_m\}$ consists of m labels. Each label $l_i, i \in \{1, 2, \ldots, m\}$ represents a group $g_i \in \mathcal{G}$. Thus assigning a label $l_i \in \mathcal{L}$ to a pixel $p \in \mathcal{P}$ means that pixel should only belong to group $g_i \in \mathcal{G}$. By solving this multi-labeling problem with the graph cut approach [12], we find regularized boundaries between blobs of pixels which belong to different groups. Figure 5.5(e) shows the final result of semi-dense visual correspondence for the image in Figure 5.5(a). Pixels with the same colour belong to the same disparity group, and the disparities in these regions vary smoothly.

The grouping stage (second stage) and labeling stage (third stage) can be iterated, if we merge small blobs of our semi-dense visual correspondence with larger blobs according to their locations and associated disparity group. This will result in bigger groups of blob-like visual cues and produces smoother boundaries and surfaces. The resulting groups are more reasonable if we start with a large number of groups and decrease this number with each iteration. Performing multi-labeling based on these new groups will generate more accurate results. Therefore, finally we iterate our grouping stage and labeling stage until the results can not be improved anymore.

5.4 Detailed Description of Our Algorithm

In this section, we describe our semi-dense visual correspondence method in details. We start with describing our classifier for selecting useful sparse matching features in section 5.4.1. Next, in section 5.4.2, we introduce the clustering algorithm that groups the sparse visual cues into denser clusters. In section 5.4.3, we apply binary labeling on the visual cue groups to generate blob-like features for stereo. In section 5.4.4, we formulate our semi-dense visual correspondence problem as a multi-labeling problem and solve it with the graph cuts algorithm. In section 5.4.5, we illustrate how we iterate the clustering stage and labeling stage to improve the results.

5.4.1 Detecting Sparse Features for Stereo

The first stage of our approach is to build a binary classifier which, given a pixel in the textured region, determines whether this pixel supports a particular disparity or not. Therefore, to build such a classifier, we need to collect a set of features with discriminant power for visual correspondence detection. For instance, if we want to measure how disparity d fits a pixel p, then the absolute intensity difference between pixel p in the right image and pixel $p' = p + d$ in the left image is a good indication. In our approach, our training images and most of our testing images are from the Middlebury stereo data sets [76, 78, 77, 40], and the ground truth of these images is based on the right images. Therefore, for our approach, we use the right image as the reference image, and matching the left image to the right image. In [91], the author claims that if the matching error is lower then the edge strength at pixel p by some threshold t, then it is likely that the associated disparity fits pixel p. This implies that in the textured regions, measuring the difference between the matching cost and the edge strength is a good indication for visual correspondence. In this thesis, we also refer to the difference between the matching cost and the edge strength as the "boundary condition". Other important features for detecting visual correspondence include shifted matching cost [8], boundary conditions based on shifted matching cost and edge strength, the difference between matching cost of neighbouring pixels, etc.

Our pool of features can be divided into two major categories, the basic features and combined features. First, let us define the basic features. There are four types of basic features in our feature set. Let I_l and I_r be the left and right image of the input stereo image pair. Let $\mathcal{P}$ denotes the set of all pixels in I_r. The absolute intensity difference between corresponding pixels is then defined as:

$$E_{abs}(p, d) = |I_r(p) - I_l(p + d)|$$

Here $p \in \mathcal{P}$ is a pixel in the right image I_r and d is a possible disparity. E_{abs} measures the intensity difference between pixel p and its corresponding pixel $p + d$ in the left image.

We also adopt shifted matching cost from the work of Brichfield and Tomasi [8]. It is defined as following:

$$\overline{d}(p+d, p, I_l, I_r) = \min_{p-\frac{1}{2} \le x \le p+\frac{1}{2}} |I_l(p+d) - \hat{I}_r(x)|$$

Here $\hat{I}_r$ is the linearly interpolated image of I_r and d is the disparity. Similarly we can define:

$$\overline{d}(q-d, q, I_r, I_l) = \min_{q-\frac{1}{2} \le x \le q+\frac{1}{2}} |\hat{I}_l(x) - I_r(q-d)|$$

where $\hat{I}_l$ is the linearly interpolated left image I_l, and $q = p + d$ is the corresponding pixel of pixel p in the left image. Then the matching cost between pixel p in the right image and its corresponding pixel $q = p + d$ in the left image is defined as:

$$E_{shifted}(p, d) = \min(\overline{d}(p+d, p, I_l, I_r), \overline{d}(q-d, q, I_r, I_l))$$

The shifted matching cost measures the difference between the linearly interpolated pixels. This reduces errors brought by the sampling artifacts and image noise.

Our basic features also include boundary conditions. To compute the boundary conditions, we need to define the edge strength at pixel p. We first define:

$$e_r(p) = \max(|I_r(p) - I_r(p-1)|, |I_r(p) - I_r(p+1)|)$$

$$e_l(p) = \max(|I_l(p+d) - I_l(p+d-1)|, |I_l(p+d) - I_l(p+d+1)|)$$

These are the intensity changes between pixel p and its neighbours on both left and right sides in the input image pair. Then the edge strength at pixel p is:

$$e(p) = \max(e_r(p), e_l(p))$$

There are two types of boundary conditions defined in our feature set: the difference between edge strength and absolute intensity difference, and the difference between edge strength and the shifted matching cost:

$$B_{abs}(p,d) = e(p) - E_{abs}$$

$$B_{shifted}(p,d) = e(p) - E_{shifted}$$

The boundary condition features measure how a disparity fits a pixel in a textured region by comparing the matching error with the edge strength. It is more robust than both absolute matching error and shifted matching error in the textured regions. Thus our set of basic features is:

$$Basic(p,d) = \{E_{abs}(p,d), E_{shifted}(p,d), B_{abs}(p,d), B_{shifted}(p,d)\}$$

Besides the basic features, we also develop a set of combined features. The combined features are inspired by the window matching methods. For pixel $p \in \mathcal{P}$, its nearby pixels in a small neighbourhood may provide more useful information for the visual correspondence than only the basic features extracted from the pixel itself. For example, for disparity d, if both pixel p and its neighbours have small matching error and good boundary condition, then it is more likely that d is the correct disparity for pixel p. To compute the combined features, for every pixel $p(x,y) \in \mathcal{P}$, we first compute the basic features (absolute intensity difference, shifted matching cost and boundary conditions) for the pixels in a 3×3 neighbourhood $\mathcal{N} = \{q(i,j) | (x-1) \leq i \leq (x+1), (y-1) \leq j \leq (y+1)\}$. Then the combined features are defined as the combinatory difference between the basic features of the pixels in $\mathcal{N}$:

$$C_{abs}(p,d) = \{|E_{abs}(q,d) - E_{abs}(r,d)|, q, r \in \mathcal{N}, q \neq r\}$$

$$C_{shifted}(p,d) = \{|E_{shifted}(q,d) - E_{shifted}(r,d)|, q, r \in \mathcal{N}, q \neq r\}$$

$$C_{boundary\_1}(p,d) = \{|B_{abs}(q,d) - B_{abs}(r,d)|, q, r \in \mathcal{N}, q \neq r\}$$

$$C_{boundary\_2}(p,d) = \{|B_{shifted}(q,d) - B_{shifted}(r,d)|, q, r \in \mathcal{N}, q \neq r\}$$

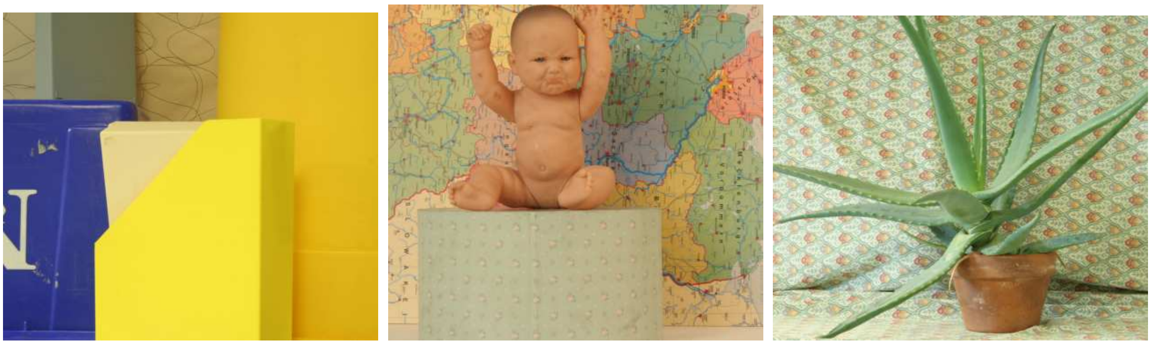
Here C_{abs} measures the difference between the absolute matching error for pixels in the 3 by 3 neighbourhood. $C_{shifted}$ is the difference between the shifted matching cost for pixels in the small window. $C_{boundary\_1}$ is the set of combined feature for boundary conditions based on absolute intensity difference. $C_{boundary\_2}$ is the same with $C_{boundary\_1}$, and the only difference is that the boundary condition is now the difference between the shifted matching error and the edge strength. By computing the combined features, we can infer the information about the visual correspondence from the neighbouring pixels of pixel p. Therefore, they provide more support for the visual correspondence for pixel p than the basic features. Finally our feature set for pixel p at disparity d is:

$$F(p,d) = C_{abs}(p,d) \bigcup C_{shifted}(p,d) \bigcup C_{boundary\_1}(p,d) \bigcup C_{boundary\_1}(p,d) \bigcup Basic(p,d) \tag{5.6}$$

Our training set consists of stereo pairs with different portion of textured regions. We first measure the ratio of number of pixels whose edge strength is greater than 10 over the total number of pixels in a training image:

$$r_{texture} = \frac{|\{p|e(p) > 10, p \in \mathcal{P}\}|}{|\mathcal{P}|}$$

Then we selected several stereo image pairs from the Middlebury stereo image data set [76]. And their $r_{texture}$ range from 1.33%, to 17.28% respectively. The training examples are pixels in these image pairs whose edge strength is greater than 10. Therefore, our training set covers examples from modestly textured images to highly textured images. The training examples are split into positive and negative sample sets. The features for the positive examples are computed with the ground truth provided with Middlebury stereo set. For the negative examples, their disparities are randomly picked and are different from the ground truth. Figure 5.6 shows three



(a) (b) (c)

Figure 5.6: *The training set for the sparse visual cues classifier: the ratio $r_{texture}$ of the textured pixels in Figure (a), (b), and (c) are 1.33%, 7.16% and 17.28% respectively. Our training set covers examples from modestly textured images to highly textured images.*

images in our training set.

Ada-boost [30, 75, 74] is used in our work as a classifier. Ada-boost, short for Adaptive Boosting, is a machine learning algorithm which can be used in conjunction with other learning algorithms to improve their performance. The basic idea behind Ada-boost is that the performance of a set of weak classifiers can be improved gradually by combining them together sequentially. The subsequent classifiers are adapted to favour those examples which are misclassified by the previous classifiers.

Ada-boost procedure can also be interpreted as a greedy feature selection process. If weak learners used in Ada-boost are simply one-node decision trees which best separate the positive and negative examples, then for each feature, the weak learner algorithm determines the optimal threshold such that the number of misclassified examples is minimum. In each iteration of the Ada-boost algorithm, the weak classifier which has least classification error is selected and combined with the previously selected classifiers. With the one-node decision trees based on single feature, the Ada-boosting algorithm acts as a greedy feature selection algorithm.

To complete our learning approach, we now need to construct a set of simple classifiers based on our feature set F defined in Equation (5.6). A weak classifier $h_j(x)$ consists of a feature $f_j \in F$, a threshold θ_j and a polarity p_j.

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

Here x is any pixel in the input stereo image pair whose edge strength $e(x)$ is greater than 10. Since the features are only reliable in the textured regions, we ignore the pixels with little intensity change.

Now we are ready to construct our learning framework based on Ada-boosting. The weak classifiers defined in Equation (5.7) usually perform just slightly better than random (with classification error slightly lower than 50%). Therefore, we employ Ada-boost algorithm to combine these weak classifiers together to improve their performance. Algorithm 1 shows the detailed procedure for our learning framework based on Ada-boost. In this work, we use the implementation of Vezhnevets et al. [93]. By using Ada-boost, we construct a binary classifier which detects the pixels that strongly agree or disagree with a particular disparity d. Since we only use pixels whose edge strength is greater than 10 in our training set, we only apply the final classifier to testing examples whose edge strength are also greater than 10. The classification results for this classifier are sparse visual cues that are located in the textured regions, see Figure 5.5(b).

5.4.2 Visual Cues Clustering

The classifier constructed in the previous step can detect positive and negative visual cues for stereo in mildly textured regions. This step is performed separately for each disparity. The classification results are sparse visual cues which either support or reject the associated disparity, see Figure 5.5(b). For images with large textureless surfaces, the visual cues detected by our classifier are too sparse. Furthermore, our

Algorithm 1 The Ada-boost algorithm for learning the useful features for stereo correspondence. Each weaker classifier $h_j(x)$ is a simple decision boundary based on a single feature, defined in Equation (5.7). The final classifier is a weighted linear combination of T simple classifiers.

```
• Given a set of training samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $x_i$ are the pixels in the training images, and $y_i = 0, 1$ for the positive and negative examples respectively.
• Initialize weights $w_{1,i} = \frac{1}{n}$ where $n$ is the number of training samples.
for $t = 1, 2, \ldots, T$: do
   1. Normalize the weights:
      $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=i}^{n} w_{t,j}}$
   2. For each feature, $j$, train a classifier $h_j$ which best separates the positive and negative samples. The error of $h_j$ is evaluated by $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
   3. Choose the classifier $h_t$ with the lowest error $\epsilon_t$.
   4. Update the weights:
      $w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$
   where $e_i = 0$ if example $x_i$ is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
end for
• The final strong classifier is:
   $h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$
where $\alpha_t = \log \frac{1}{\beta_t}$
```

goal is to allow detection of a visual feature that may straddle several disparities, as long as the disparity variation inside that feature is smooth. Therefore, in the second step of our approach, we propose a clustering method that groups together the sparse visual cues detected in the previous step. The resulting visual cue groups consist of sparse visual cues which have smoothly varying disparities, and which are spatially close to each other, see Figure 5.5(c) and (d).

