University of Alberta

Robust Background Estimation with GPU Speed Up

by

Xida Chen

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

©Xida Chen
Fall 2009
Edmonton, Alberta

Examining Committee

Yee-Hong Yang, Computing Science

Mario A. Nascimento, Computing Science

Arie Croitoru, Earth & Atmospheric Sciences

Abstract

Given a set of images from the same viewpoint, in which occlusions are present, background estimation is to output an image with stationary objects in the scene only. Background estimation is an important step in many computer vision problems such as object detection and recognition. With the growing interest in more sophisticated video surveillance systems, the requirement for the accuracy of background estimation increases as well.

In this thesis, we present two novel methods whose fundamental objectives are the same, namely, to estimate the background of a set of related images. In order to make our methods more general, we assume that the input images can be taken either from the same viewpoint or from different viewpoints. Both methods combine information from multiple input images by selecting the appropriate pixels to construct the background. Our first method is a scanline energy optimization method, and our second method is based on graph cuts optimization. We apply these two methods to datasets with different feature and the results are encouraging. Furthermore, we use the CUDA (Compute Unified Device Architecture) programming language to make full use of the GPU processing power. GPU stands for Graphics Processing Unit, which employs parallel processing and is more powerful than the CPU. In particular, we implement an efficient graph-based image segmentation algorithm as well as a linear blending method using the CUDA programming language for acceleration, both of which are used in our first method. The speedup of our GPU implementation can be 20 times faster.

Acknowledgements

First of all, I would like to thank Dr. Herb Yang sincerely for taking me as his student and allowing me to pursue my research interests. Throughout the past two years, he has been constantly sharing his considerable wisdom in this field, and providing invaluable guidance in the supervision of this thesis work. He is absolutely the most enjoyable person I could imagine as a research advisor.

I am very grateful to the committee members: Dr. Mario Nascimento and Dr. Arie Croitoru, for their time on reading and commenting on this work. I am sure that their suggestions are greatly valuable to improve my thesis.

I must also thank my current and former group members in Computer Graphics Lab at University of Alberta, Cheng Lei, Daniel Neilson, Yilei Zhang, Gopinath Sankar, Omar Rodriguez-Arenas, Jason Gedge and Yufeng Shen, for many illuminating discussions. In particular, I would like to thanks Cheng Lei and Yilei Zhang, their help has been greatly appreciated. I had the good fortune of investigating topics that overlap with the work of Cheng Lei, and his advice is always useful to my research.

Last but not least, I wish to thank my parents, for encouraging me in the pursuit of my dreams. They have set an example of hard work and patience that I strive to live up to. I couldn't have done it without their support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Background estimation is to construct the background without occluding foreground objects from an image sequence that includes occluding foreground objects in the scene. In some traditional methods, such as [42] [17] [35] [30], the camera is assumed to be static, and the background objects are defined to be the stationary objects throughout the input sequence. In this thesis work, the definition of the background is the same as traditional methods if the camera is static. However, if the camera is moving, the background is defined to be the objects that are in the same position among the input images after the images are aligned to each other. Background estimation is an important task and has found many applications in the area of computer vision because knowing the background image of a sequence simplifies many computer vision problems. For example, background estimation is normally the first step in a background subtraction algorithm [21] [17] [29]. The estimated background is used as a mask for each input frame. In video segmentation, the background of a scene provides a lot of information to extract foreground objects. Several effective algorithms have been proposed [16] [53] for real-time foreground/background segmentation. In video surveillance, traffic monitoring and object tracking, the background of a scene provides useful information to the segmentation of moving objects.

With the fast advancement of hardware, digital cameras are so popular nowadays that they appear in many different forms from low cost web cameras to high end single lens reflex cameras. Despite the proliferation of cameras, their views are still limited by the field of view in the design of the cameras. For people who are not satisfied with images from a single view, they can stitch images from multiple views taken by one camera to form a panorama with a larger field of view. As a matter of fact, many cameras come with software that can do image stitching. Some newer ones can even do it in the camera. Hence, the field of view is not an issue any more. As well, Brown et al. [11] develop an automatic image stitching system, which creates nice-looking panoramas from multiple view images. The main limitation of this method is that the final image may have undesirable objects in the scene. For example, when someone is in a room and wants to take images of the beautiful natural scenery outside of the window, the scene may be occluded by the window frames, which will remain in the final panorama when the automatic stitching system is used. A more challenging example is to get rid of a fence which happens to be in the foreground of the images (Figure 1.1). Not only will the fence prevent the automatic stitching system from creating a correct panorama, it will also remain in the final panorama. Recently, Liu et al. [40] propose a method for image de-fencing using an image inpainting technique, which is to synthesize the background that is occluded and is different from estimating the background. Therefore, a method to estimate background from multiple images with undesirable foreground objects removed has many practical applications.

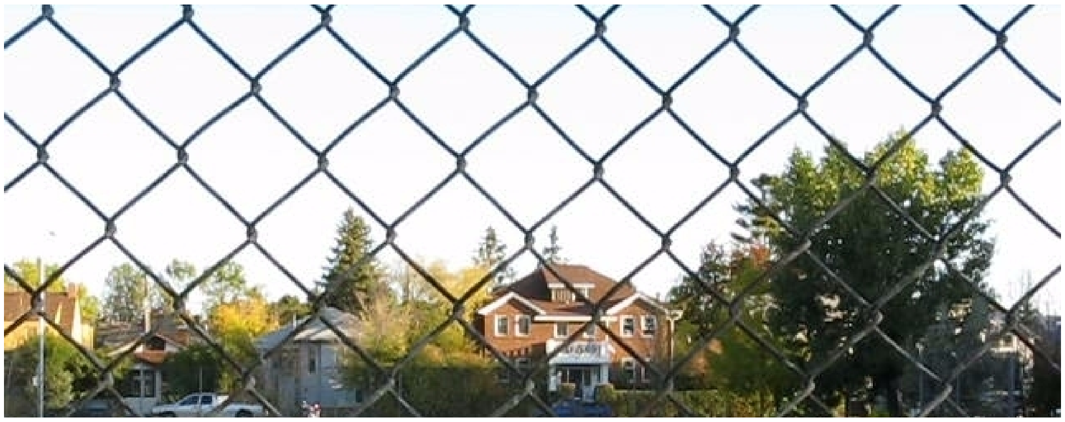


Figure 1.1: An image with a fence as the foreground object.

Humans have an amazing ability to interpret the scenes from their images. Even from a single image, humans can obtain a lot of information of the scene such as ground orientation, relative positions of major landmarks, and so on. A typical explanation for such an ability is that our perception of a scene is based on both the immediate sensory evidence and our visual experience and interactions with the real world. In fact, humans take advantage of geometry, shapes of objects that are shown in an image, and determine the relationship among them. In particular, given a set of images taken from the same viewpoint, a person can identify the background objects as well as foreground objects easily. The reason is because we can infer the depth information from the image sequence. However, for a computer, to obtain the depth information from an image sequence could be even more difficult than to estimate the background. Therefore, the depth information of an image is normally assumed to be unknown when tackling the problem of background estimation. There are several factors such as camera motion, scene brightness and moving objects, that make the background estimation problem challenging. Even when the factors listed above do not exist, the problem is still difficult because the background in some areas might be visible for only a small amount of time in the input image sequence.

In this thesis, we propose two novel algorithms to estimate the background using a set of related images. We assume that the cameras can be either static or moving freely, which means that the images can be from the same viewpoint or different viewpoints. As a result, the illumination conditions among input images could vary significantly. Given an input image sequence, the output of our algorithms is an image with only background objects. There is an important assumption for both of our proposed methods, namely, each background pixel has to be disclosed at least once throughout the entire image sequence. This is an essential assumption for all the background estimation methods that have been developed so far. Some algorithms even make an assumption that every background pixel is exposed for more than 50% of the whole sequence [45]. Under this assumption [45], the background can be removed by a simple median filter. Furthermore, a natural image sequence can be complex and such an assumption rarely holds. As a result, the constraint of our algorithm that requires every background pixel to appear at least once is more general and practical. A unique feature of our first algorithm is to integrate segmentation information into the objective function, the optimization of which gives the estimated background. Our second method incorporates an image inpainting technique to define a new predicted term, which has never been done before. Because of the integrated image inpainting technique, our second method has an additional constraint, which requires some background regions never be occluded in the image sequence. We observe that this is also an essential assumption for most of the previous methods [58] [13] [28]. Our algorithms utilize multiple input images of a scene and select the appropriate pixels to construct the background. In the output image, the transient foreground objects will be removed.

The proposed algorithms address several problems which are normally present in background estimation.

➢ Illumination difference among input images. Because we assume that the input images can be taken from different viewpoints, or at different times, the illumination change can be significant among them. Figure 1.2 shows that the appearance of the ground is quite different when the same scene is taken at different times of the day. As a result, the pixel value that corresponds to the same physical point can be different in different images.

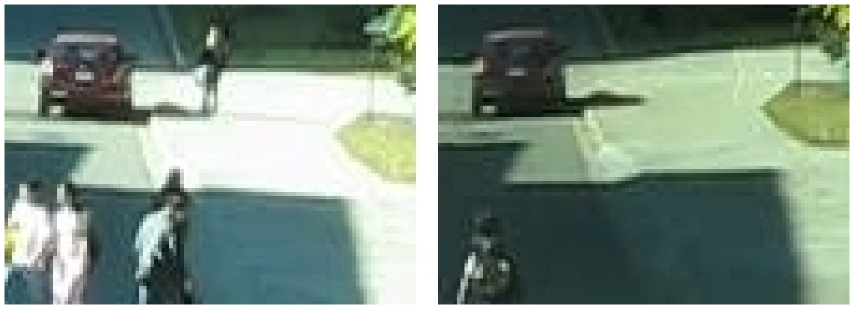


Figure 1.2: Two regions from two different images of the same scene. Although these regions are at the same position, the illumination changes significantly between them.

➢ Artifacts when fusing images. Seams could exist in the output image due to varying illumination conditions.

➢ Robustness of the algorithm. A robust algorithm should have minimal assumptions of the foreground or the background.

Besides the high quality results, the processing time of an algorithm is also important in real world applications. Sometimes a particular task may require a real-time solution. To address this issue, parts of the proposed algorithm are implemented using the GPU (Graphics Processing Unit). The GPU is specifically designed for compute-intensive, highly parallel graphics computation [46]. More recently, it has been applied to general purpose computation [2]. It is a multithreaded, many core processor with tremendous computational horsepower and very high internal memory bandwidth. For example, the Floating-Point Operations per Second (FLOP/s) for Geforce GTX 295 is about 1800 Gflops/s, while it is only about 100 Gflops/s for a 3.2GHz Intel Harpertown. The memory bandwidth is 223.8 GB/s for GTX 295 and about 14 GB/s for 3.2GHz Harpertown. The GPU is very efficient at computer graphics, and its highly parallel structure makes it more effective in some forms of computation than the general purpose CPU. In particular, the GPU is the best choice to address problems that require data-parallel computations, that is, the same program is executed on many processing elements in parallel. Recently, NVIDIA has developed a programming model for multithreaded processors. CUDA, short for Compute Unified Device Architecture, which is known as a parallel computing architecture, is a C-style programming language that is well suited to expose the parallel capabilities of the GPU to applications.

In this thesis, we implement an efficient graph-based image segmentation algorithm and a linear blending method using the CUDA programming language for acceleration purpose. Both of these methods are used in our first background estimation method.

The remaining part of this thesis is organized as follows. The second chapter discusses some related works. In Chapter 3, we present two novel algorithms for background estimation. Then in Chapter 4, we give more details of the implementation of an image segmentation algorithm and a blending algorithm on the GPU. Finally we conclude and introduce future work in Chapter 5.

Chapter 2

Related Works

Driven by the increasing popularity of panoramic photography, many software programs for creating panoramic images have been developed, e.g. Adobe Photoshop and Corel PhotoImpact. Most of the methods for image stitching developed so far are completely automatic [11]. A drawback of some typical image stitching systems is that ghosts maybe present in the final panorama. As a result, methods to eliminate ghosting effect in panoramic images are proposed. The methods proposed for ghost elimination normally select the regions from the input frames to composite a panorama. However, the regions that are selected by these methods may not always be the background regions.

On the other hand, background estimation, as well as background modeling, is a common problem in many areas of computer vision. Researchers have been working on this topic for more than a decade. In this chapter, we discuss some related works in both the image stitching and the background estimation areas.

2.1 Image Stitching

Brown et al. [11] develop an automatic image stitching system to create panoramic views from image datasets. The objective of this system is to operate on a database of images and find all the matching images. Then the subsets of matching images can be combined into nice-looking panoramas. A unique feature of this system is that there is no need to order the input images and the images can be taken from any viewpoints. The system utilizes Lowe's Scale Invariant Feature Transform (SIFT)

[43] [44] to detect local features in images. The SIFT features are based on the appearance of the object at particular interest points. They are geometrically invariant under similarity transforms and under affine changes in intensity. Therefore, SIFT features are suitable for matching different images of an object or a scene. For example, given two images with overlapping regions, the image stitching system detects SIFT features in both images and finds the matching features. After that, the matching features are used for image registration, which is to compute a homography between two images. A homography is a projective transform that maps points from one projective plane to another projective plane. The details of homography is discussed in the Appendix A.

Since the input images could be taken from different viewpoints, the illumination conditions might change significantly among them. However, the panoramic image created by this system is smooth overall (Figure 2.1). When the system performs multi-band blending [12] to render the panorama, it uses all the source images that project into a given destination pixel and computes a weighted blend of the source images. As a result, if there are transient objects in the scene, they will remain and appear blurred in the panorama (Figure 2.2). Some applications may accept blurred regions, however, many applications may prefer a single focused image.

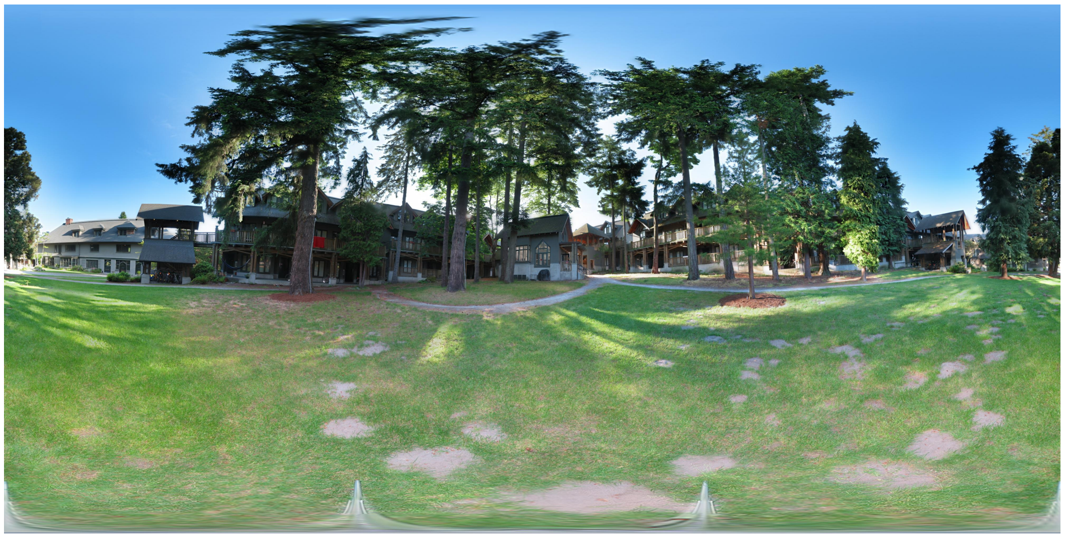


Figure 2.1: Panoramic image created by the automatic image stitching system. (from http://www.cs.ubc.ca/∼mbrown/panogallery/panogallery.html)

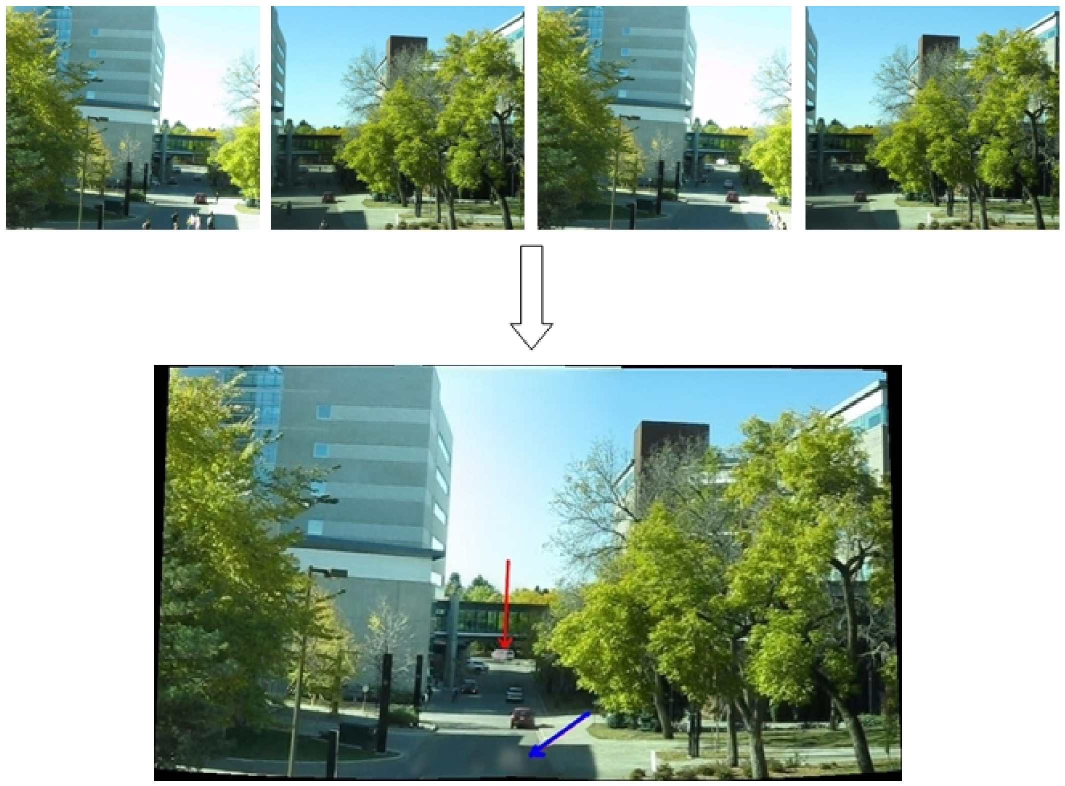


Figure 2.2: Panoramic image created by the automatic image stitching system with moving objects in the scene.

Because of the existing blurred regions in the results produced by a typical image stitching system, several algorithms have been proposed to eliminate the ghosts in the panoramic images. Davis [19] proposes a method that cuts the images between regions with moving objects and finds the best cut using Dijkstra's algorithm. In particular, this method considers the overlapping regions in a pair of registered images at a time. The relative difference between two input images are calculated and provided as a measure of similarity. If two images have identical pixels in the overlapping regions, then the difference is zero. However, if the pixels are from different objects, the intensity difference can be large. Then this method uses Dijkstra's algorithm to find a path dividing the overlapping section. The path avoids areas where the source pair is inconsistent due to the moving objects. This method is extended to a sequence of input images. It first compute the mosaic of two input images, then the mosaic and another input image is used as input to the above described process. The process is iterated for each input image.

Uyttendaele et al. [56] propose a method to eliminate the ghosts in image mosaic. The method determines which objects to keep and which ones to remove in the final composite. The first step is to detect the unstable regions, which are regions that differ across images. The method only searches for regions of difference (ROD) in the overlap areas of the input images. For each input image, pixels that differ by more than a pre-defined threshold from pixels in the overlap regions are flagged as unstable pixels. Figure 2.3 shows how the RODs are detected for each input image. There are three images in the figure with a smiling face as the moving object. The shadow regions that are shown in Figure 2.3(b) are the RODs. Then the algorithm uses the RODs as a mask and determines which image should be selected to fill in each ROD. It gives each ROD a weight that is proportional to its size and proximity to the center of its image. In other words, the larger and more central RODs are assigned higher weights. The computation of the weight is described in [50]. For a certain ROD, an image with the minimum weight in that region is selcted. As a result, only one image is used to fill in each ROD. The comparison between the result without ghost elimination and the result with it is given in Figure 2.4. In the image, we can see that there is no blurred region with the ghost elimination method. However, the method cannot guarantee that all the regions in the final panorama are background regions.