To cluster the sparse visual cues into larger groups, we first need to define the distance measure between two visual cues. Let the 4-tuple (x, y, d, l) denotes a visual cue detected by our classifier. Here x and y are the horizontal and vertical coordinates of the pixel, d is the associated disparity. Label $l \in \{0, 1\}$ is a binary indicator. If $l = 1$, then this visual cues is a positive cue which indicates that pixel (x, y) supports disparity d. If $l = 0$, then this is a negative cue which means that pixel (x, y) does not support disparity d. Then the distance between two visual cues $c_1 = (x_1, y_1, d_1, l_1)$ and $c_2 = (x_2, y_2, d_2, l_2)$ is:

$$D(c_1, c_2) = \begin{cases} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} & \text{if } |d_1 - d_2| \le t \\ \infty & \text{otherwise} \end{cases} \tag{5.8}$$

Here t is the threshold for the disparity difference between c_1 and c_2. Since we want our visual cue groups to have smoothly varying disparities, only neighbouring visual cues with small disparity difference are allowed to be grouped together. In practice, we set $t = 1$. $D(c_1, c_2)$ measures the Euclidian distance between two visual cues, provided that their associated disparities are close to each other. Neighbouring visual cues with similar disparities will have small distance between each other, and are more likely to be grouped together.

Let $C_{positive} = \{c_1, c_2, \ldots, c_m | l_i = 1, i = 1, 2, \ldots, m\}$ be the set of all positive visual cues. Our goal is to dived $C_{positive}$ into K subsets $C_1, C_2, \ldots, C_K$ so that:

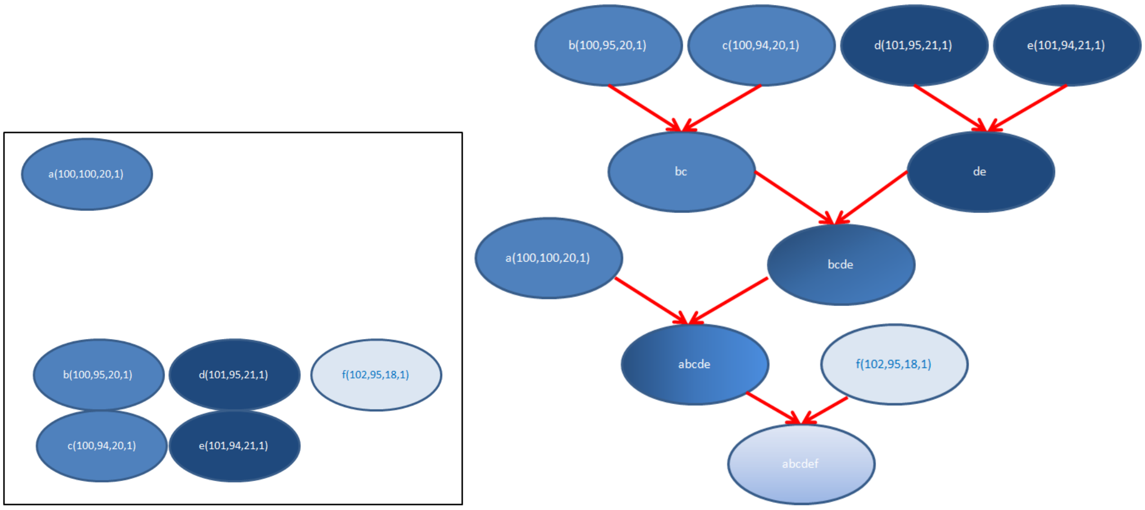
$$\bigcup_{1 \le j \le K} C_j = C_{positive},$$

and

$$C_i \bigcap C_j = \emptyset \text{ for } i, j \in \{1, \ldots, K\} \text{ and } i \ne j.$$

Moreover, the disparities of the visual cues in the same group should vary smoothly, and the locations of these visual cues should be close to each other. Our grouping method is only applied on positive cues. The negative cues will be clustered separately after we have groups of positive cues.

Linkage Clustering is used as our clustering algorithm. Linkage clustering is an agglomerative clustering method which builds a hierarchy of clusters that can be represented in a tree structure. It creates the cluster hierarchy from individual elements by merging the clusters progressively. Figure 5.7 shows a simplified example of our



(a) Visual cues

(b) Linkage clustering process

Figure 5.7: *The linkage clustering process: Figure (a) shows a synthetic example of six positive visual cues. Figure (b) shows the linkage clustering process based on the samples given in Figure (a).*

linkage clustering algorithm.

In Figure 5.7(a), the ovals illustrates six positive visual cues a, b, c, d, e and f, detected by our classifier. Visual cue a, b and c support disparity 20. Visual cue d and e agree with disparity 21. Visual cue f has disparity 18. Here b, c, d, e and f are adjacent to each other, and a is some distance away from the other visual cues. Since we do not want to cluster together visual cues whose disparities are quite different from each other, visual cue f will have infinite distance from the other visual cues, as defined in Equation (5.8).

Our linkage clustering algorithm is first initialized with each example in a singleton cluster. Then in each iteration of the linkage cluster algorithm, two closest clusters are found and merged together. This step is repeated until the total number of cluster is equal to K, which is set by the user. For the example in Figure 5.7(a), visual cue b and c are clustered together to form the first non-singleton cluster. Visual cue d and e are then clustered together in the second iteration, since they have least distance compared with other pairs of clusters. In the third iteration, cluster bc and cluster *de* are merged together because they are adjacent to each other and have similar disparities. In the fourth iteration, visual cue a is merged with cluster *bcde*. Then visual cue f is clustered together with other visual cues in the last iteration, since f has disparity 18 and the others have disparity 20 or 21. The clustering process of this synthetic example is shown in Figure 5.7(b).

The most important step in linkage clustering algorithm is measuring the distance between two clusters. Single-linkage algorithm uses the minimum distance between examples from the two different clusters. Complete-linkages algorithm measures the distance between two clusters by computing the maximum distance between examples from these two clusters. There are also other methods using average distance or mean distance.

In our visual cue clustering methods, we want the resulting groups to have smoothly varying disparity, and the geometric locations of the visual cues in the same cluster should be close to each other. Therefore, we use Ward's method introduced in [96]. Ward's method measures the distance between two clusters by computing the increase in variance for the cluster being merged. The visual cue clusters generated with Ward's method have smoother disparities compared with the results of other methods such as single and complete linkage.

Now we have clusters of positive cues $C_1, C_2, \ldots, C_K$ whose disparities vary smoothly. And the visual cues within the same cluster are located close to each other, see Figure 5.5(c) and (d). In the next step, we need to cluster together the negative visual cues.

For a cluster of positive cues, C_i, let $D_i = \{d_{i1}, d_{i2}, \ldots, d_{il}\}$ be the range of disparities for all the visual cues in cluster C_i. Let N_{ij} be the set of negative visual cues which do not support disparity $d_{ij} \in D_i$. That is, if a visual cue $c = (x, y, d, l)$ and $c \in N_{ij}$, then we have $d = d_{ij}$ and $l = 0$ (Recall that for a visual cue $c = (x, y, d, l)$, x, y are the coordinates of the visual cue, d is its associated disparity, and $l \in \{0, 1\}$ shows if its a positive cue or negative cue). Let N_i be the cluster of negative visual cues which disagrees with the disparity range D_i. Then N_i is computed by the following procedure. For every pixel $(x, y) \in \mathcal{P}$, if $c = (x, y, d_{ij}, 0) \in N_{ij}$ for all $d_{ij} \in D_i$, then $c = (x, y, d_{ij}, 0) \in N_i$ for all $d_{ij} \in D_i$.

Informally, N_i is the intersection of N_{ij} regardless of their disparities. Figure 5.5(c) and (d) shows two examples of the visual cue groups. Notice the clusters of negative visual cues, shown with green pixels, are much less dense than the negative cues for a single disparity (shown with the green pixels in Figure 5.5(b)). That is because we only alow visual cues which disagree with every disparity in D_i to be added into the negative cue cluster.

5.4.3 Blob-like Visual Cues

In the previous step, we clustered the sparse visual cues detected by our binary classifier into lager groups. Within each group, the sparse visual cues have smoothly varying disparities and are spatially close to each other. The purpose for the clustering step is to enable our approach to deal with textureless regions which strand several disparities. Our goal is to propagate the visual cues detected at the boundaries of the textureless regions into the interior of these regions. Therefore, in this step, for each group of sparse visual cues, we generate dense, blob-like visual features which support the disparity range associated with this group, see Figure 5.5(e).

We formulate this problem as a binary labeling problem. Recall that in the previous step, we first cluster all the positive cues detected by our classifier into several clusters $C_1, C_2, \ldots, C_K$, where K is the number of clusters and is set by the user. The positive visual cues inside cluster $C_i, 1 \leq i \leq K$ support the disparity range $D_i = \{d_{i1}, d_{i2}, \ldots, d_{il}\}$. We also cluster the negative cues together by computing the intersection of $N_{i1}, N_{i2}, \ldots, N_{il}$, which are the sets of negative cues that do not support disparity $d_{i1}, d_{i2}, \ldots, d_{il}$ respectively. The resulting negative cue cluster is denoted as N_i.

Our binary labeling framework for blob-like dense visual feature is formulated as following. For a certain group of positive cues C_i, we need to assign a label $f_p \in \{0, 1\}$ to every pixel $p \in \mathcal{P}$, where $\mathcal{P}$ is the set of all pixels in the right image. If pixel $p(x, y) \in \mathcal{P}$ is labeled with 1, then p supports the disparity range $D_i = \{d_{i1}, d_{i2}, \ldots, d_{il}\}$. Otherwise, pixel p does not support any disparity in D_i. Let f be the collection of the labeling for all the pixels. An energy function is formulated to evaluate f:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{pq}(f_p, f_q). \tag{5.9}$$

Here the data term $D_p(f_p)$ is defined as following:

$$D_p(f_p) = w_m \cdot D_p^{matching}(f_p) + w_f \cdot D_p^{feature}(f_p). \tag{5.10}$$

The matching error term $D_p^{matching}(f_p)$ is the matching cost for assigning label f_p to pixel p. And the feature term $D_p^{feature}(f_p)$ encodes both the positive cues in group C_i and the negative cues in group N_i.

$D_p^{matching}(1)$ is defined by:

$$D_p^{matching}(1) = \min\{20, \min_{1 \le j \le l}(E_{abs}(p, d_{ij}))\},$$

where $E_{abs}(p, d_{ij})$ is the absolute intensity difference between pixel p in the right image and pixel $p + d_{ij}$ in the left image. It is defined in Section 5.4.1. $D_p^{matching}(1)$ measures the minimum matching cost for assigning disparities in group D_i to pixel p. We take the minimum matching cost, because intuitively pixel p only has one disparity d, and if $d \in D_i$, then the matching cost for assigning d to pixel p should be the smallest compared with that of the other disparities in D_i. We also truncate the data term so that we will not over-penalize for assigning label 1 to pixel p with very high matching cost for all the disparities in D_i. $D_p^{matching}(0)$ is computed with the following equation:

$$D_p^{matching}(0) = \max\{0, 20 - \max_{1\leq j\leq l}(E_{abs}(p, d_{ij}))\}.$$

$D_p^{matching}(0)$ is simply the reverse of $D_p^{matching}(1)$.

$D_p^{feature}(1)$ for pixel $p(x, y) \in \mathcal{N}$ is defined as the following:

$$D_p^{feature}(1) = \begin{cases} 10 & \text{if there is } c = (x, y, d, 0) \text{ and } c \in N_i \\ 0 & \text{otherwise} \end{cases}$$

Therefore, $D_p^{feature}(1)$ is the penalty for assigning label 1 to pixel p where there is a negative cue at pixel p. Similarly, we define $D_p^{feature}(0)$ as:

$$D_p^{feature}(0) = \begin{cases} 10 & \text{if there is } c = (x, y, d, 1) \text{ and } c \in C_i \\ 0 & \text{otherwise} \end{cases}$$

Thus $D_p^{feature}(0)$ is the penalty for assigning label 0 to pixels with positive cues. By encoding the sparse visual cues in $D_p^{feature}(f_p)$, we have more confidence about the disparity range of the resulting dense visual cues.

The smoothness term $V_{pq}(f_p, f_q)$ is the Potts model defined in Section 2.1. That is $V_{pq}(f_p, f_q) = u_{pq} \cdot T(f_p \neq f_q)$, here $T(f_p \neq f_q) = 1$ if $f_p \neq f_q$, and $T(f_p \neq f_q) = 0$ otherwise. Because disparity changes tend to coincide with intensity changes, we define u_{pq} as:

$$u_{pq} = \begin{cases} 2K & \text{if } |I_p - I_q| \leq 5 \\ K & \text{otherwise} \end{cases} \tag{5.11}$$

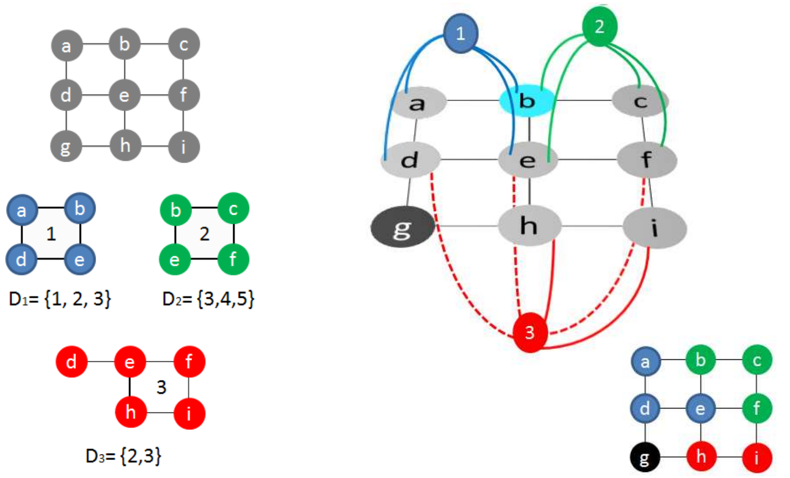
Here K is the Potts model parameter. The neighbourhood system $\mathcal{N}$ is the standard 4-connected neighbourhood.

Thus our energy function encodes the matching error ($D_p^{matching}$) and sparse visual cues ($D_p^{feature}$) in the data term. And contextual information is taken into account by encoding the intensity changes u_{pq} in the smoothness term. The graph cut algorithm [13] is used to optimize the energy function defined in Equation (5.9). Since our labeling problem is only binary, the energy function is optimized exactly with the graph cut algorithm. For each sparse positive cue group C_i, the resulting dense features are blobs of pixels which support the disparity range D_i associated with group C_i, see Figure 5.5(e). By encoding the sparse visual cues in our energy function, we have more confidence in the disparity ranges of our dense visual features. Most of the pixels in the textureless regions are covered by our dense visual features, given there are some sparse visual cues detected at the boundaries of these regions.

5.4.4 Semi-dense Visual Correspondence

The blob-like, dense features generated in the previous step consist of pixels whose disparities are within the disparity range associated with that dense feature. We have more confidence in the disparity range of these features since we encode the sparse visual cues in our energy function. However, for a pixel p in the right image, it can be assigned to zero, one or more dense features. Thus we need to resolve the ambiguities so that each pixel can only belong to zero or one dense feature. We formulate this problem as a multi-labeling problem.

Let $H_i = \{p_{i1}, p_{i2}, \ldots, p_{in_i}\}$ denote the set of pixels which belong to the dense feature that supports the disparity range $D_i = \{d_{i1}, d_{i2}, \ldots, d_{il}\}$ of positive cue group C_i. Figure 5.8(a) shows three synthetic examples of the dense visual feature. Let $H = \bigcup_{1 \leq i \leq K} H_i$ be the set pixels in all the dense visual features. Our label set is $\mathcal{L} = \{1, 2, \ldots, K\}$, where each label $i, 1 \leq i \leq K$ represents the dense visual feature H_i. Our goal is to assign a label $l \in \mathcal{L}$ to each pixel $p(x, y) \in H$ in the right image, so that p supports the associated disparity range of dense feature H_l, see Figure 5.8(b). For pixels which are not in any dense feature, that is $p \notin H$, we do not assign any label to them since there is no visual cue detected at these pixels.



(a) Input image and the dense features

(b) The labeling process and the result

Figure 5.8: *A simple artificial example to illustrate the semi-dense correspondence algorithm: The input image in Figure (a) consists of pixel a to i. There are 3 dense visual features detected for this image. The blue feature has a disparity range from 1 to 3. The green feature supports disparity range from 3 to 5. The red feature supports disparity range from 2 to 3. Notice pixel g is not covered by any dense visual feature. In Figure (b), a multi-labeling problem is constructed to resolve the ambiguities between the three dense features shown in Figure (a). The label set consists of three labels* $\{1, 2, 3\}$*, which correspond to the blue, green and red dense features. For each pixel covered by the dense features, we want to assign one label in* $\{1, 2, 3\}$*. For instance, pixel b is included in both the blue and the green dense features. Therefore, we need to assign a label in* $\{1, 2\}$ *to pixel b. Pixel g is not covered by any dense visual feature, therefore, it is not assigned any label. The answer to the multi-labeling problem is a semi-dense visual correspondence, where each pixel either supports a disparity range or is not covered, as shown in Figure (b).*

We also formulate an energy function which is similar to Equation (5.9) to evaluate the labeling f:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{pq}(f_p, f_q).$$

The data term $D_p(f_p)$ also measures if pixel p likes label f_p:

$$D_p(f_p) = \begin{cases} w_m \cdot D_p^{matching}(f_p) + w_f \cdot D_p^{feature}(f_p) & \text{if } p \in H_{f_p} \\ \infty & \text{otherwise} \end{cases}$$

Since only pixels in the dense feature H_{f_p} are confident about the disparity range D_{f_p} associated with H_{f_p}, we want to assign label f_p only to the pixels in H_{f_p}. Therefore, the penalty of assigning label f_p to pixel $p \notin H_{f_p}$ will be infinite. For pixels inside H_{f_p}, we first measure the matching cost of assigning label f_p to pixel p. Let $D_{fp} = \{d_{f_p1}, d_{f_p2}, \ldots, d_{f_pl}\}$ denote the range of disparity associated with dense feature H_{f_p}. Then the matching cost term $D_p^{matching}(f_p)$ is:

$$D_p^{matching}(f_p) = \min\{20, \min_{1 \leq j \leq l}(E_{abs}(p, d_{f_pj}))\}.$$

This is the same as the matching cost term defined in the previous step. The feature term $D_p^{matching}(f_p)$ for pixel $p(x,y) \in H$ is now defined as:

$$D_p^{feature}(f_p) = \begin{cases} \max\{(0 - D_p^{matching}(f_p)), -10\} & \text{if there is } c = (x,y,d,1) \text{ and } c \in C_{f_p} \\ 10 & \text{if there is } c = (x,y,d,0) \text{ and } c \in N_{f_p} \\ 0 & \text{otherwise} \end{cases}$$

Here N_{f_p} is the set of negative sparse cues associated with dense feature H_{f_p}, and C_{f_p} is the set of positive sparse visual cue for H_{f_p}. If there is a positive visual cue for H_{f_p} located at pixel $p(x,y)$, then we decrease the penalty of assigning f_p to p. If the visual cue at $p(x,y)$ is a negative visual cue for H_{f_p}, we increase the penalty of assigning f_p to p. If there is no sparse visual cue located at p, then the feature term is zero.

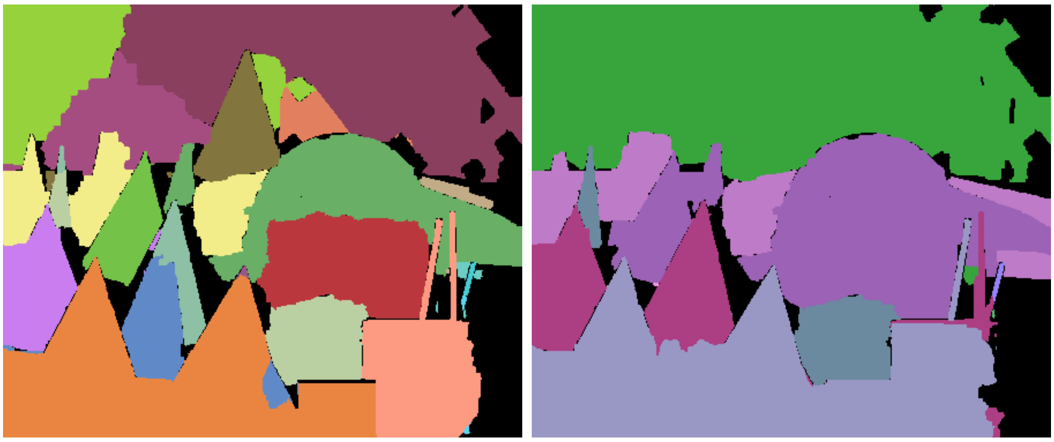
The smoothness term V_{pq} is defined as the same with the smoothness term used in the previous step. That is $V_{pq}(f_p, f_q) = u_{pq} \cdot T(f_p \neq f_q)$, where $T(f_p \neq f_q) = 1$ if $f_p \neq f_q$, and $T(f_p \neq f_q) = 0$ otherwise. Here u_{pq} encodes the intensity change between pixel p and q, as defined in Equation (5.11). This Potts model V_{pq} encourages the final labeling f to be piecewise constant, as described in Section 2.1. Since we aim at dense visual cues which support a range of disparities with confidence, the labels for neighbouring pixels within the same dense visual cue should the same. And changes of the labels are only allowed at the boundaries of the dense visual cues, see Figure 5.5(f). Thus our desired labeling is piecewise constant, which is encouraged by our smoothness term. The neighbourhood system $\mathcal{N}$ is also the standard 4-connect neighbourhood in this step.

Since our V_{pq} is not convex, as mentioned in Section 2.1, it is NP-hard to optimize exactly. We use the α-expansion algorithm [12] to find an approximate solution to our multi-labeling problem. The answer is within a known factor from the global optimal. The result of this step is a semi-dense visual correspondence, where each pixel is assigned either zero or one label. If pixel p is assigned label f_p, then p supports the disparity range associated with label f_p with confidence. For pixels which are not assigned any label, it is because they are located in the textureless regions, and there are no visual cues detected either in the interior or on the boundaries of these regions. The background regions in Figure 1.8 are good examples of such regions.

5.4.5 Iterative Refinement

In the previous steps, K, the number of the visual cue groups (or the number of dense visual features) is a parameter set by the user. However, how to choose K is a difficult problem. If K is too large, then the large surfaces which strand many disparities will be broken into many pieces, see Figure 5.9(a). If K is too small, then we will over-group surfaces which support different ranges of disparities, see Figure 5.9(b). Therefore, in the last step of our approach, we propose an iterative method to improve our results.

The clustering step and the labeling step of our approach can be naturally iterated. The semi-dense visual correspondence generated in the previous step consists of a set of pixel blobs. Pixels in the same blob support the same range of disparities. Let $S_i = \{p_{i1}, p_{i2}, \ldots, p_{in_i}\}$ be a blob of pixels which support the disparity range $D_i = \{d_{i1}, d_{i2}, \ldots, d_{il}\}$. Then our semi-dense visual correspondence can be denoted



(a) $K = 30$ (b) $K = 10$

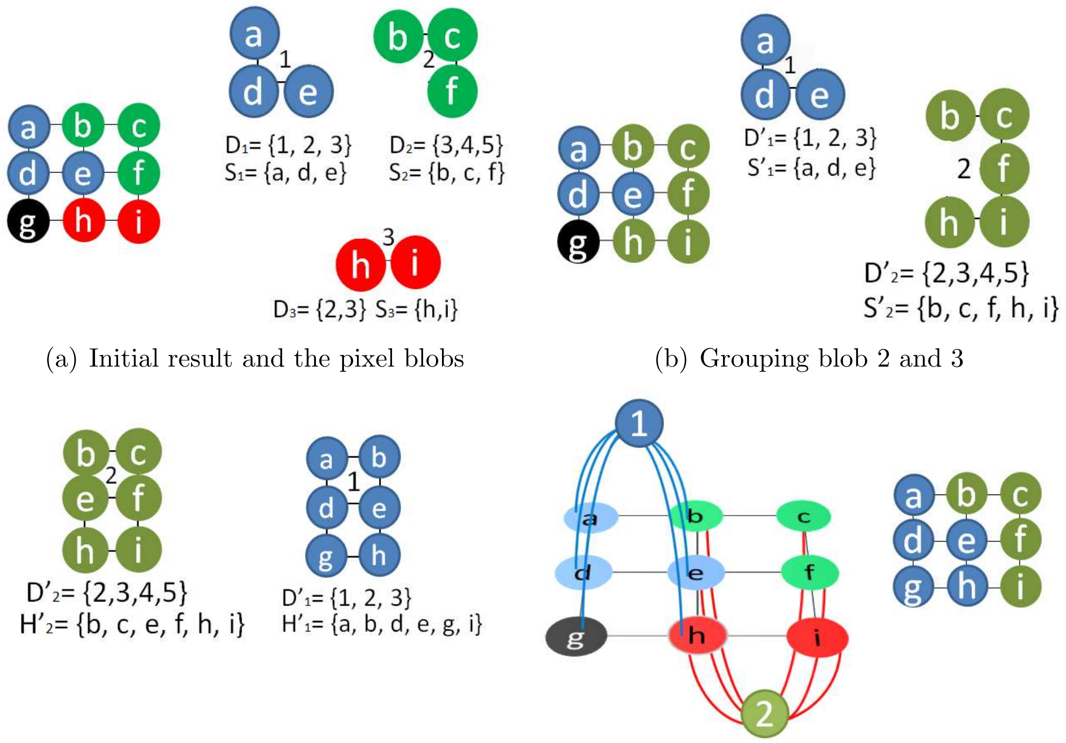
Figure 5.9: *Semi-dense correspondence with different group number K: Figure (a) is the result for $K = 30$. Notice the background is broken into several pieces. Figure (b) is the result for $K = 10$. In Figure (b), some small cones are over-grouped with the larger cones.*

as $S = \{S_1, S_2, \ldots, S_m\}$, which is a set of m pixel blobs. Figure 5.10(a) shows a simple artificial example of the refinement process.

The first step of the iteration algorithm is to cluster the blobs into K' clusters, where K' should be less then the original cluster number K. Figure 5.10(b) shows a simplified example, where the green blob and the red blob in Figure 5.10(a) are grouped togther. As mentioned in the previous paragraph, it is hard to choose an appropriate value of K. We solve this problem by letting the user pick a range of values for K, and start with the largest allowed value for K. In each iteration, we decrease the group number K, until we reach the smallest group number.

For clustering, we need to measure the distance between two pixel blobs. We define the distance between blob S_i and S_j as the following:

$$D(S_i, S_j) = D_{perimeter}(S_i, S_j) + D_{disparity}(S_i, S_j).$$



(a) Initial result and the pixel blobs

(b) Grouping blob 2 and 3

(c) Dense features and the new result

Figure 5.10: *A synthetic example for the refinement process: Figure (a) shows the initial result (before the refinement starts) and its pixel blobs. Here blob S_1 supports disparity 1 to 3. Blob S_2 supports disparity 3 to 5, and blob S_3 supports disparity 2 and 3. In Figure (b), blob S_2 and S_3 are grouped together. The new blob S_2' support disparity 2 to 5. The right part of Figure (c) shows the new dense features H_1' and H_2' generated based on the new grouping S'. The middle part of (c) shows the labeling process of the refinement. The left part of (c) is the result after refinement. The new result has two pixel blobs. The blue pixels support disparity 1 to 3, and the green pixels supports disparity 2 to 5.*

Here $D_{perimeter}$ is defined as:

$$D_{perimeter}(S_i, S_j) = 1 - max\{\frac{L_{ij}}{L_i}, \frac{L_{ij}}{L_j}\},$$

where L_i is the perimeter of blob S_i, L_j is the perimeter of blob S_j, and L_{ij} is length of the boundary between S_i and S_j. If blob S_i and S_j are adjacent to each other, and they have long common boundary between them, then $D_{perimeter}(S_i, S_j)$ is small, and S_i and S_j are more likely to be clustered together.