Levin et al. [39] propose a method for image stitching in the gradient domain. The goal of their method is to make the seams between input images invisible. The method presents a cost function to measure the stitching quality in the intersection regions in the gradient domain and the minimization of that cost function computes the mosaic image. In particular, the cost function is a dissimilarity measure between the derivatives of the mosaic image and the derivatives of the input images.

In the system described in [11], multi-band blending [12] is applied at the last step to create a smooth panorama. Besides this blending method, linear blending is another straightforward strategy. The basic idea of linear blending is simple. In order to combine the information from multiple input images for a certain pixel in the mosaic image, each input image is assigned a weight $W(x, y) = w(x)w(y)$. The weight is computed based on the location of that pixel. $w(x)$ is set to be 1 at the

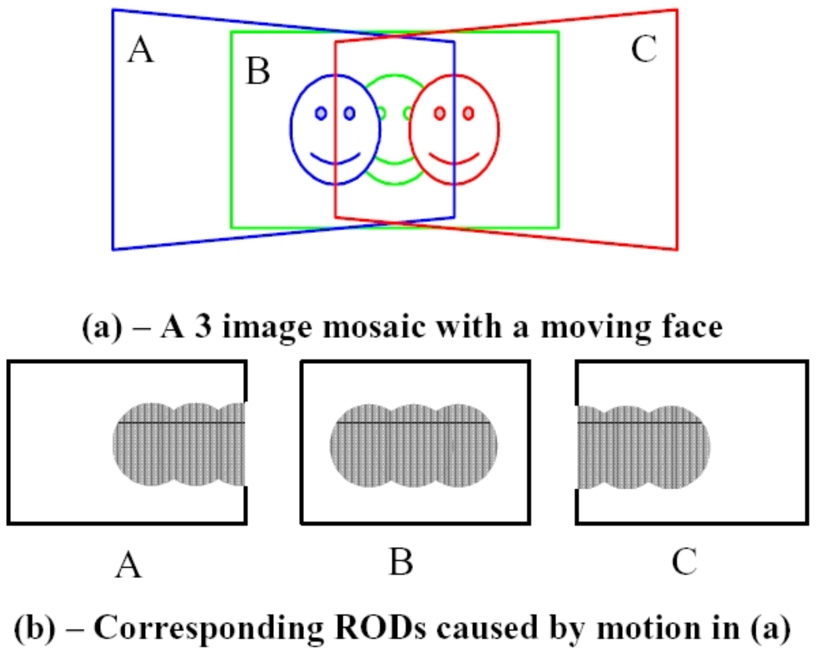

A
B
C
(a) – A 3 image mosaic with a moving face
A
B
C
(b) – Corresponding RODs caused by motion in (a)

Figure 2.3: Detection of RODs for each input image and the graph representation. (from [56])

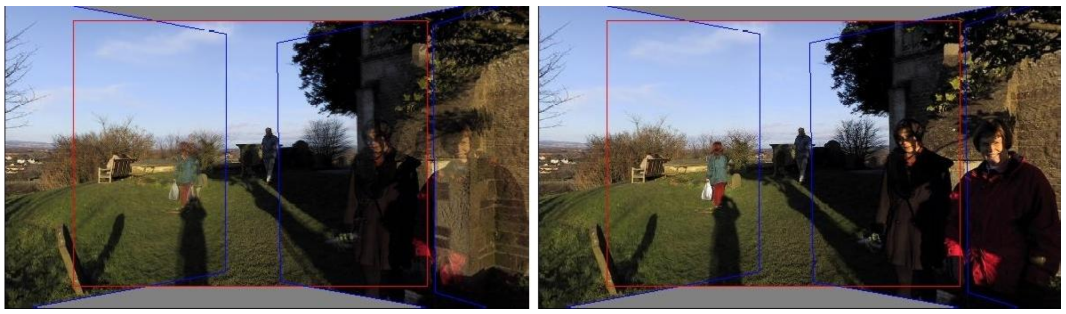


Figure 2.4: Left: Result when applying a typical image stitching system. Right: Result by applying some method for ghost elimination. (from [56])

center of the image and it varies linearly to 0 at the edge. Combining watersheds and graph cuts methods [27] is another approach for image blending. This method only focuses on the overlap regions . It first applies the watershed approach to divide the overlap regions into disjoint segments. Then a cost based on the photometric difference along the seams is assigned to each segment and graph cuts is applied to minimize the cost. The method has several advantages. For example, graph cuts guarantees the globally optimal solution for the overlap regions. Also, each overlap region are independent of each other, which means that the algorithm is suitable for parallel implementation.

2.2 Background Estimation

The first background estimation algorithm could be traced back 19 years ago to the work of Long and Yang [42] for detecting moving objects in a scene. The fundamental assumption for the proposed methods is that for each pixel, the background value is always stable and always stays the longest through the image sequence. The first proposed method is called the "Smoothness Detector Method." It only searches for stable regions through the image sequence and ignores the unstable areas. The method defines a threshold, and uses a window (in time) to detect stable values. That is, the window is moving along the temporal trace of the values of a pixel. If the values inside that window all fall within the range of a pre-defined threshold, then the average value of those pixels inside the window is calculated. When the size of the window is increased, the average value needs to be updated. The final background value is the average value of the pixels in the maximum window.

The second method proposed in [42], called the "Adaptive Smoothness Detector Method," is an improvement to the first method described above. Instead of using a fixed threshold for the whole process, both the threshold and the length of the window are adaptive. The method starts with a very small threshold and a very large window length (half of the number of images in the sequence). Although the chance of finding a stable region that fits inside such a long window is small, such region is assumed to be the background if one is found. And the average value of the pixels inside that window is used as the background value. If the regions inside the window is not stable, the length of the window is decreased and the above process is repeated. The length of the window keeps decreasing until it reaches a minimum value. After that, the threshold increases and the length of the window is set to be half of the number of images in the sequence again. The whole process is terminated when all the background pixels are determined or the length of the window reaches a minimum.

Both methods described above make the assumption that a background value stays longest. Similar to that, median filtering assumes that the pixel stays in the background for at least half of the total number of frames throughout the entire image sequence. Suppose we have N input frames, then the median filter operates as follows. For a certain pixel at location (x, y), it computes the median value in the temporal direction over N frames, and then the median value is set to be the background value for this pixel. This process is repeated for all the pixels in the image. Since the underlying idea of median filtering is simple, it is one of the most commonly-used background estimation techniques. It has been applied to many applications. For example, Cutler et al. [18] use it to construct background for detecting periodic motion. Cucchiara et al. [17] extend the median filtering to color, and use it to detect not only moving objects, but also shadows and ghosts in video streams. In their experiments, they prove that median filtering is effective and has less computational cost than Gaussian filtering or other complex statistical methods. Median filtering has also been applied to track moving objects in a video [60], in monitoring [41] and in traffic surveillance [26]. Traffic control is very important in real-world applications. Figure 2.5 shows the flow chart of a particular stage in the monitoring system used in [26].

Driven by the success of the simple median filtering method, McFarlane and Schofield [45] propose a recursive filter to estimate the background, which is referred to as the "approximated median filter." The estimate of the median is increased by one if the corresponding pixel in the current input image is larger in value, and decreased by one if smaller. Finally, the estimate converges to a value for which half of the input values are larger and half are smaller, which is the median. This method has been applied to traffic monitoring [49].

Although it has been applied to many areas, median filtering has its own limitation. It has an essential assumption that for every pixel, the background has to appear in more than half of the number of the images. If this assumption were false, the result produced by a median filter would be wrong. In background estimation, it is likely that in some particular regions, the foreground objects appear more often than the background objects. Figure 2.6 shows a result by applying the median filter. The top row are three out of 25 images that are extracted from a traffic monitoring

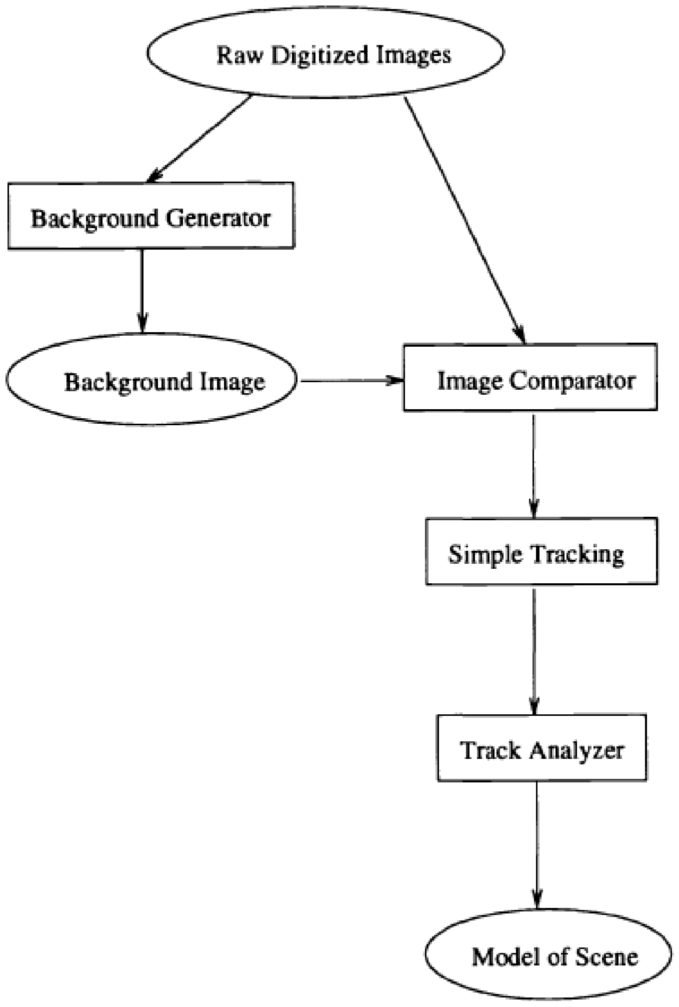


Figure 2.5: The flow chart of the road modeling stage (from [26])

sequence, and the bottom row shows the result. The error region are indicated by a red circle where the foreground shows up more than half of the total number of input images.

Kalman filter is a recursive technique that has been widely used for tracking linear dynamical system under Gaussian noise. For example, Koller et al. [35] use Kalman filter for multiple car tracking. The method uses an adaptive background model which is updated by a Kalman filter formalism, which allows dynamics in the model as lighting conditions change. The Kalman filter formalism is given in as follows.

$$B_{t+1} = B_t + (\alpha_1(1 - M_t) + \alpha_2 M_t)D_t \tag{2.1}$$

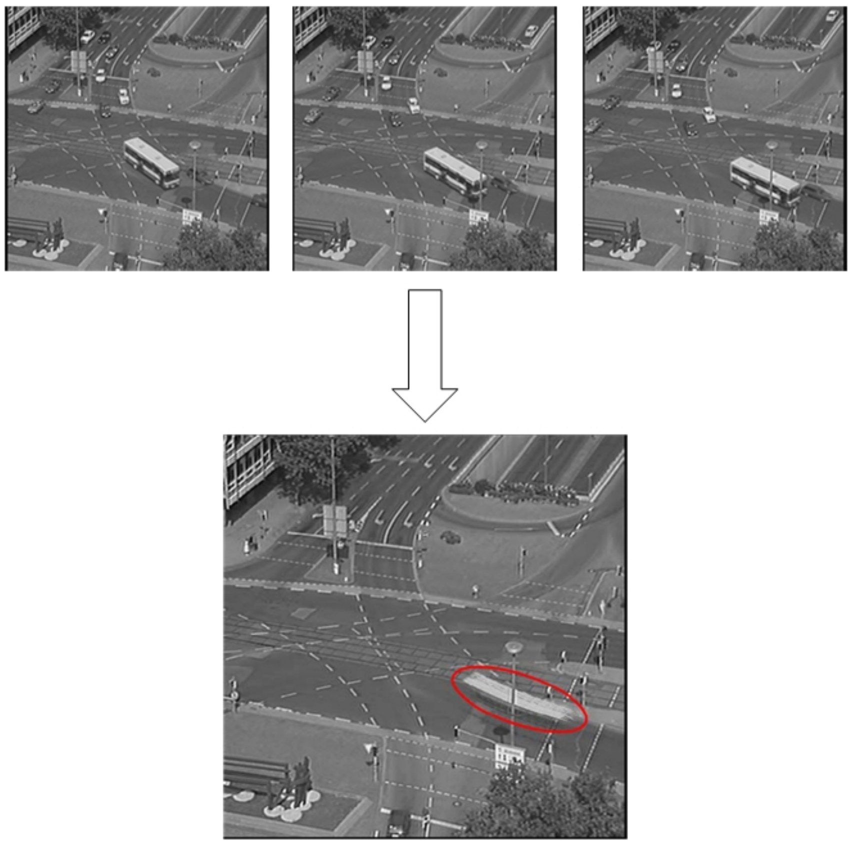


Figure 2.6: Result by median filtering (from [13]). Error exists in the circled region.

In the above equation, B_t is the background model at time t. The background model is updated recursively, and D_t is the difference between the current frame and the background model. The value of α_1 and α_2 are obtained based on an estimate of the rate of change of the background in the original implementation. However, Koller et al. [35] use small constant values $\alpha_1 = 0.1$ and $\alpha_2 = 0.01$. M_t is a binary hypothesis mask and is for identifying moving objects in the current frame. The mask is calculated by applying a threshold to the difference image D_t. In particular, for a certain image position p,

$$M_t(p) = \begin{cases} 1 & \text{if } |D_t(p)| > T_t \\ 0 & \text{else} \end{cases} \tag{2.2}$$

where T_t is the threshold. The block diagram of this method is provided in Figure 2.7.

Besides the implementation described above, there are many other versions of Kalman filter proposed for background modeling, the difference is mainly in the

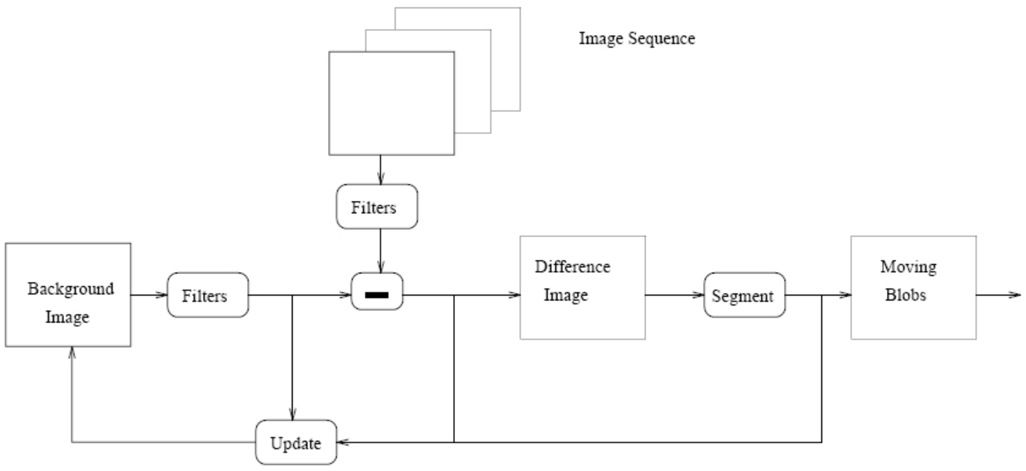


Figure 2.7: Block diagram of background update procedure in the system for multiple car tracking. (from [35])

state spaces used for tracking. For example, there are some simple implementations using luminance intensity only [57] [32] [31] [8]. A more sophisticated implementation uses both intensity and its temporal derivatives [34].

While Kalman filter tracks the evolution of a single Gaussian, the Mixture of Gaussians method tracks multiple Gaussian distributions at the same time. It is first proposed for background modeling in [24] and then in [51] [52], a different implementation is presented. This method models a pixel distribution as a mixture of K Gaussians. In particular:

$$P(I_t) = \sum_{i=1}^{K} w_{i,t} \cdot \eta(I_t, \mu_{i,t}, c_{i,t}) \tag{2.3}$$

where P is the probability distribution. K is the number of distributions and it normally ranges from three to five. $w_{i,t}$ is the portion of the data accounted for by the i^{th} Gaussian. I_t is an input pixel at time t, $\mu_{i,t}$ the intensity mean and $c_{i,t}$ the covariance matrix. Here η is a Gaussian probability density function defined as follows.

$$\eta(I_t, \mu, c) = \frac{1}{(2\pi)^{n/2}|c|^{1/2}} e^{-\frac{1}{2}(I_t-\mu_t)^T c^{-1}(I_t-\mu_t)} \tag{2.4}$$

For each input pixel I_t, the method needs to find the closest Gaussian distribution among K of them. If the pixel value is with 2.5 standard deviation of a distribution, then it is defined to be a match with this Gaussian distribution. If there is a match for this pixel, then the parameters of the matched Gaussian distribution are updated as follows.

$$w_{k,t} = (1 - \alpha) w_{k,t-1} + \alpha \tag{2.5}$$

$$\mu_{k,t} = (1 - \rho) \mu_{k,t-1} + \rho I_t \tag{2.6}$$

$$\sigma_t^2 = (1 - \rho) \sigma_{t-1}^2 + \rho (I_t - \mu_{k,t})^T (I_t - \mu_{k,t}) \tag{2.7}$$

In the above equations, α is a pre-defined learning rate and $0 \leq \alpha \leq 1$. ρ is the learning factor for adapting current distribution and is approximated to be:

$$\rho \approx \frac{\alpha}{w_{k,t}} \tag{2.8}$$

If there is no match for the given pixel, then the least probable distribution is replaced with another distribution with mean I_t, a high initial variance σ and a low prior weight w.

After the parameters are updated, the sum of weights is normalized to one. Finally, the method determines whether or not I_t is a background pixel. The Gaussians are sorted by the value of $w_{i,t}/\sigma_{i,t}$. If a distribution has a high rank, which means with high probability and low variance, then it has a high probability to be the background. Suppose $i_1, i_2, ..., i_K$ are the ordered Gaussian distributions, then the first M distributions satisfying the following criteria are declared as the background distributions:

$$\sum_{k=i_1}^{i_M} w_{k,t} > T \tag{2.9}$$

where T is the weight threshold. Then I_t is labeled as a foreground pixel if its value is within 2.5 standard deviation from the mean of any of the background distributions.

There are some recent improvements to the original version of the mixture of Gaussians method, such as a sensitivity analysis of parameters [25], and improvements on complexity and adaptation [33] [47] [38].

Gutchess et al. [30] apply optical flow analysis to background modeling. This method generates hypotheses by locating intervals of relatively constant intensity, then the likelihood of each hypothesis is evaluated using optical flow information from the neighborhood around the pixel and the most likely one is set to represent the background. Suppose the intensity of a pixel through the entire sequence is represented as $< a_0, a_1, ..., a_n >$, the method first finds all sub-intervals that satisfy the following two conditions:

$$w \leq j - i \tag{2.10}$$

$$\forall(s,t)|a_s - a_t| \leq \delta_{max} \tag{2.11}$$

In their implementation, the value $\delta_{max} = 10$ and $w = 6$ were used. The subsequence $< a_i, ..., a_j >$ is a set of candidate hypotheses and one of them will be selected to represent the background. The method then computes the input flow and output flow of each pixel from the local neighborhood $\mathcal{N}$.

$$f^+(x,y) = \sum_{i \in \mathcal{N}} \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}[(x - x_{i,1}/\sigma)^2 + (y - y_{i,1}/\sigma)^2]} \tag{2.12}$$

$$f^-(x,y) = \sum_{i \in \mathcal{N}} \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}[(x - x_{i,2}/\sigma)^2 + (y - y_{i,2}/\sigma)^2]} \tag{2.13}$$

where $f^+(x,y)$ denotes the input flow and $f^-(x,y)$ the output flow for the pixel located at (x,y). Denote F_t^+ to be the input flow and F_t^- the output flow for the entire image frames. Then the "net flow" is obtained by subtracting the output flow from the input flow, that is,

$$N_t = F_t^+ - F_t^- \tag{2.14}$$

After that, the accumulated net flow S_t is served as an estimate of the total amount of the input and output flow up to time t.

$$S_t = \sum_{i=0}^{t} N_i \tag{2.15}$$

This measure is updated for each frame in the sequence. The background likelihood is defined as $L_t = -S_t$. Finally the candidate hypotheses at each pixel are computed by the following equations.

$$\mathcal{L}_{[t_1,t_2]} = \frac{\sum_{i=t_1}^{t_2} l_i}{t_2 - t_1} \tag{2.16}$$

where $\mathcal{L}_{[t_1,t_2]}$ is the average likelihood and $[t_1, t_2]$ is the stable interval. The interval $[\alpha, \beta]$ which has the largest average likelihood is selected to represent the background. Algorithm 1 is a step-by-step description of the optical flow analysis.