$D_{disparity}$ is defined as:

$$D_{disparity}(S_i, S_j) = 1 - max\{\frac{|D_i \bigcap D_j|}{|D_i|}, \frac{|D_i \bigcap D_j|}{|D_j|}\},$$

where D_i and D_j are the disparity range supported by S_i and S_j respectively. $D_{disparity}$ measures the similarity of the disparity ranges supported by S_i and S_j. If blob D_i and D_j have large intersection, then $D_{disparity}(S_i, S_j)$ is small, and S_i and S_j are more likely to be clustered together. We also use linkage cluster algorithm with Ward's method to cluster the pixel blobs in S into K' clusters, where K' is smaller than the original visual cue group number K.

Let $S'_j = S_{j1} \cup S_{j2} \cup \ldots \cup S_{jr}$ be a cluster of the pixel blobs in S. Then the disparity range associated with S'_j is $D'_j = \bigcup_{1 \le i \le r} D_{ji}$, where D_{ji} is the disparity range associated with pixel blob S_{ji}. Figure 5.10(b) shows an example of the new blob clusters. In Figure 5.10(b), blob $S_2 = \{b, c, f\}$ and $S_3 = \{h, i\}$ are grouped together, and form the new group $S'_2 == \{b, c, f, h, i\}$. The disparity range D'_2 of S'_2 is the union of $D_2 = \{3, 4, 5\}$ and $D_3 = \{2, 3\}$, that is $D'_2 = \{2, 3, 4, 5\}$.

The set of positive visual cues C'_j associated with S'_j consists of positive cues which are located inside S'_j, and whose disparity is in D'_j. That is $C'_j = \{c(x, y, d, 1)|(x, y) \in S'_j, d \in D'_j\}$. In the simple example in Figure 5.10(b), any positive visual cue located on pixel b, c, f, h and i is in C'_2, which is the set of the positive visual cue for S'_2. The set of negative visual cues N'_j for S'_j is defined as $N'_j = \bigcap_{1 \le i \le r} N_{D_{ji}}$. N'_j is the intersection of negative cues whose disparity is in D'_j.

Then for each C'_j and N'_j, we apply the binary labeling algorithm developed in Section 5.4.3. This results in blob like dense visual features $H'_1, H'_2, \ldots, H'_{K'}$ which support disparity ranges $D'_1, D'_2, \ldots, D'_{K'}$ respectively, see Figure 5.10(c) for example. Now we have a set of dense visual features $H'_1, H'_2, \ldots, H'_{K'}$, their disparity ranges $D'_1, D'_2, \ldots, D'_{K'}$, and the clusters of sparse visual cues $C'_1, C'_2, \ldots, C'_{K'}$ and $N'_1, N'_2, \ldots, N'_{K'}$. Then for pixels in $H'_1 \cup H'_2 \cup \ldots \cup H'_{K'}$, we apply the multi-labeling algorithm developed in Section 5.4.4 based on the new clusters. Figure 5.10(c) shows the labeling process for the simplified example in Figure 5.10(a).

The energy of the semi-dense visual correspondence is not guaranteed to decrease as K goes down. However, with our observation (see Figure 5.21), when the energy goes down, it is highly possible that the error rate of the semi-dense correspondence also goes down. Therefore, we keep tracking the solution with the best energy.

We start with a large number of clusters. Then we repeat the clustering and the labeling step. And in each iteration, we decrease the cluster number by one. The algorithm stops when the cluster number reaches a threshold, which is set by the user. The semi-dense visual correspondence with the smallest energy is picked as the final result. The pseudo code for our algorithm is in Algorithm 2:

5.5 Experimental Results

In this section, we present the experimental results for our semi-dense visual correspondence approach. In Section 5.5.1, we describe the set of images used for testing. In Section 5.5.2, we illustrate the sparse visual cues detected by our binary classifier. In Section 5.5.3, we present the initial semi-dense visual correspondence results, together with the discussion on their error rates. In Section 5.5.4, we show how the initial results can be refined by the iterative approach. In Section 5.5.5, we give a qualitative comparison of our results and the results of [91].

5.5.1 The Testing Set

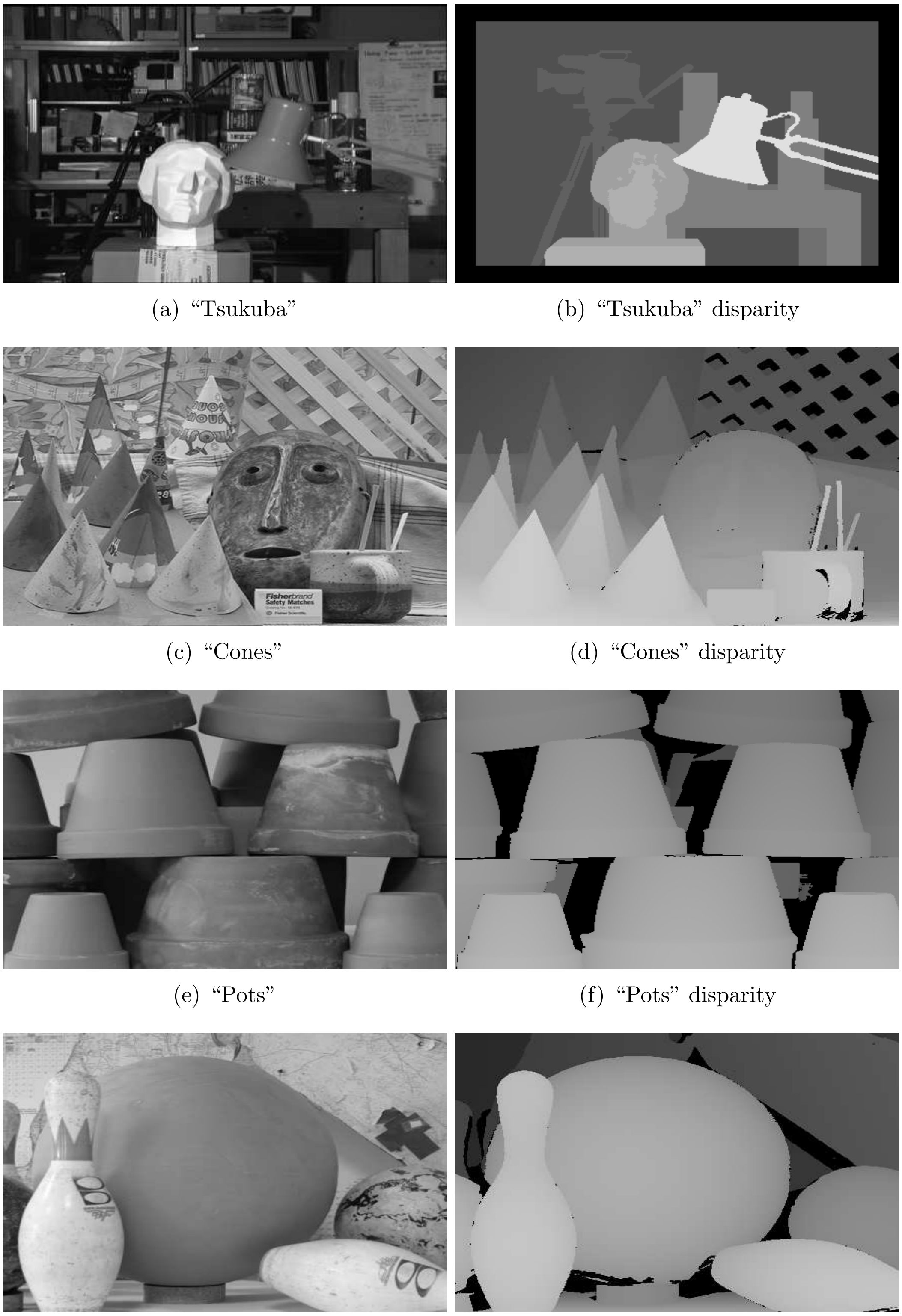
As stated in Section 5.4.1, our training set for detecting sparse visual cues consists of stereo pairs from modestly textured images to highly textured images. Therefore, our

Algorithm 2 The iterative semi-dense visual correspondence algorithm. The final group number K and the starting group number K_s are set by the user.

```
• Given a stereo image pair, apply the classifier $h(x)$ to detect the sparse positive cues $C$ and negative cues $N$.
• Cluster $C$ and $N$ into $K_s$ groups $C_1, C_2, \ldots, C_{K_s}$ and $N_1, N_2, \ldots, N_{K_s}$.
• Perform binary labeling for each $C_1, C_2, \ldots, C_{K_s}$ separately, which generates dense visual features $H_1, H_2, \ldots, H_{K_s}$.
• Apply multi-labeling for pixels in $H = \bigcup_{1 \leq i \leq K_s} H_i$. The result consists blobs of pixels $S = \{S_1, S_2, \ldots, S_m\}$ which support for disparity ranges $D_1, D_2, \ldots, D_m$ respectively.
• Set the best solution $S_{best} = S$.
• Set the best energy $E_{best} = E(S)$, where $E(S)$ is the energy of the current result $S$.
for $k = K_s - 1, K_s - 2, \ldots, K$ do
  1. Cluster pixel blob set $S = \{S_1, S_2, \ldots, S_m\}$ into $k$ clusters $S'_1, S'_2, \ldots, S'_k$.
  2. Cluster new sparse feature groups $C'_1, C'_2, \ldots, C'_k$ and $N'_1, N'_2, \ldots, N'_k$ according to $S'_1, S'_2, \ldots, S'_k$.
  3. Compute new dense visual features $H'_1, H'_2, \ldots, H'_k$ by using binary labeling according to $C'_1, C'_2, \ldots, C'_k$ and $N'_1, N'_2, \ldots, N'_k$.
  4. Apply multi-labeling on pixels in $H'_1 \cup H'_2 \cup \ldots \cup H'_k$, based on $H'_1, H'_2, \ldots, H'_k$, which results in new pixel blob set $S = \{S_1, S_2, \ldots, S_{m'}\}$.
  if $E(s) < E_{best}$ then
    $E_{best} = E(S)$.
    $S_{best} = S$.
  end if
end for
• The final semi-dense visual correspondence is $S_{best}$.
```

testing set also contains stereo images with different degree of texture, see Figure 5.11.

Here the “Tsukuba” stereo pair is highly textured, and it is widely used by many stereo approaches [76, 12, 91, 90] for evaluation purpose. The “Cones” pair contains some regions with little texture, which makes it harder to compute its disparity than the “Tsukuba” pair. The “Pots” and “Bowling” pairs contain large textureless surfaces, which are very difficult to establish their visual correspondence. All these image pairs and their disparity maps are from the Middlebury stereo data sets [76, 78, 77, 40].



(a) “Tsukuba”

(b) “Tsukuba” disparity

(c) “Cones”

(d) “Cones” disparity

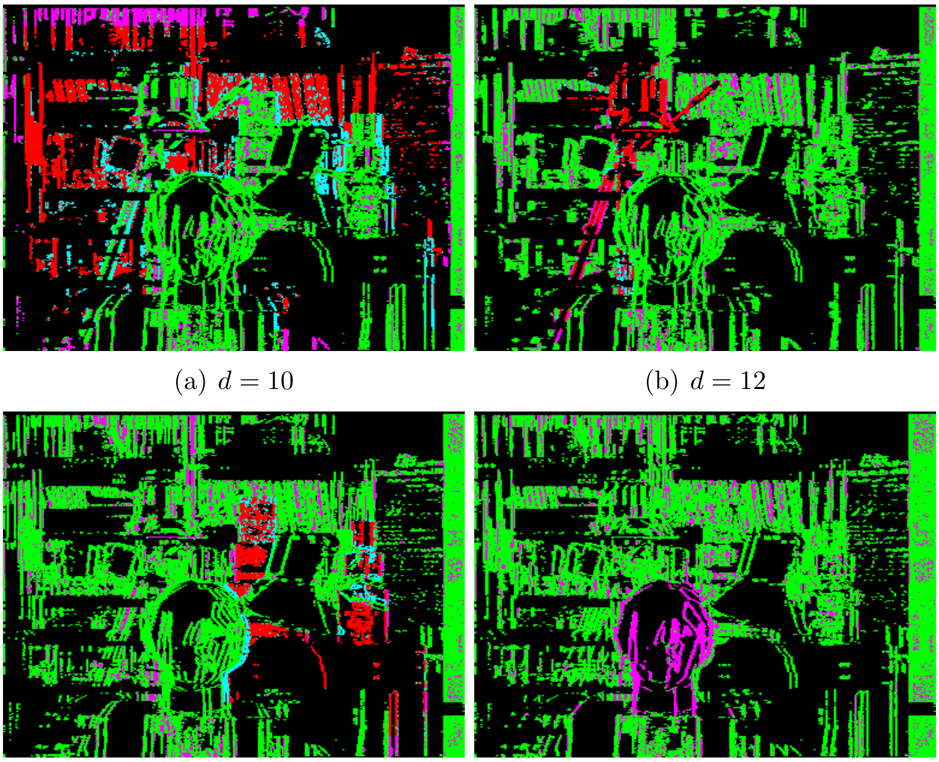
(e) “Pots”

(f) “Pots” disparity

(g) “Bowling”

(h) “Bowling” disparity

Figure 5.11: *The testing set: Figure (a), (c), (e), and (g) are the right images of the testing stereo pair. We want our testing set to cover images from highly textured images to modestly textured images. Figure (b), (d), (f), and (h) are the disparity maps of Figure (a), (c), (e), and (g).*



(a) $d = 10$

(b) $d = 12$

(c) $d = 16$

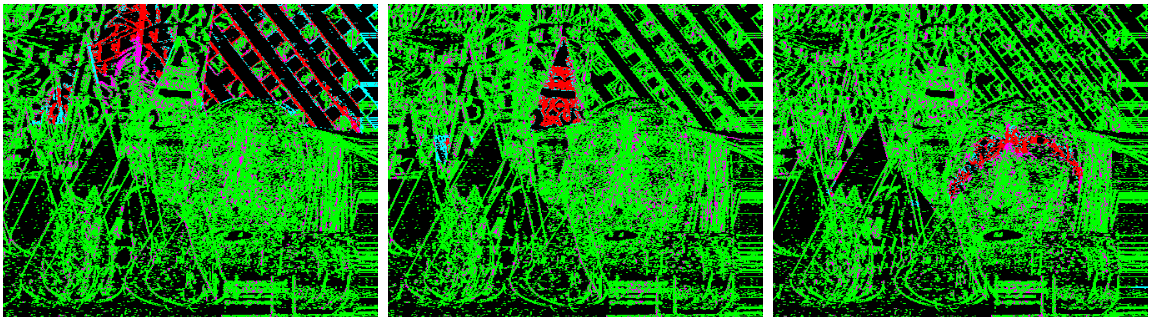
(d) $d = 21$

Figure 5.12: *The sparse visual cues for Tsukuba pair at disparity 10, 12, 16 and 21. The green pixels do not support the associated disparity d, and the true disparities for these pixels are not d either. They are classified correctly. The red pixels support the associated disparity d, and their true disparities are also d. They are also classified correctly. The purple pixels are classified as the positive visual cues which support disparity d, but their true disparities are not d. The blue pixels should be classified as the positive visual cue since their true disparities are d. But our binary classifier misclassifies them as the negative visual cues.*

5.5.2 Sparse Visual Cues

In Section 5.4.1, we developed a binary classifier which detects sparse visual cues that either support or reject a certain disparity in the textured regions. Our whole method is based on the accuracy of this binary classifier. Therefore, in this section, we present the results of detecting the sparse visual cues, and we also give an quantitative analysis about its accuracy.