Algorithm 1 Local Image Flow Algorithm

```
➢ For each pixel, find all intervals of stable intensity.
➢ Initialize accumulated net flow and likelihood to be zero.
➢ Compute the optical flow for each pair of consecutive image frames $I_t$ and $I_{t+1}$.
➢ Calculate the net flow $N_{t+1}$ between the two frames by applying Equation 2.14.
➢ Compute the accumulated net flow for time $t + 1$ using Equation 2.15.
➢ The likelihood for this pixel to be background is computed by $L_{t+1} = -N_{t+1}$.
➢ For each pixel, compute the average likelihood for each stable interval and select the interval with maximum likelihood.
➢ The mean and variance of intensity over the selected interval are used as parameters for the background model.
```

In computer graphics, Agarwala et al. [1] develop a general, and powerful framework for combining a set of images into a single composite image. This framework has been used for a wide variety of applications. For example, to create an image with all the best elements from the input images, to extend the depth of field, to create panoramic mosaic from multiple images, and so on. A typical interaction process is described as follows. Suppose we have a set of input images, then the first image in the sequence is the initial composite image. The user selects the objective such as "Max Luminance", and the system defines the cost function according to the objective. After the cost function is defined, the method uses graph cuts to minimize the cost and chooses the labels for the composite. The user can manually select the labels for refinement purpose. Finally gradient-domain fusion [22] is applied to remove any visible seams that exist in the composite image.

Besides the above described applications, this framework can be used for background reconstruction. It defines a cost function which includes a data penalty and an interaction penalty. The cost function is given in Equation 2.17.

$$C(L) = \sum_{p} C_d(p, L(p)) + \sum_{p,q} C_i(p, q, L(p), L(q)) \tag{2.17}$$

where L is a label, p and q are two neighboring pixels. C_d is the data penalty and

C_i is the interaction penalty. The data penalty is defined according to the objective selected by the user. For background estimation, the maximum likelihood should be used as the objective. Then the data term is the probability of the color at $I_{L(p)}(p)$, given the probability distribution function from the color histogram of all pixels. Here $I_i(p)$ denotes pixel p in input image i. The interaction penalty gives a higher cost to visible seams. Then the method uses graph cuts to minimize the cost and select corresponding labels to construct the background. However, using graph cuts with the defined cost function cannot always provide good results. It still requires user interactions for refinement. Figure 2.8 shows a result with the defined cost function and graph cuts optimization. Some of the foreground objects still show up in the rectangle region. To remove them, the user can select the eraser objective to replace the rectangle region with a region where the background is visible.

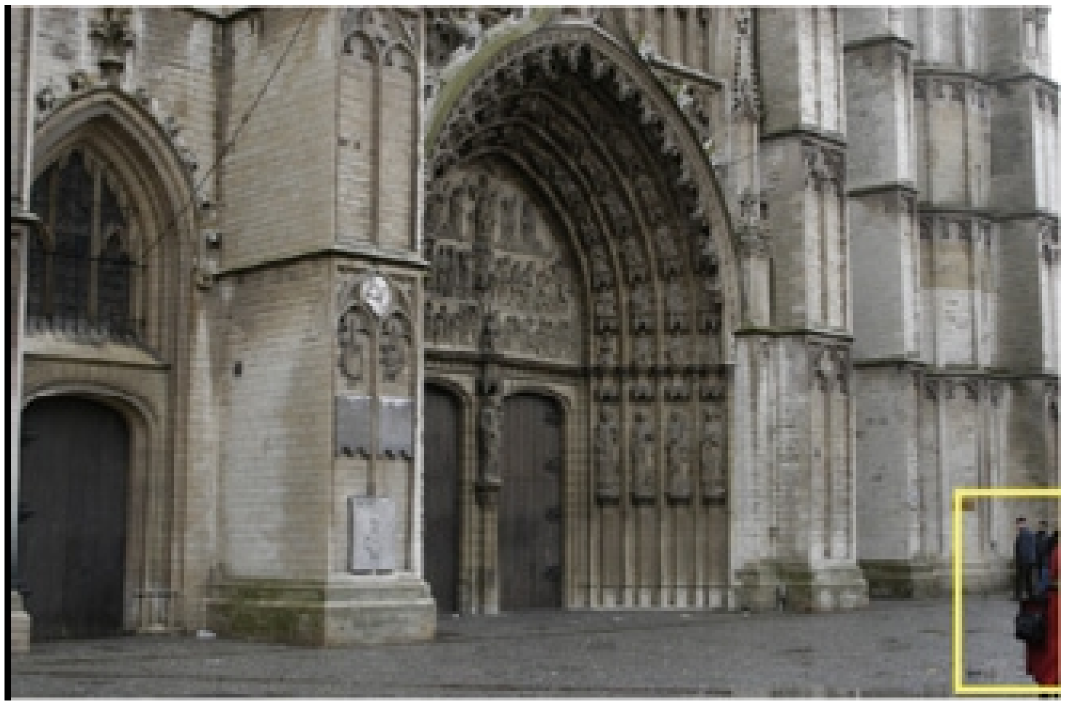


Figure 2.8: Constructed background by the defined cost function and graph cuts optimization. (from [1])

More recently, Colombari et al. [14] propose a patch-based background initialization technique to construct background in cluttered image sequences. The method can be divided into two main stages. First, the method finds a seed patch to represent the stable background region. The method divides the images into multiple windows with window size $N \times N$, and the windows overlap by half of their size in both dimensions as shown in Figure 2.9. In their experiments, they set the size of the window $N = 17$ for images with size 200×260. Let $v_{(W,f_1)}$ be the patch at the window W in frame f_1. Since this method uses patches to construct the final background, a clustering method is applied to reduce temporal redundancy. That is, the clustering method is applied to patches at the same location but in different frames. It measures the distance between two image patches by calculating the Sum of Squared Distances (SSD) between them. Specifically, the distance between $v_{(W,f_1)}$ and $v_{(W,f_2)}$ is obtained by the following formula

$$SSD(W, f_1, f_2) = \frac{1}{2\sigma_m^2} \sum_{x,y \in W} ||v_{x,y,f_1} - v_{x,y,f_2}||^2 \tag{2.18}$$

where v_{x,y,f_1} is the pixel value of (x, y) in frame f_1 and σ_m^2 the deviation of the Gaussian noise used in the pre-processing stage. This method uses a threshold, called *cutoff distance* to measure if two image patches should be in the same cluster or not. If the SSD is smaller than the threshold, then the two patches are in the same cluster. After clustering, the seed patches which are the representative of the background are selected. That means, if a certain cluster contains most of the image patches and because the patches belong to this cluster are similar to each other, which means that a patch in this cluster stays the longest time, then the patches in this cluster are the representative of the background.

After the seed patches have been selected, the method starts region growing from these patches. Let us consider the patch v_{W_0} in Figure 2.9 to be the seed patch that is selected as the representative of the background. The next step is to select the patches for the neighbor of W_0. For example, the method needs to choose one of the patches from different frames at W_1 in order to construct the background. The selected patch has to fulfill the following two requirements:

(1) It has to depict the same scene points as the background patch that overlaps with W_0.

(2) It has to be the "best continuation" of the background in the non-overlapping part with W_0.

In order to determine if the patch depicts the same scene points with W_0 in

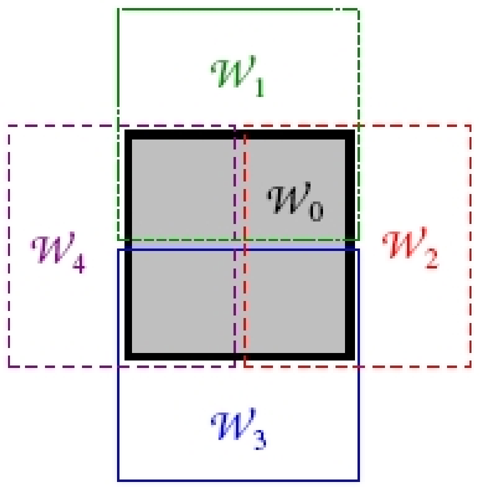


Figure 2.9: Overlapping windows. (from [14])

the overlap part, the SSD in the overlap region between W_0 and W_1 is computed. Suppose $v_{(W_0,f_0)}$ is the seed patch that represents the background and $v_{(W_1,f_1)}$ is the candidate patch, the SSD in the overlap part is computed by:

$$SSD(W_0 \cap W_1, f_0, f_1) = \frac{1}{\sigma_{f_0}^2 + \sigma_{f_1}^2} \sum_{x,y \in W_0 \cap W_1} |v_{x,y,f_0} - v_{x,y,f_1}|^2 \qquad (2.19)$$

Similar to the clustering method, if the SSD is below a certain threshold, then this patch can be used to represent the background. Finally, the method selects the patch that has the "best continuation" of the background patch. Because the most important assumption in the method is that the background is cluttered, the visual cuts are more likely to appear in the background. Therefore, when considering two candidate patches, if the visual cuts appear in one of them only, then this patch is considered more likely to be a background patch than the other ones. As a result, when the region growing stage is finished, the background is recovered by the selected patches. Although this method provides some promising results, it assumes that the background is cluttered, which is not always true in the real world.

Xu and Huang [58] propose a simple, yet robust approach for background estimation. The assumption made in this method is that the background objects are stationary in all the input frames and the background is disclosed at least once at each pixel, which is the minimal assumption for background estimation. This method defines a cost function that is based on visual smoothness only, in particular,

$$E_s = \sum_{(p,q)\in\mathcal{N}} ||I_{L_p}(p) - I_{L_q}(q)|| \tag{2.20}$$

where E_s is the energy. Suppose $\mathcal{L}$ is the labeling space, then $I_{L_p}(p)$ is the value of a pixel p in the input frame L_p and $L_p \in \mathcal{L}$. $\mathcal{N}$ is the set of all neighboring pixels and $|| * ||$ refers to the l_1 norm.