Figure 5.12 shows the sparse visual cues detected at disparity 10, 12, 16, and 21 for

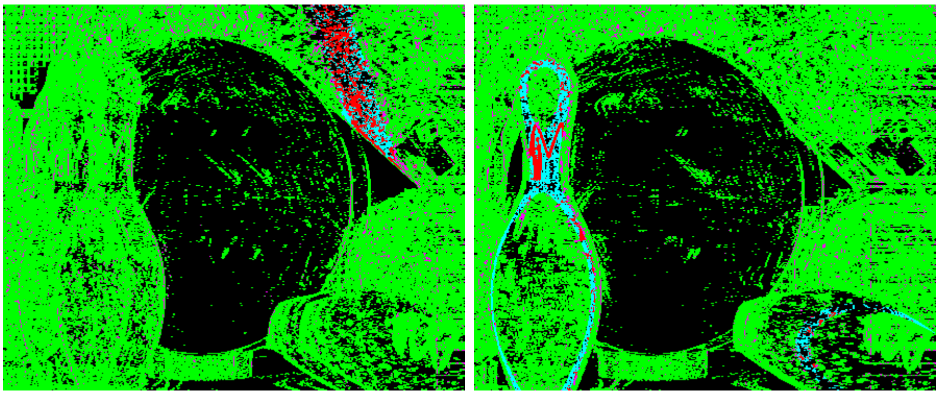


(a) $d = 21$ (b) $d = 26$ (c) $d = 33$

Figure 5.13: *The sparse visual cues for the Cones pair at disparity 21, 26, and 33.*

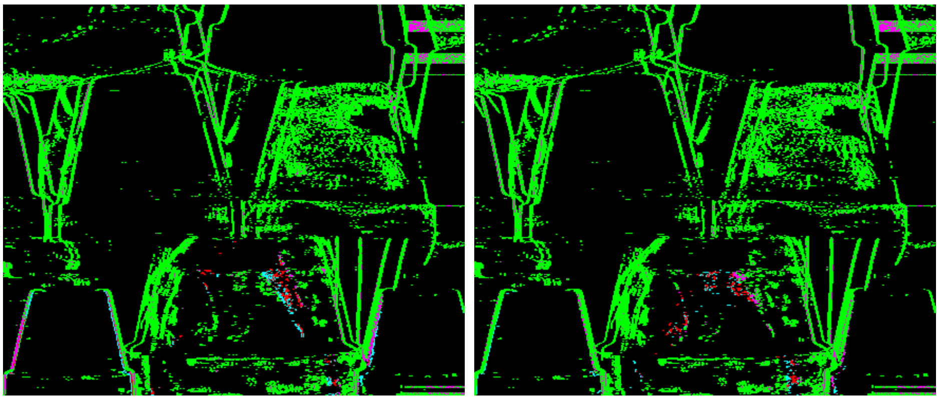
the Tsukuba stereo pair. The green pixels do not support the associated disparity d, and the true disparities for these pixels are not d either. These are the true negative pixels which are classified correctly. The red pixels support the associated disparity d, and their true disparities are also d. These are the true positive pixels. The purple pixels are the false positive pixels which are misclassified. They are classified as the positive visual cues which support disparity d, but their true disparities are not d. The blue pixels are the false negative visual cues. They should be classified as the positive visual cue since their true disparities are d. But our binary classifier misclassifies them as the negative visual cues. Notice for disparity 21, our result shows larger area of purple pixels, which are the false positive examples. But the true disparities for these purple pixels are either 20 or 22. Therefore, assigning these pixels to disparity 21 is not far from the ground truth.

Figure 5.13 is the classification result at disparity 21, 26 and 33 for the Cones pair. Figure 5.14 is the classification result at disparity 26 and 62 for the Bowling pair. Figure 5.15 shows the sparse visual cues detected at disparity 55 and 56 for the Pots pair. Notice that we only apply the binary classifier on pixels whose intensity change $e(p)$ is higher than a threshold $t = 10$. For images with large textureless regions, such as the Pots pair and the Bowling pair, the visual cues can be too sparse to be used in the next steps. Therefore, for images with little texture, we set the threshold for



(a) $d = 26$ (b) $d = 62$

Figure 5.14: *The sparse visual cues for the Bowling pair at disparity 26 and 62. Our classifier is set to be biased towards the negative cues. Therefore, in Figure (b), we have large false negative rates, shown with blue pixels.*



(a) $d = 55$ (b) $d = 56$

Figure 5.15: *The sparse visual cues for the Pots pair at disparity 55 and 56.*

the intensity change to be $t = 5$ to get denser results.

Table 5.1 shows the error rate of the binary classifier. Here the total error rate $Error$ is defined as $Error = \frac{|P_m|}{|P|}$, where P_m is the set of misclassified visual cues, and P is the set of all sparse visual cues. The false positive rate $Error_p$ is computed as $Error_p = \frac{|P_{mp}|}{|P_n|}$. Here P_{mp} is the set of pixels labeled as positive cues which are actually negative cues, and P_n is the set of true negative cues.

The false negative rate $Error_n$ is defined similarly as E_p. That is $Error_n = \frac{|P_{mn}|}{|P_p|}$,

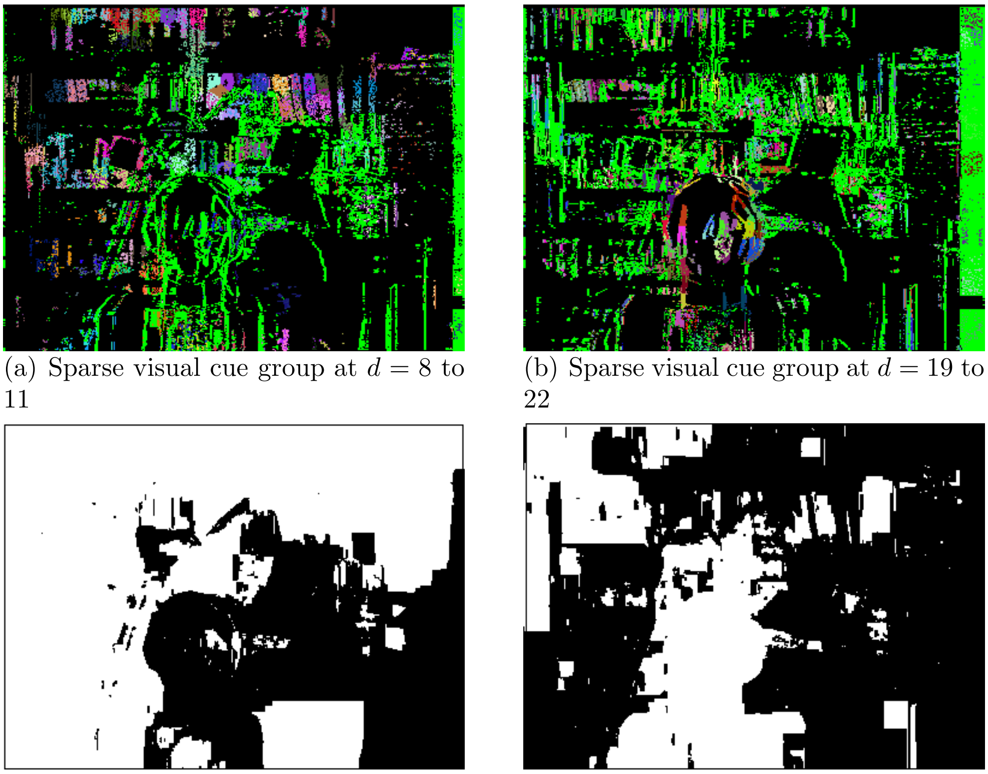
Image	$Error$	$Error_p$	$Error_n$
Tsukuba	14.84%	14.14%	10.82%
Cones	10.34%	9.84%	48.83%
Bowling	5.40%	4.65%	58.81%
Pots	4.97%	4.13%	48.50%

Table 5.1: *The error rates of detecting sparse visual cues. We have large false negative rates since we set our binary classifier to bias towards the negative visual cues.*

where P_{mn} is the set of false negative cues, and P_p is the true positive cues. The false negative error rate for our classifier is high, since we set our binary classifier to be biased towards the negative cues. Having a high false negative error is not as harmful for our algorithm as having a high false positive error. This is because we can still segment out the pixels that should belong some disparity but were missed at the step when we get dense visual cues. If the false positive rate is high, we have a high chance of producing dense visual features that are actually wrong, for example, do not match at the associated range of disparities. Therefore we bias our classifier (by changing the learnt thresholds) towards a lower false positive error and still have positive cue groups which are dense enough to be used in the next steps, see Figure 5.16.

5.5.3 The Initial Semi-dense Visual Correspondence

In this work, we aim at semi-dense visual correspondences for which we are more confident in their associated ranges of disparities. Therefore, after detecting the sparse visual cues, we need to group these visual cues into larger groups according to their locations and disparities. Figure 5.16(a) and (b) show two visual cue groups for the Tsukuba stereo pair. The green pixels belong to the negative cue cluster. Pixels with colours other than green are the positive visual cues. Positive visual cues which belong to the same objects are mostly clustered together. The false positive cues are also grouped together, which makes them denser and harder to eliminate. However,



(a) Sparse visual cue group at $d = 8$ to 11

(b) Sparse visual cue group at $d = 19$ to 22

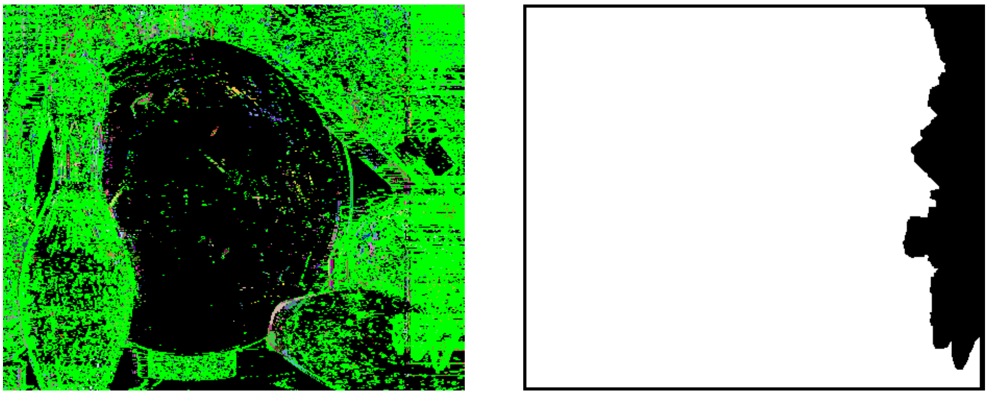
(c) Dense visual features at $d = 8$ to 11

(d) Dense visual features at $d = 19$ to 22

Figure 5.16: *The sparse visual cues groups and dense visual features for the Tsukuba pair. Figure (a) and (b) are the visual cue groups for the Tsukuba pair at disparity ranges 8 to 11 and 19 to 22 respectively. These are exactly the disparity ranges for the background and the sculpture. Figure (c) and (d) are the dense visual features generated with the binary labeling algorithm in Section 5.4.3. The white pixels belong to the dense visual features that support the associated disparity ranges.*

our visual cue groups also include negative visual features, which strongly disagree with these false positive visual cues. Therefore, these false positive cues will not affect the final result too much, see Figure 5.16(c) and (d).