In the above equation, we can see that the cost function is a measure of visual smoothness, which is simple and waives the need to tune parameters. This method performs energy minimization using Loopy Belief Propagation [59]. When minimizing the cost function, the method selects labels and copy the pixels from input images to composite the background. The final step of this method is to apply gradient-domain fusion when constructing the background image.

However, this approach seems to work on images with smooth background only. Figure 2.10 shows two background images shown in [58]. We can see that the background is relatively smooth but not complex.

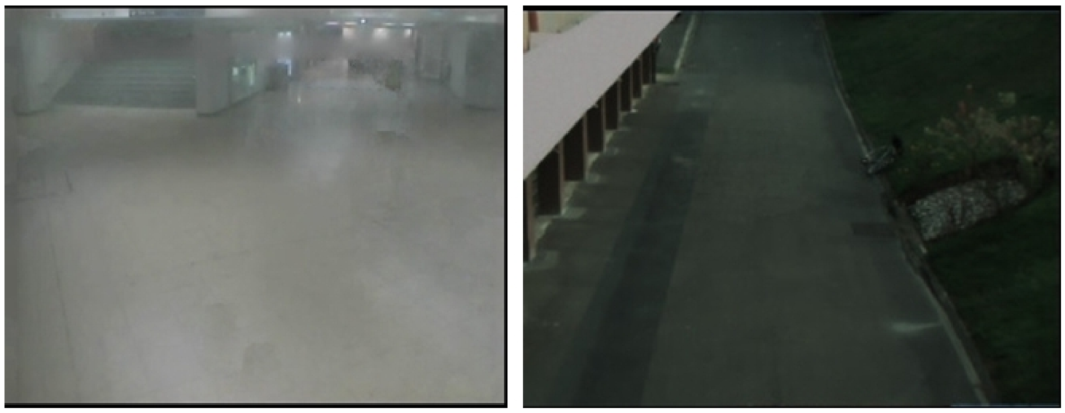


Figure 2.10: Background estimation results. (from [58])

Compare to the images used in [58], the scenes in the real world can be much more complicated. As a matter of fact, the method proposed in [58] fails when the background is complex. Since the energy function is simple, we use the framework [54] that is publicly available to re-implement this method. The details of this framework is introduced in Chapter 3. This framework contains several energy minimization methods, we use the belief propagation software developed by Tappen et al. [55] and apply this method to a dataset used in [1]. Figure 2.11 shows that there are still many error regions in the image and the result clearly shows that a cost function measuring visual smoothness only is not enough.

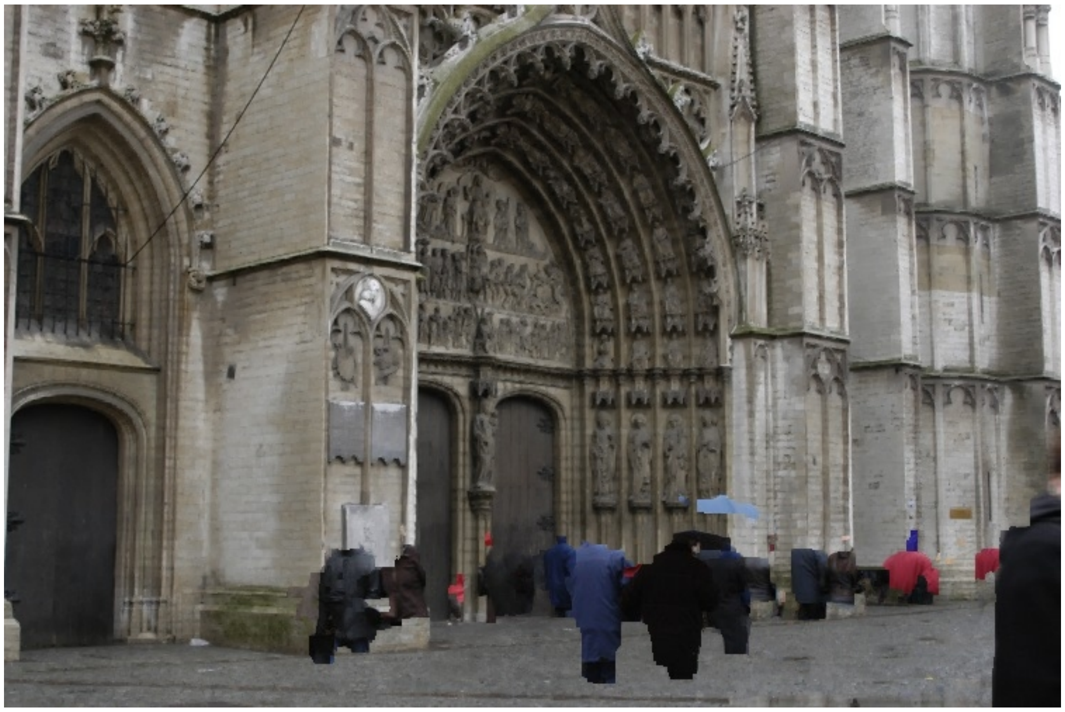


Figure 2.11: Background estimation result by applying the method proposed in [58].

Cohen [13] introduces a background estimation method which casts the problem into a labeling problem. It defines an energy function based on MRF formulation and uses graph cuts for energy minimization. Let $\{I_1, I_2, ..., I_N\}$ denote the given N input frames, $\mathcal{P}$ the set of pixels in an input image, and $I_m(p)$ the color value at pixel p in image m. $\mathcal{F}$ denotes the label space with $\mathcal{F} = \{1, 2, ..., N\}$, which are the indices of the input frames, and let f_p be a label of pixel p with $f_p \in \mathcal{F}$. Then this method constructs the background by copying the color at pixel p from input image I_p^*, where I_p^* is a minimum cost labeling.

The energy function $E(\mathcal{F})$ includes a data term and a smoothness term.

$$E(\mathcal{F}) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{p,q \in \mathcal{N}} V_{p,q}(f_p, f_q) \tag{2.21}$$

where $\mathcal{N}$ is the set of pair of neighboring pixels. $D_p(f_p)$ is the cost of assigning label f_p to pixel p, and accounts for color stationariness and motion boundary consistency. In particular,

$$D_p(f_p) = D_p^S(f_p) + \beta D_p^C(f_p) \tag{2.22}$$

The color stationariness cost is calculated based on the variance of the color $I_f(p)$ over several frames close to frame f_p. We use $Var_{f_1 \sim f_2}(p)$ to denote the average of the variances of color $I_f(p)$ from image f_1 to image f_2. Then,

$$D_p^S(f_p) = min\{Var_{f_p - r \sim f_p}(p), Var_{f_p \sim f_p + r}(p)\} \tag{2.23}$$

where r is a constant number. This stationary cost assigns high penalty to transient objects, which is consistent with the assumption that background objects are likely to appear more often than foreground objects in most but not all regions. The motion boundary consistency cost is defined as follows,

$$D_p^C(f_p) = \sum_{f=1}^{N} \frac{||\nabla M_{f_p f}(p)||_2^2}{||\nabla I_f(p)||_2^2 + \varepsilon^2} \tag{2.24}$$

where $\nabla I_f(p)$ is the intensity gradient and ε is a small constant number in case $\nabla I_f(p)$ is zero. The term $\nabla M_{f_p f}$ measures the motion gradient. Assume that f_p is the background image, then a difference image $\nabla M_{f_p f} = ||I_{f_p} - I_f||_2$ has a large gradient magnitude if I_{f_p} matches I_f poorly. The motion boundary consistency penalizes locations with large motion gradients but small intensity gradients.

In addition to the data term, a smoothness term is defined to assign a high cost to an area that contains a highly textured moving object, and a low cost to an untextured and still object. Specifically,

$$V_{p,q}(f_p, f_q) = \lambda \left[\frac{||I_{f_p}(p) - I_{f_q}(p)||_2^2 + ||I_{f_p}(q) - I_{f_p}(q)||_2^2}{2 \times \text{\# of color planes}} \right] \tag{2.25}$$

From the smoothness term, we can see that the cost is small if I_{f_p} and I_{f_q} match with each other well. The constants β in Equation 2.22 and λ in Equation 2.25 are used to trade off the importance of enforcing stationariness, consistency of motion boundaries and seamless cutting.

The experimental results provided in [13] show that their method performs better than median filtering. This method can produce correct background in some

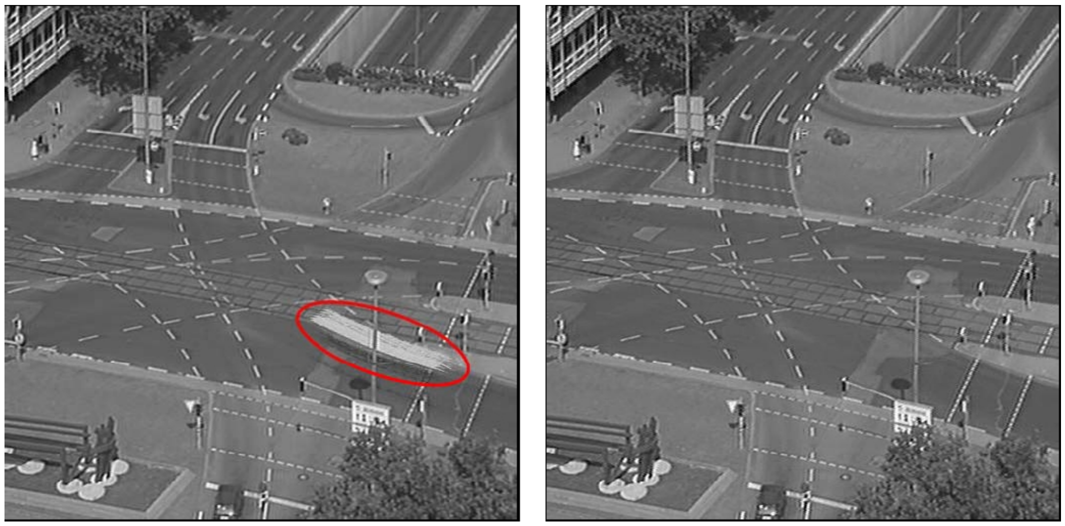


Figure 2.12: Background estimation results (from [13]). Left: result by median filtering. Right: result by [13].

critical regions where error exists with median filtering. We show a sample comparison of results provided by [13] and median filtering in Figure 2.12.