Figure 5.17(a) shows a visual cue group for the Bowling pair. Notice the positive visual cues on the big bowling ball in the middle are grouped together. Hence they are dense enough to be propagated into other part of the image in the next step, which generates a dense visual feature, see Figure 5.17(b). The sparse positive visual cues detected on the boundary and inside the big bowling ball are propagated into



(a) Sparse visual cue group at $d = 49$ to 59

(b) Dense visual features at $d = 49$ to 59

Figure 5.17: *The sparse visual cues group and dense visual feature for the Bowling pair. Figure (a) is the visual cue group for the Bowling pair at disparity range 49 to 59. This is the disparity range of the big bowling ball in the middle. Figure (b) is the dense visual feature generated based on the visual cue group in Figure (a).*

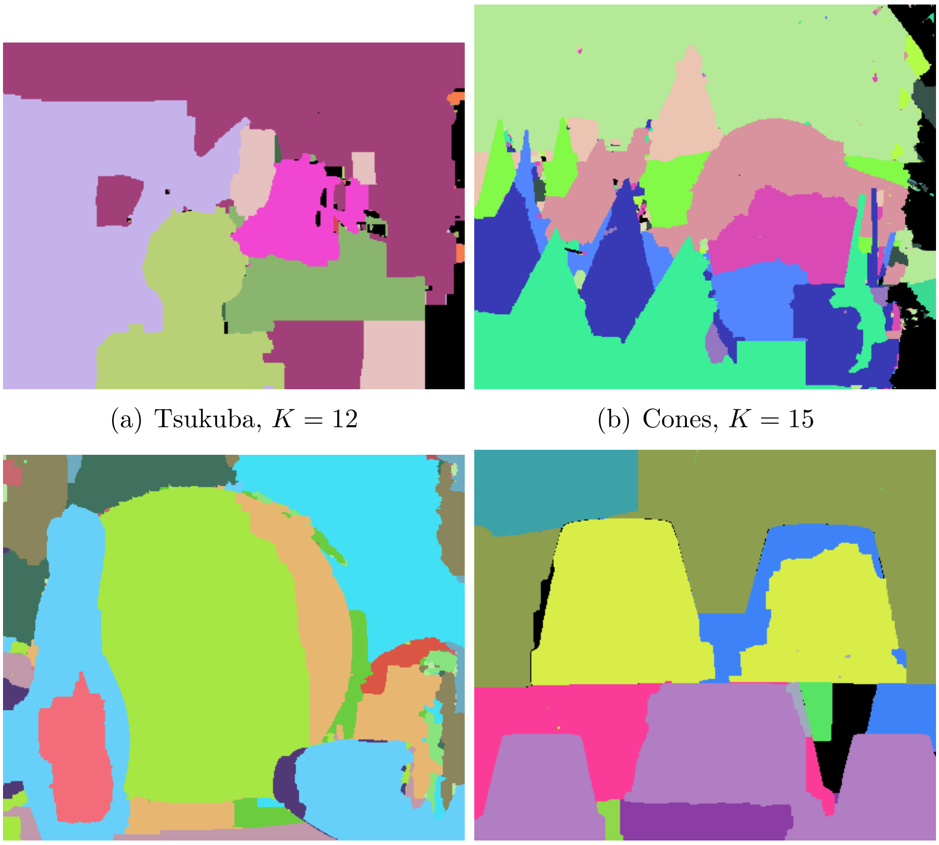
the other parts of the bowling ball. Thus the disparity range of the bowling ball is recovered.

Figure 5.18 shows the initial semi-dense visual correspondence results for the testing images in Figure 5.11. To evaluate the accuracy of our semi-dense visual correspondence, we formulate the following equation:

$$Error(S) = \frac{|\{p|p \in \mathcal{P}_S, T(p) \notin D_S(p)\}|}{|\{p|p \in \mathcal{P}_S\}|},$$

where S is the semi-dense visual correspondence, $\mathcal{P}_S$ is the set of pixels which are covered by S, $D_S(p)$ is the disparity range assigned to pixel p by S, and $T(p)$ is the true disparity of pixel p. Then the error rate of S is the ratio of the misclassified pixel number over the total number of pixels which are covered by the semi-dense visual correspondence. We also measure the coverage of S by $Coverage(S) = \frac{|\mathcal{P}_S|}{|\mathcal{P}|}$, where $\mathcal{P}$ is the set of all pixels in the right image. Table 5.2 shows the error rates and coverage of the initial semi-dense visual correspondence for the testing images in Figure 5.11.

Figure 5.19 shows the error rates of the initial results under different group number



(a) Tsukuba, $K = 12$

(b) Cones, $K = 15$

(c) Bowling, $K = 17$

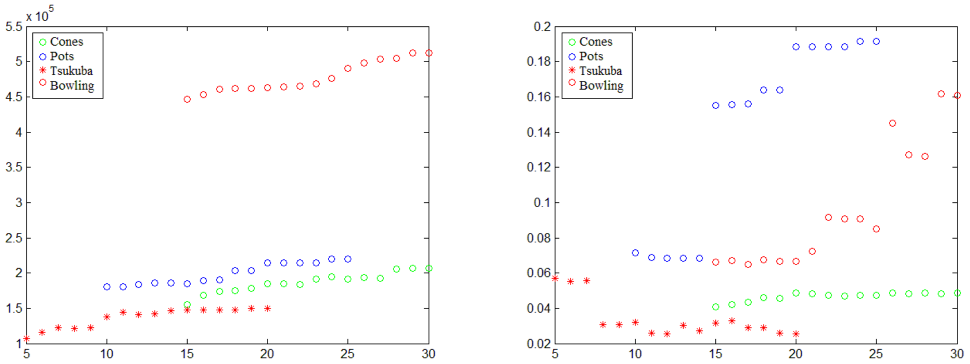
(d) Pots, $K = 12$

Figure 5.18: *The initial semi-dense visual correspondence for the images in Figure 5.11. Pixels with the same colour support the same range of disparities. Black pixels are not assigned any disparity range, since there is no sparse visual cue detected in these regions or on the boundaries of these regions.*

K. Generally, for the Bowling pair, Pots pair and Cones pair, the smaller is K, the lower is the error rate and the energy. However, for the Tsukuba pair, it is reversed: the greater is K, the lower is the error rate, but the higher is the energy. Therefore, we can not conclude on any relationship between the error rate, the energy and the group number.

Image	K	$Error(S)$	$Coverage(S)$
Tsukuba	12	2.56%	98.22%
Cones	15	4.06%	95.24%
Bowling	17	6.48%	100%
Pots	12	6.83%	97.65%

Table 5.2: *The error rates and coverage of the initial semi-dense visual correspondence.*



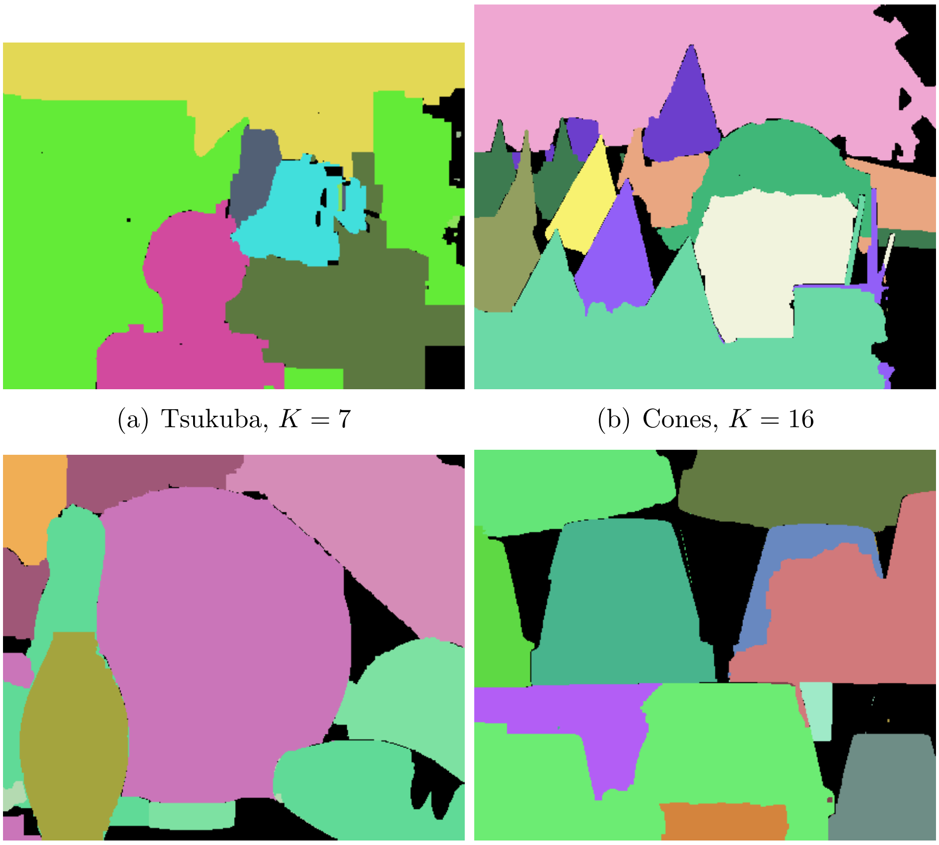
(a) Energy by K

(b) Error by K

Figure 5.19: *The error rate and energy of the initial semi-dense visual correspondence. The horizontal axe is the group number K. The vertical axe in Figure (a) is the energy. The vertical axe in Figure (b) is the error rate $Error(s)$. Generally, for the Bowling pair, Pots pair and Cones pair, the smaller is K, the lower is the error rate and the energy. However, for the Tsukuba pair, it is reversed: the greater is K, the lower is the error rate, but the higher is the energy.*

5.5.4 Refined Semi-dense Visual Correspondence

In Section 5.4.5, we developed an iterative refinement method to improve our initial semi-dense visual correspondence. It first groups the pixel blobs which support similar disparity ranges and are located next to each other together into K' groups, where K' is smaller than the original group number. The refinement process requires the user to input a large initial group number and a small final group number. In each iteration, pixel blobs of the current solution will be grouped to form new clusters. And the group number is decreased by one. New visual correspondence is then computed based on the new visual cue groups. The refinement process stops when the group



(a) Tsukuba, $K = 7$

(b) Cones, $K = 16$

(c) Bowling, $K = 15$

(d) Pots, $K = 16$

Figure 5.20: *The refined semi-dense visual correspondence for the images in Figure 5.11. Pixels with the same colour support the same range of disparities. Black pixels are not assigned any disparity range.*

number reaches the final group number. The visual correspondence with the lowest energy is selected as the final result. By using this refinement process, we avoid problem of picking the group number K by hand.

Figure 5.20 shows the refined semi-dense visual correspondence for images in Figure 5.11. Compared with the initial results shown in Figure 5.18, the boundaries of the pixel blobs in Figure 5.20 are much clearer, especially for images with large textureless regions, such as the Bowling pair and the Pots pair. Table 5.3 shows the error rates and the coverage of the refined results in Figure 5.20. The error rates are lower than that of the initial results, shown in Table 5.2. The coverage of the refined

Image	K	$Error(S)$	$Coverage(S)$
Tsukuba	7	1.20%	98.45%
Cones	16	0.70%	90.24%
Bowling	15	1.26%	93.09%
Pots	16	5.82%	88.62%

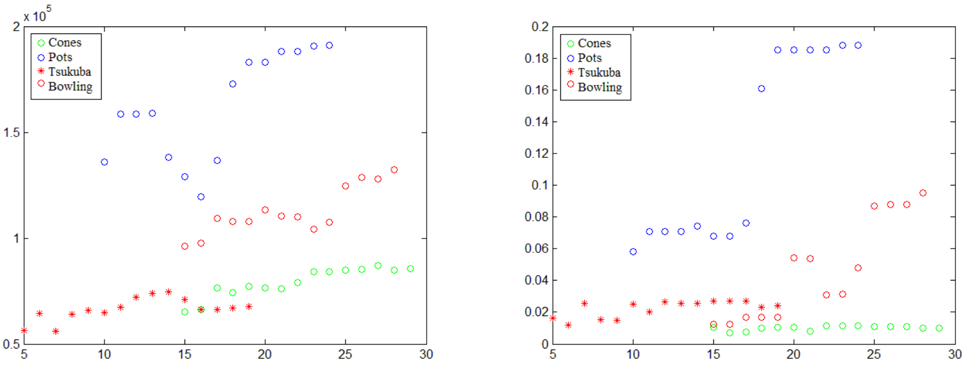
Table 5.3: *The error rates and coverage of the refined semi-dense visual correspondence.*

results are lower than the initial result, since the refinement process eliminates lots of the false positive visual cues. Then the sparse visual cues found on the boundaries or in the interior of the textureless regions are much less than that of the initial results.

Figure 5.21 shows the relationship between the group number K, the energy E and the error rates $Error$. For the refined results, we observed that when the energy decreases, the error rate of the semi-dense visual correspondence also mostly decreases. Intuitively, this is because in the refinement process, we only group together the positive cues within the pixel blobs. The false positive cues which are usually located out of the pixel blobs are eliminated by the process, which results in lower energy and higher accuracy.

5.5.5 Qualitative Comparison

In this section, we give a qualitative comparison between our results and the results of Veksler [91]. Our algorithm recovers the disparity ranges of large textureless regions that strand several disparities, see Figure 5.20. Unlike the previous methods [90, 91], which only allow each dense visual feature to have one disparity, our dense visual features have disparity ranges that vary smoothly. Figure 5.22 shows the results for Tsukuba, Cones, Bowling, and Pots pairs generated by the method in [91]. The error rates for these results are 5.28%, 4.68%, 2.71% and 1.32% respectively. And their coverage is 79%, 69%, 16% and 10%. When there are large textureless regions in the image, their results can be very sparse, compared with our results in Figure 5.20.



(a) Energy by K

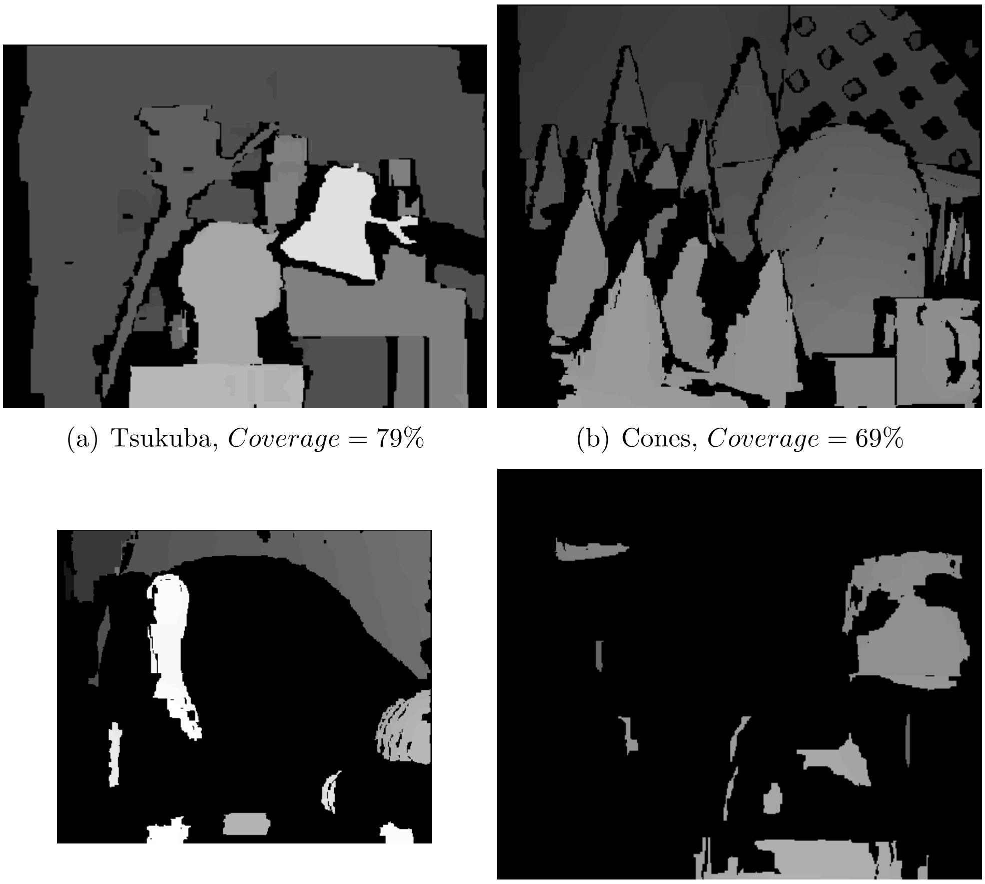
(b) Error by K

Figure 5.21: *The error rate and the energy of the refined semi-dense visual correspondence with different group number K. The horizontal axe is the group number K. The vertical axe in (a) is the energy for the multi-labeling framework developed in Section 5.4.4. The vertical axe in (b) is the error rate $Error(S)$. When refining the results, as K decreases, the energy and the error rates do not always decrease. However, from these examples, we can see that when the energy is small, it is highly possible the error rate is small.*

Figure 5.23 shows another example which is extremely textureless in both the background regions and the object regions, see Figure 5.23(a). The method in [91] detects the disparities at the textured objects very well, however, the disparities of the background and the table are missing in their result, as shown in Figure 5.23(c). Figure 5.23(b) shows our result. Our algorithm made gross error in the blue blobs, where pixels that should be assigned the disparity range of the background are assigned the disparities of the bleach bottle. However, we recovered the disparity ranges of the background and parts of the table.

5.6 Summary

In this chapter, we developed a method that recovers the disparity ranges of the large textureless regions for stereo images. Compared with the previous methods [90, 91], which only allow a dense visual feature to have one disparity, our method handles the textureless regions that strand many disparities with confidence.



(a) Tsukuba, $Coverage = 79\%$

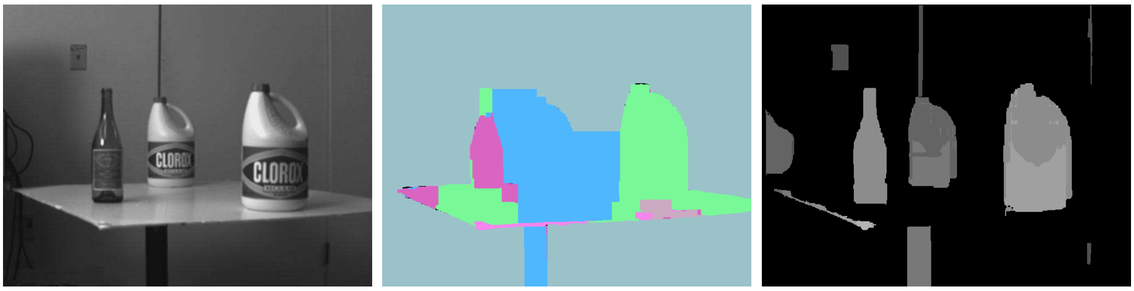
(b) Cones, $Coverage = 69\%$

(c) Bowling, $Coverage = 26\%$

(d) Pots, $Coverage = 10\%$

Figure 5.22: *The results of Veksler [91]: Figure (a) to (d) show the results for Tsukuba, Cones, Bowling, and Pots. Their coverage are 79%, 69%,26%, and 10% respectively. The method in [91] works well in textured images, such as the Tsukuba pair and the Cones pair. However, for images with large textured regions, such as the Pots and the Bowling pair, the coverage of this method is low, compared with our results in Figure 5.20.*

Our approach is based on a binary classifier which distinguishes the pixels that support or reject a particular disparity in the textured regions. By developing such a classifier, we avoided the problem of picking a good threshold value for the boundary condition feature as mentioned in Section 5.4.1. In the training stage of the binary classifier, we also discovered many useful features for stereo other than the basic intensity difference and boundary condition. Grouping together these sparse visual cues detected by the binary classifier generates larger visual cue groups that have smoothly



(a) Input image (b) Our result (c) Result of Veksler [91]

Figure 5.23: *Comparison between our results and the results of Veksler [91]: Figure (a) shows the input image which is highly textureless. Figure (b) shows our result. The background is assigned the disparity range from 4 to 5. The big bleach bottle on the right (the green blob) is assigned disparity range from 7 to 8. The wine bottle (pink blob) on the left is assigned disparity range from 6 to 7. The bleach bottle in the middle (the blue blob) is assigned disparity range from 5 to 6. Our algorithm made gross error in the blue blobs, where pixels that should be assigned the disparity range of the background are assigned the disparities of the bleach bottle. Figure (c) shows the result generated by the method in Veksler [91]. The disparities of the two bleach bottles and the wine bottle are recovered. However, the disparities of the table and the background are missing in Figure (c), as shown in black colour.*

varying disparities. This enables us to propagate these visual cues into textureless regions that strand several disparities. The blob-like, dense visual features generated in this step recover the disparity range of the textureless regions with confidence. A multi-labeling framework is formulated to resolve the ambiguities between the dense features. The result of this step is a semi-dense visual correspondence which assigns either one or zero disparity range to every pixel in the image.

The problem with these initial results is that the false positive cues detected by the binary classifier in the first step of our approach are also grouped together in the grouping stage, and this makes the false positive visual cues denser and harder to eliminate. Therefore, we came up with an iterative refinement approach to improve the initial results. During the refinement, false positive cues are mostly eliminated from the sparse visual cue group. This results in higher accuracy in the final semi-dense visual correspondence.

Moreover, our iterative refinement approach provides us a useful method to select the value of the group number parameter. By eliminating the false positive visual cues, we established the relationship between the energy of the multi-labeling problem and the error rate of the resulting semi-dense visual correspondence. By observing the results, we can see that the lower is the energy, the higher tends to be the error rate. Then by tracking the result with lowest energy, we avoid the problem of selecting a good group number. Instead of choosing one group number, we can provide a range of values for the group number, and our approach selects the group number with lowest energy.

The main limitation of our approach is that it may fail if the textureless region is too large and strands too many disparities, where the disparities in the center of this region are far way from the disparities on the boundary. That is because no visual cues supporting the disparities in the center of this region can be detected, and the visual cues on the boundary of this region support disparities which are quite far way from that of the center. Therefore, even if a disparity range is assigned to the center pixels of this huge textureless region, this disparity range may not contain the true disparities of the center pixels.

Chapter 6

Conclusion and Future Work

Traditionally, computer vision and graphics were two distinct fields which focus on opposite topics. Computer vision infers information from images to “understand” a scene, and computer graphics manipulates the content of the images to synthesize the real world. With the growing demand on the new techniques such as 3D television and virtual reality, computer graphics and vision now need to address new issues. These issues include displaying the results in high quality and modeling user interaction with the environment naturally. To address these new issues, graphics and vision are increasingly borrowing ideas from each other. Therefore, we need to investigate the computer vision and graphics problems in a new way which integrates these two fields.

In this thesis, we started with a graphics problem: rendering static and animated mosaics from real images and video. In Chapter 3, we first restated our approach for rendering static mosaics from real images, which was the foundation of our animated mosaic approach. We formulated the problem of classic mosaic generation in a global optimization framework, and designed an energy function encoding the properties that lead to visually pleasing mosaics. Our global optimization framework offered a more principled approach than the previous work, which was mostly based on heuristics.

The desired mosaic properties were directly modeled into an energy function, instead of devising a sequence of heuristics steps that may possibly lead to the desired result. Another advantage was that the value of the energy function itself could be an effective measure to assess the quality of a mosaic. Compared with the preliminary version of our static mosaic rendering method in [64], we formulated our energy function in a new way so that all the constraints for a good classic mosaic (edge alignment, edge avoidance, tight tile packing) were encoded in one energy function. The restating of the mosaic rendering process made it easier to comprehend the problem, since the relationship between all the mosaic constraints was revealed as a whole energy function.

Next, we extended our approach to rendering animations with mosaic effects in Chapter 4. The animations generated by our approach are composed of hundreds of colourful square tiles, which are arranged to present the shape and colour of the objects in the video, moving in a timely coherent manner. Each frame of the resulting animation is a classic mosaic image composed of a large number of square tiles, which are aligned to the strong edges in the input scene. Between the consecutive frames, the tiles are moved according to the motion of their center pixels. Therefore, the whole animation has a consistent motion effect. Our method estimates the motion of the pixels in the video, renders the frames with mosaic effect based on both the colour and motion information from the input video. Our mosaic animation style is uniquely expressive, part of its appeal steams from the fact that mosaics is an ancient art form. The algorithm relies extensively on our novel motion segmentation algorithm. The state of the art in motion segmentation is such that user interaction is still required to get convincing results. We produced temporally coherent and visually appealing animations.

The biggest limitation for our animated mosaic approach is that the motion models are restricted to rigid motion. To improve the quality of our animated mosaics, we need to detect the motion between the frames of a given video sequence more accurately. To simulate the deformation of the objects in the input video, we need to extend our motion models to be more general. Although our original goal was to find a more robust way to detect motion in a video, stereo problems have a similar setting with motion but are easier to start with. Therefore, in Chapter 5, we proposed an approach which found semi-dense visual correspondences for which we could estimate the range of disparities with a high confidence.

The idea behind our approach is that the visual cues that are reliable for establishing correspondence are usually located at pixels with high texture. If there is a region in the image that is surrounded by texture cues, its range of disparities can be estimated more reliably. Even if the region is textureless inside, we can propagate the visual cues from the texture cues at the boundaries.

We started with selecting useful features that had discriminant power for stereo. This resulted in a binary classifier which distinguishes the pixels that support or reject a particular disparity in the textured regions. Then by grouping together these sparse visual cues detected by the binary classifier, we generated larger groups of visual cues that had smoothly varying disparities. We then propagated the visual cues in these groups into textureless regions that strand several disparities, by formulating this problem into a binary labeling framework. The blob-like, dense visual features generated in this step recovered the disparity range of the textureless regions with confidence. The ambiguities of between the dense features were then resolved by a multi-labeling framework. The semi-dense visual correspondence generated in this step assigned either one or zero disparity range to every pixel in the image.

Next, we came up with an iterative refinement approach to improve the initial results. We aimed at eliminating the false positive visual cues during the refinement process, which resulted in clearer boundaries between the pixel blobs in the final semi-dense visual correspondence. The error rates of the refined results were also reduced compared with the initial results. More importantly, by removing the false positive visual cues, we also established a correlation between the energy of the multi-labeling problem and the error rate of the resulting semi-dense visual correspondence. Thus we avoided the problem of choosing a good value for the group number. Unlike the previous methods [90, 91], which only allow a dense visual feature to have one disparity, our method recovered the disparity range of the textureless regions that strand many disparities with confidence.

Our approach may fail if the textureless region is too large and strands too many disparities, where the disparities for the pixels in the center of this region are far way from the disparities of the boundary pixels. For such a huge textureless region, no visual cues supporting the disparities of the center pixels can be detected inside the region, and the visual cues on the boundary of this region support disparities which are quite far way from that of the center. Therefore, even a disparity range is assigned to the center pixels of this huge textureless region, this disparity range may not contain the true disparities of the center pixels. This is the main limitation of our approach.

In the future, we plan to extend our semi-dense visual correspondence approach to motion estimation. In this case, we need to overcome the difficulties brought by motion blur, in addition to all the difficulties of stereo. We also plan to apply our visual correspondence approach to improve the quality of our mosaic animations. Thus we can extend our motion model to more flexible models other than the rigid motions. Occlusion reasoning is another issue we need to address in the future.

Bibliography

[1] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graph.*, 23:294–302, August 2004.

[2] Marc Alexa. Extracting the essence from sets of images. In Douglas W. Cunningham, Gary W. Meyer, Laszlo Neumann, Alan Dunning, and Raquel Paricio, editors, *Computational Aesthetics 2007: Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging, Banff, Alberta, Canada, June 20-22, 2007*, pages 113–120. Eurographics Association, 2007.

[3] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2:283–310, 1989. 10.1007/BF00158167.

[4] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *Int. J. Comput. Vision*, 12(1):43–77, 1994.

[5] S. Battiato, G. Di Blasi, G. Gallo, G.C. Guarnera, and Puglisi G. Artificial mosaics by gradient vector flow. In *Proceedings of EuroGraphics 2008*, 2008.

[6] Sebastiano Battiato, Gianpiero Di Blasi, Giovanni Maria Farinella, and Giovanni Gallo. A Novel Technique for Opus Vermiculatum Mosaic Rendering. In Joaquim Jorge and Vaclav Skala, editors, *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and*

Computer Vision (WSCG 2006, February, 2006, Plzen, Czech Republic), pages 133–140, Plzen, 2006. University of West Bohemia, UNION Agency.