The figure shows that the method in [13] provides a clean and smooth background. However, the foreground objects do not occupy a large portion of the input frames in the dataset that are used, and the background is also relatively smooth. For comparison purposes, we re-implement this method by using the framework provided by [54]. Since this method uses graph cuts optimization for energy minimization, we use the graph cuts software of [10] [36] [9]. The details of both the framework and the graph cuts optimization method are discussed in Chapter 3. Figure 2.13 shows the result when applying this method to the dataset provided in [1]. In the figure, the error region is indicated by the red rectangle. In that region, the background only shows up once and the color of the foreground objects is uniform. Therefore, this method can still be improved to get better results.

Similar to [13], Granados et al. [28] propose a method to estimate background from non-time sequence images. The method defines a similar energy function and uses graph cuts for energy optimization. The energy function contains a data term $D_p(f_p)$, a smoothness term $V_{p,q}(f_p, f_q)$, and a hard constraint $H_{p,q}(f_p, f_q)$. That is,

$$E(f_p) = \sum_{p \in (P)} D_p + \sum_{(p,q) \in \mathcal{N}} V_{p,q}(f_p, f_q) + \sum_{(p,q) \in \mathcal{N}} H_{p,q}(f_p, f_q) \tag{2.26}$$

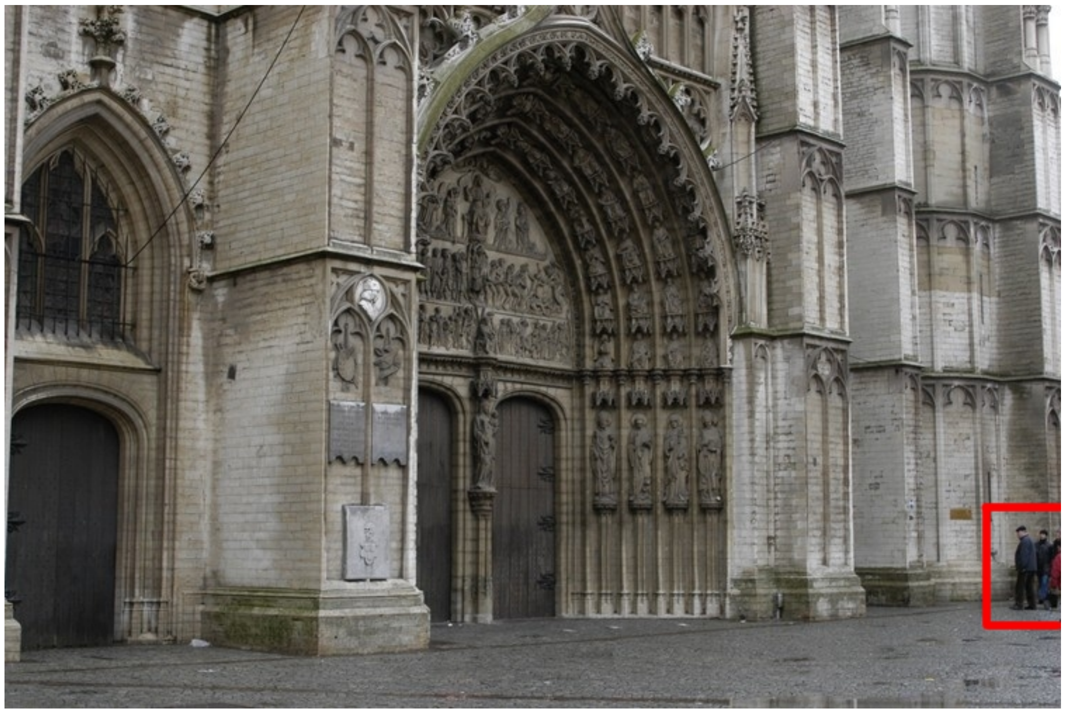


Figure 2.13: Background estimation by applying the method in [13].

The data term includes a likelihood term D^L and a stationariness term D^S.

$$D_p(f_p) = (1 - \beta(p))D_p^L(f_p) + \beta(p)D_p^S(f_p) \tag{2.27}$$

The parameter β controls the relative importance of the likelihood term and stationariness term. The stationariness term defined in this method is similar to the motion boundary consistency term in [13] (Equation 2.24). This term penalizes locations with large motion gradient but small intensity gradient because the boundary of moving objects occur at these locations. In particular, the stationariness term is defined as follows.

$$D_p^S(f_p) = \begin{cases} ||\nabla M_{f_p}(p)||_2 - ||\nabla \mathbf{I}(p)||_2 & \text{if } ||\nabla M_{f_p}(p)||_2 > ||\nabla \mathbf{I}(p)||_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

where $M_{f_p} = I_{f_p}(p) - \mathbf{I}$ is the difference with the average image of all input frames ($\mathbf{I}$). Compared to the original motion boundary consistency term, using the stationariness term has two benefits. First, the runtime complexity is lower. Second, it reduces the possibility of the occurrence of false motion boundaries caused by flat, textureless occluders.

Besides the stationariness term, the data term also includes a likelihood term.

$$D_p^L(f_p) = 1 - \prod_{c=1}^{3} \int_{I_{f_p}^c(p)-3\lambda_c}^{I_{f_p}^c(p)+3\lambda_c} \hat{d}_p^c(x)dx \tag{2.29}$$

where $\hat{d}_p^c$ is the estimated probability density function based on histogram with fixed intervals. c is the color channel and λ_c the expected variation on each color channel. This method applies Gaussian density estimators to compute $\hat{d}_p^c$, and λ_c is obtained experimentally from datasets with known ground truth.

The smoothness term is similar to the one used in [13]. Specifically,

$$V_{p,q}(f_p, f_q) = \frac{\gamma}{2}(||I_{f_p}(p) - I_{f_q}(p)||_2 + ||I_{f_p}(q) - I_{f_q}(q)||_2) \tag{2.30}$$

where γ is a parameter which determines the weight of this term. Different from [13], this method proposes a hard constraint in order to make sure that a transient object can be completely included in or removed from the background. Hence, a transient object in the scene cannot be split into two parts, which makes the background more realistic. The hard constraint is defined as follows:

$$H_{p,q}(f_p, f_q) = \begin{cases} 0 & \text{if } \min_{l \in \mathcal{L}}(||I_{f_p}(p) - I_l(p)|| + ||I_{f_q}(q) - I_l(q)||) < t_H \\ \infty & \text{otherwise} \end{cases} \tag{2.31}$$

where t_H is a threshold and it is set to be 5% of the intensity range in their experiments. The condition when $H_{p,q}(f_p, f_q)$ is assigned to be 0 is true when its color distance to the closest image l falls below the threshold t_H, in which case f_p and f_q are consistent with each other in color.

The method applies gradient domain fusion to composite the final background. The experimental results by applying this method are really promising. However, when defining the likelihood term, this method uses a parameter that is obtained experimentally from datasets with known ground truth, which makes it difficult to be re-implemented.

Motivated by the work in [1] and [13], we develop two novel methods for background estimation. Both methods cast this problem into a labeling problem. In our first method, we propose a cost function and apply a dynamic programming framework to minimize the cost. This method incorporates segmentation information in the cost function, which improves the quality of the final results. In our second method, we define an energy function based on MRF formulation and apply graph cuts optimization. The novelty of our second method lies in integrating image inpainting, which has never been done by any other researcher.

Chapter 3

Robust Background Estimation

In this chapter, two novel background estimation methods are presented. Then the experimental results follow. We also compare our results to the ones produced by [1] and by the automatic image stitching system [11].

3.1 Segmentation-Based Background Estimation

The input to our algorithm is a set of related images, which can be taken from the same viewpoint or from different viewpoints with overlapping regions among them. Our goal is to estimate the background from the image sequence. There are two basic assumptions for this algorithm:

(i) The background objects are stationary throughout the entire image sequence and each background pixel has to be exposed at least once.

(ii) The background objects are likely to appear more often than the transient ones.

These assumptions are commonly used in many background estimation algorithms [3] [28]. The first assumption implies that pixels of the same background object are always in the same position and every background pixel is visible in the input images. The second assumption provides a constraint that background objects should appear more frequently in most regions.

Suppose that the given image set has N input images. Since we assume that the images can be taken from different viewpoints, the pixels in every input image must be transformed to a global coordinate system at the beginning. We estimate the warping function for such transformation by the method proposed in [11], which is to compute a homography for each image with respect to the reference image. The left column in Figure 3.1 shows two input images taken from different viewpoints, and the right column the results after they are aligned to each other.