[7] F. Bignone, O. Henricsson, P. Fua, and M. A. Stricker. Automatic extraction of generic house roofs from high resolution aerial imagery. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, pages 85–96, London, UK, 1996. Springer-Verlag.

[8] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(4):401–406, 1998.

[9] Volker Blanz, Patrick Grother, P. Jonathon Phillips, and Thomas Vetter. Face recognition based on frontal views generated from non-frontal images. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 454–461, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Gianpiero Di Blasi and Giovanni Gallo. Artificial mosaics. *The Visual Computer*, 21:373–383, 2005.

[11] Y. Boykov and O. Veksler. *Graph Cuts in Vision and Graphics: Theory and Applications, in Mathematical Models of Computer Vision: The Handbook.* Springer-Verlag, 2006.

[12] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[13] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:1124–1137, September 2004.

[14] M. Z. Brown, D. Burschka, and G. D. Hager. Advances in computational stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(8):993–1008, 2003.

[15] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[16] Ingemar J. Cox, Sunita L. Hingorani, Satish B. Rao, and Bruce M. Maggs. A maximum likelihood stereo algorithm. *Comput. Vis. Image Underst.*, 63(3):542–567, 1996.

[17] F.C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Computer Graphics*, 18(3):207–212, 1984.

[18] K. Dalal, A. W. Klein, Y. Liu, and K. Smith. A spectral approach to npr packing. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 71–78, New York, NY, USA, 2006. ACM.

[19] T.J. Darrell and A.P. Pentland. Cooperative robust estimation using layers of support. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):474–487, May 1995.

[20] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 11–20, New York, NY, USA, 1996. ACM.

[21] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19:40–51, 2000.

[22] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification, 2nd Edition*. Wiley-Blackwell, 2001.

[23] G. Elber and G. Wolberg. Rendering traditional mosaics. *The Visual Computer*, 19:67–78, 2003.

[24] Gershon Elber. Line art rendering via a coverage of isoparametric curves. *IEEE Transactions on Visualization and Computer Graphics*, 1:231–239, 1995.

[25] Gershon Elber. Line art illustrations of parametric and implicit forms. *IEEE Transactions on Visualization and Computer Graphics*, 4:71–81, January 1998.

[26] A. M. Elgammal, D. Harwood, and L. S. Davis. Non-parametric model for background subtraction. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*, pages 751–767, London, UK, 2000. Springer-Verlag.

[27] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004.

[28] P.F. Felzenszwalb and O. Veksler. Tiered scene labeling with dynamic programming. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3097–3104, June 2010.

[29] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. pages 726–740, 1987.

[30] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

[31] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 136–146, New York, NY, USA, 1986. ACM.

[32] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society Series B Methodological*, 51(2):271–279, 1989.

[33] W. Eric L. Grimson and Theodosios Pavlidis. Discontinuity detection for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 30:316–330, 1985.

[34] P. L. Hammer, P. Hansen, and B. Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematicl Programming*, 28(28):121–155, 1984.

[35] M. J. Hannah. *Computer matching of areas in stereo images.* PhD thesis, Stanford, CA, USA, 1974.

[36] A. Hausner. Simulating decorative mosaics. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 573–580, New York, NY, USA, 2001. ACM.

[37] James Hays and Irfan A. Essa. Image and video based painterly animation. In *NPAR '04: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, pages 113–120, 2004.

[38] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 7–12, New York, NY, USA, 2000. ACM.

[39] Aaron Hertzmann. Paint by relaxation. In *Computer Graphics International 2001*, CGI '01, pages 47–54, Washington, DC, USA, 2001. IEEE Computer Society.

[40] Heiko Hirschmller and Daniel Scharstein. Evaluation of cost functions for stereo matching. In *Computer Vision and Pattern Recognition*, 2007.

[41] Heiko Hirschmuller and Daniel Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31:1582–1599, September 2009.

[42] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *ARTIFICAL INTELLIGENCE*, 17:185–203, 1981.

[43] Stephen S. Intille and Aaron F. Bobick. Incorporating intensity edges in the recovery of occlusion regions. In *International Conference on Pattern Recognition*, pages 674–677, 1994.

[44] H. Ishikawa. Exact optimization for markov random fields with convex priors. *IEEE Transacions on Pattern Analysis and Machine Intellegence*, 25(10):1333–1336, 2003.

[45] Nianjuan Jiang, Ping Tan, and Loong-Fah Cheong. Symmetric architecture modeling with a single image. *ACM Trans. Graph.*, 28:113:1–113:8, December 2009.

[46] Schindler K. Spatially consistent 3d motion segmentation. In *IEEE International Conference on Image Processing*, volume 3, pages 409–412, September 2005.

[47] T. Kanade, H. Kano, S. Kimura, A. Yoshida, and K. Oda. Development of a video-rate stereo machine. *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, 3:3095, 1995.

[48] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 1(4):321–331, 1988.

[49] J. Kim, V. Kolmogorov, and R. Zabih. Visual correspondence using energy minimization and mutual information. In *Proceedings of IEEE International Conference on Computer Vision*, volume 2, pages 1033–1040, 2003.

[50] A. W. Klein, P. J. Sloan, A. Finkelstein, and M. F. Cohen. Stylized video cubes. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 15–22, New York, NY, USA, 2002. ACM.

[51] Allison W. Klein, Tyler Grant, Adam Finkelstein, and Michael F. Cohen. Video mosaics. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 21–ff, 2002.

[52] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusion via graph cuts. In *Proceedings of IEEE International Conference on Computer Vision*, pages 508–515, 2001.

[53] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part III*, pages 65–81, London, UK, 2002. Springer-Verlag.

[54] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John F. Hughes. Art-based rendering of fur, grass, and trees. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 433–438, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[55] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, 2003.

[56] Jan Eric Kyprianidis, Henry Kang, and Jürgen Döllner. Image and video abstraction by anisotropic kuwahara filtering. *Computer Graphics Forum*, 28(7):1955–1963, 2009. Special issue on Pacific Graphics 2009.

[57] David Lee and Theodosios Pavlidis. One-dimensional regularization with discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10:822–829, 1988.

[58] V. Lempitsky, S. Roth, and C. Rother. Fusionflow: Discrete-continuous optimization for optical flow estimation. In *Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'08)*, pages 1–8, 2008.

[59] P. Litwinowicz. Processing images and video for an impressionist effect. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 407–414, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[60] Peter Litwinowicz. Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 407–414, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[61] Ce Liu, Antonio Torralba, William T. Freeman, Frédo Durand, and Edward H. Adelson. Motion magnification. *ACM Trans. Graph.*, 24:519–526, July 2005.

[62] Y. Liu and O. Veksler. Animated classic mosaics from video. In *ISVC '09: Proceedings of the 5th International Symposium on Advances in Visual Computing*, pages 1085–1096, Berlin, Heidelberg, 2009. Springer-Verlag.

[63] Y. Liu, O. Veksler, and O. Juan. Simulating classic mosaics with graph cuts. In *EMMCVPR'07: Proceedings of the 6th international conference on Energy minimization methods in computer vision and pattern recognition*, pages 55–70, Berlin, Heidelberg, 2007. Springer-Verlag.

[64] Yu Liu, Olga Veksler, and Olivier Juan. Generating classic mosaics with graph cuts. *Computer Graphics Forum*, 29:2387–2399, 2010.

[65] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81*, pages 674–679, 1981.

[66] D. Marr and T. Poggio. *Cooperative computation of stereo disparity*, pages 259–267. MIT Press, Cambridge, MA, USA, 1988.

[67] L. Matthies, T. Kanade, and R. Szeliski. Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, 3:209–238, 1989. 10.1007/BF00133032.

[68] Barbara J. Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 477–484, New York, NY, USA, 1996. ACM.

[69] Yuichi Ohta and Takeo Kanade. Stereo by two-level dynamic programming. In *IJCAI'85: Proceedings of the 9th international joint conference on Artificial intelligence*, pages 1120–1126, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[70] Björn Ommer and Joachim M. Buhmann. Learning compositional categorization models. In *European Conference on Computer Vision*, pages 316–329, 2006.

[71] Oscar Meruvia Pastor, Bert Freudenberg, and Thomas Strothotte. Real-time animated stippling. *IEEE Comput. Graph. Appl.*, 23:62–68, July 2003.

[72] Frederic Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H. Salesin. Synthesizing realistic facial expressions from photographs. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[73] Kirkparick S., Gelatt C. D. Jr., and Vecchi M. P. Optimization with simulated annealing. *Science*, 220:671–680, 1983.

[74] Robert E. Schapire. The boosting approach to machine learning: An overview. *Nonlinear Estimation and Classification*, 2002.

[75] Robert E. Schapire, Yoav Freund, Peter Barlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *International Conference on Machine Learning*, pages 322–330, 1997.

[76] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[77] Daniel Scharstein and C. Pal. Learning conditional random fields for stereo. In *Proceedings of the 2007 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'07, pages 1 – 8, Washington, DC, USA, 2007. IEEE Computer Society.

[78] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. In *Proceedings of the 2003 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'03, pages 195–202, Washington, DC, USA, 2003. IEEE Computer Society.

[79] S. Schlechtweg, T. Germer, and Strothotte T. Renderbots-multi-agent systems for direct image generation. *Computer Graphics Forum*, 24(2):137–148, 2005.

[80] C. Schmid and A. Zisserman. The geometry and matching of lines and curves over multiple views. *Int. J. Comput. Vision*, 40(3):199–233, 2000.

[81] T. Schoenemann and D. Cremers. High resolution motion layer decomposition using dual-space graph cuts. In *CVPR*, pages 1–7, 2008.

[82] Adrian Secord. Weighted voronoi stippling. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, pages 37–43, New York, NY, USA, 2002. ACM.

[83] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, 1994.

[84] E. Simoncelli, E. H. Adelson, and D. J. Heeger. Probability distributions of optical flow. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 310–315, 1991.

[85] K. Smith, Y. Liu, and A. Klein. Animosaics. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 201–208, New York, NY, USA, 2005. ACM.

[86] J. Sun, W. W. Zhang, X. Tang, and H. Y. Shum. Background cut. In *ECCV06*, pages II: 628–641, 2006.

[87] R. S. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transacions on Pattern Analysis and Machine Intellegence*, 30(6):1068–1080, 2008.

[88] Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan. Image-based tree modeling. *ACM Trans. Graph.*, 26, July 2007.

[89] Demetri Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8:413–242, June 1986.

[90] O. Veksler. Dense features for semi-dense stereo correspondence. *Int. J. Comput. Vision*, 47(1-3):247–260, 2002.

[91] O. Veksler. Extracting dense features for visual correspondence with graph cuts. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:689, 2003.

[92] V. Venkateswar and R. Chellappa. Hierarchical stereo and motion correspondence using feature groupings. *Int. J. Comput. Vision*, 15(3):245–269, 1995.

[93] A Vezhnevets and V. Vezhnevets. 'modest adaboost' – teaching adaboost to generalize better. In *GraphiCon*, 2005.

[94] J Wang, Y. Xu, H. Shum, and M. F. Cohen. Video tooning. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 574–583, New York, NY, USA, 2004. ACM.

[95] J.Y.A. Wang and E.H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5):625–638, September 1994.

[96] J. H Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301), 1963.

[97] Yair Weiss and Edward H. Adelson. A unified mixture framework for motion segmentation: Incorporating spatial coherence and estimating the number of models. In *Conference on Computer Vision and Pattern Recognition*, pages 321–327, 1996.

[98] J. Wills, S. Agarwal, and S. Belongie. What went where. In *CVPR*, pages I: 37–44, 2003.

[99] J. Xiao and M. Shah. Motion layer extraction in the presence of occlusion using graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1644–1659, 2005.

[100] Jiangjian Xiao and Mubarak Shah. Motion layer extraction in the presence of occlusion using graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27:1644–1659, October 2005.

[101] Chenyang Xu and Jerry L. Prince. Snakes, shapes, and gradient vector flow. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 7(3):359–369, 1998.

[102] Hui Xu, Nathan Gossett, and Baoquan Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph.*, 26, October 2007.

[103] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister. Stereo matching with color-weighted correlation, hierachical belief propagation and occlusion handling. *PAMI*, 31(3):492–504, 2009.

[104] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *ECCV '94: Proceedings of the third European conference on Computer Vision (Vol. II)*, pages 151–158, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

Copyrights

The content of Chapter 3 was published by Wiley Company and is included in this thesis

with kind permission from John Wiley and Sons, Inc:
Computer Graphics Forum, Generating Classic Mosaics with Graph Cuts,
2387 -- 2399, Dec, 2010.
Y. Liu, O. Veksler, O. Juan

The content of Chapter 4 was published by Springer and is included in this thesis

with kind permission from Springer Science+Business Media:
Springer eBook, Animated Classic Mosaics from Video,
1085 -- 1096, Nov, 2009.
Y. Liu, O. Veksler

VITA

Name:

Yu Liu

Education

University of Western Ontario — January 2007 to Present

Ph.D. (Computer Science), expected, October 2011
Supervisor: Dr. Olga Veksler.

University of Western Ontario — September 2005 to December 2006

M.Sc. (Computer Science), received, December 2006
Supervisor: Dr. Olga Veksler.

Peking University — September 2001 to July 2005

B.Sc. (Computer Science), received, July 2005,
B.A. (Economics), received, July 2005,

Related Work Experience

Research Assistant September 2006 - August 2011
Department of Computer Science,
University of Western Ontario, London, Ontario.

Limited Duty Lecturer January 2011 - April 2011
Department of Computer Science,
University of Western Ontario, London, Ontario.

Teaching Assistant September 2005 - December 2010
Department of Computer Science,
University of Western Ontario, London, Ontario.

Publications

1. Yu Liu, Olga Veksler, and Olivier Juan. Generating classic mosaics with graph cuts, *Computer Graphics Forum*, 29:2387 – 2399, 2010.
2. Yu Liu, Olga Veksler. Animated classic mosaics from video, *5th International Symposium on Advances in Visual Computing*, II:1085 – 1096, 2009.
3. Yu Liu, Olga Veksler, and Olivier Juan. Simulating classic mosaics from video, *The 6th International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*, 55–70, 2007.