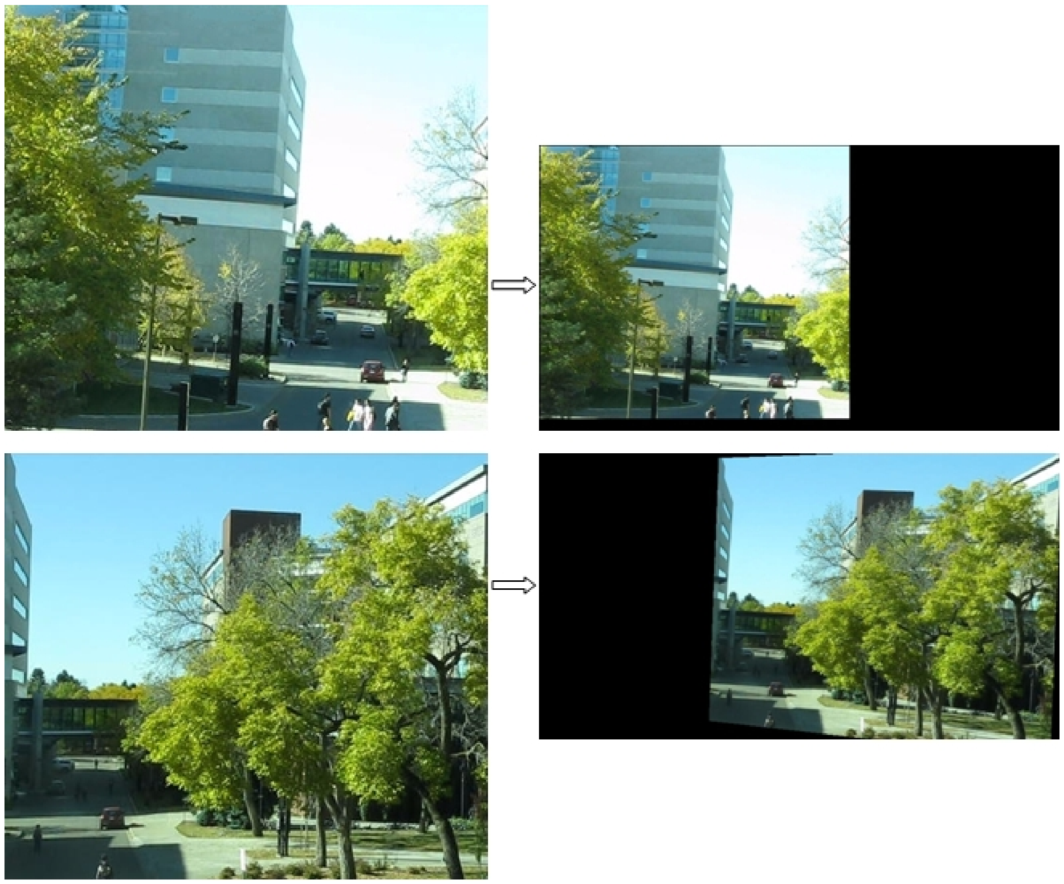


Figure 3.1: Image alignment. Left column: two input images taken from different viewpoints. Right column: aligned results for the two input images.

In the rest of this section, we assume that all the images have been projected into a global coordinate system. After the transformation, we denote the color value of a pixel in the m^{th} image at coordinates (x, y) as $I_m(x, y)$ where $1 \leq m \leq N$. The color of a pixel in the output background image is denoted as $O(x, y)$. Then the background estimation problem can be formulated as

$$O(x,y) = \sum_{m=1}^{N} \alpha_m(x,y) I_m(x,y) \tag{3.1}$$

In the above equation, $\alpha_m(x, y)$ is a binary selection function which is defined as

$$\alpha_m(x, y) = \begin{cases} 1 & \text{if } m \text{ is selected} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

That means, if $\alpha_m(x, y)$ is set properly for each pixel, then the background is recovered using Equations 3.1 and 3.2. The candidate pixels are selected from the input images to composite the background.

We observe that an output image without any foreground object achieves global visual smoothness. In other words, there exist significant differences when comparing an image with only the background and an image with both the foreground and the background. The former is visually smoother than the latter one. Therefore, if we have an appropriate cost function to measure smoothness, then the final result with only the background has the minimum cost.

The following is an overview of our method.

(1) Assign a cost to each pair of adjacent pixels according to the proposed cost function (Section 3.1.1), which includes a smoothness measure (Section 3.1.2) and a stationary coefficient (Section 3.1.3).

(2) Apply dynamic programming (DP) to minimize the total cost of each scanline (Section 3.1.4), and determine α_m for each pixel.

(3) Apply linear blending to reduce the seams in the estimated background (Section 3.1.5).

Figure 3.2 shows the flow chart corresponding to the above described overview.

3.1.1 Cost Function

The proposed cost function is shown in the following.

$$C_{ij}(x, Y) = \Big(S_x\Big(I_i(x, Y), I_j(x-1, Y)\Big) + \gamma \cdot S_y\Big(I_i(x, Y)\Big)\Big) \cdot \rho\Big(I_i(x, Y), I_j(x-1, Y)\Big) \tag{3.3}$$

where $C_{ij}(x, Y)$ is the cost assigned to a pair of adjacent pixels along one scanline $I_i(x, Y)$ and $I_j(x-1, Y)$, $1 \leq i, j \leq N$. $S_x\Big(I_i(x, Y), I_j(x-1, Y)\Big)$ is the smoothness measure along the x-direction, $S_y\Big(I_i(x, Y)\Big)$ along the y-direction, γ is a coefficient assigned to balance the weight of the smoothness measure along two dimensions, and $\rho\Big(I_i(x, Y), I_j(x-1, Y)\Big)$ the stationary coefficient. In this method, γ is set to be 1 for most datasets.

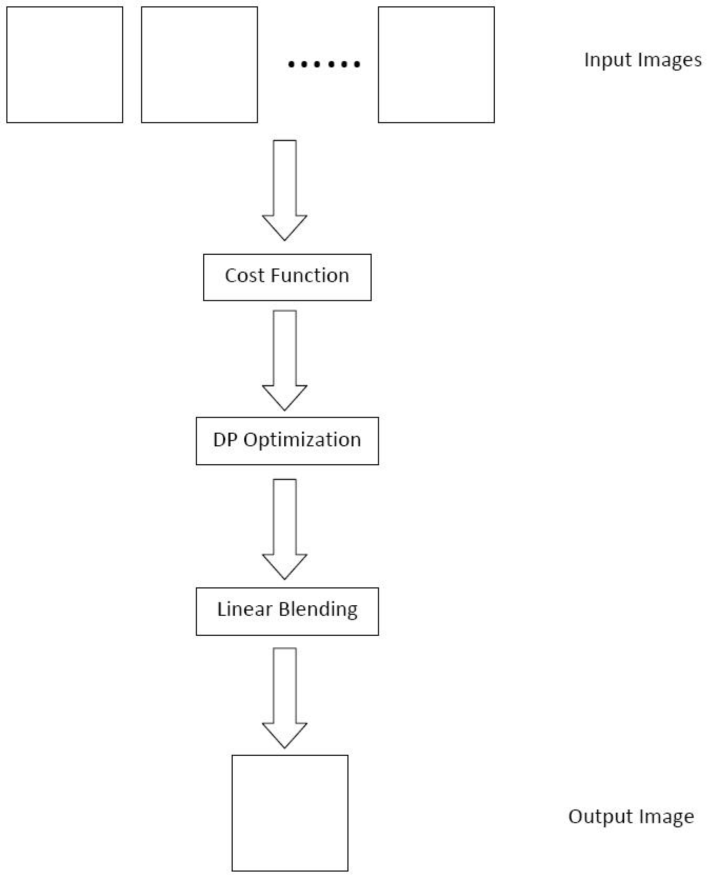


Figure 3.2: Flow chart of our method.

3.1.2 Smoothness Measure

The smoothness measure is defined to measure how well a pair of adjacent pixels satisfy the visual smoothness constraint. We measure the smoothness in two directions. In this section, we first introduce the smoothness measure along the x-direction and then the y-direction. The definition of our smoothness measure along the x-direction is given as follows.

$$S_x\Big(I_i(x,Y), I_j(x-1,Y)\Big) = \begin{cases} \beta \cdot \sum_{k=r,g,b} |I_i^k(x,Y) - I_j^k(x-1,Y)| & \text{if } I_i(x,Y) \in R \text{ and } I_j(x-1,Y) \in R \\ \sum_{k=r,g,b} |I_i^k(x,Y) - I_j^k(x-1,Y)| & \text{otherwise} \end{cases} \tag{3.4}$$

where R denotes a segment, and r, g, b represent the red, green and blue channels, respectively. It measures the similarity between two neighboring pixels $I_i(x, Y)$ and $I_j(x-1, Y)$ along the same scanline. The parameter β is set to be 0.01 for all the datasets, and the reason of setting it with this particular value is discussed below.

In order to assign a more reasonable cost to each pair of adjacent pixels, the information of image segmentation is integrated into the smoothness measure. Although any segmentation algorithm could be used, we apply a graph-based image segmentation algorithm [23] in this method. The segmentation algorithm can preserve detail in smooth regions while ignore detail in cluttered regions. The algorithm is briefly described as follows. Suppose the input image for this algorithm is I. First, I is represented by a graph $G = (V, E)$ with vertices $v \in V$, and the weight of each edge $e \in E$ is

$$w(v_p, v_q) = \sum_{i=r,g,b} |I^i(p) - I^i(q)|, \quad v_p, v_q \in V \tag{3.5}$$

where $I(p)$ denotes the intensity value of a certain pixel in the image, and v_p, v_q are the two adjacent nodes connected by the edge e. Suppose that v_p and v_q belong to two separated segment R_1 and R_2, that is, $v_p \in R_1$ and $v_q \in R_2$. In order to determine whether or not R_1 and R_2 should be merged, the algorithm assigns a pairwise comparison predicate $P(R_1, R_2)$.

$$P(R_1, R_2) = \begin{cases} true & \text{if } Dif(R_1, R_2) > MInt(R_1, R_2) \\ false & \text{otherwise} \end{cases} \tag{3.6}$$

$$Dif(R_1, R_2) = \min_{v_p \in R_1, v_q \in R_2, (v_p, v_q) \in E} w(v_p, v_q) \tag{3.7}$$

$$MInt(R_1, R_2) = \min\Big(Int(R_1) + \tau(R_1), Int(R_2) + \tau(R_2)\Big) \tag{3.8}$$

$$Int(R) = \max_{e \in MST(R,E)} w(e) \tag{3.9}$$

$$\tau(R) = \frac{k}{|R|} \tag{3.10}$$

where $|R|$ is the size of segment R and k is defined by the user. If $P(R_1, R_2)$ is true, then R_1, R_2 should be separated, otherwise they are merged into a bigger segment. Algorithm 2 describes this image segmentation algorithm step-by-step. The input to this algorithm is a graph $G = (V, E)$ with n vertices and m edges, and the output of this algorithm is a partition of V into regions $\mathbf{R} = (R_1, R_2, ..., R_k)$.

Algorithm 2 Image Segmentation Algorithm

```
➢ Sort $E$ into $\pi = (e_1, e_2, ..., e_m)$ by non-decreasing edge weight.
➢ Repeat the next step for $q = 1, ..., m$.
➢ Construct $\mathbf{R}^q$ given $\mathbf{R}^{q-1}$ as follows. Let $e_q = (v_i, v_j)$ be the $qth$ edge connecting the vertices $v_i$ and $v_j$, $R_i^{q-1}$ be the region of $\mathbf{R}^{q-1}$ containing $v_i$ and $R_j^{q-1}$ be the region containing $v_j$. If $R_i^{q-1} \neq R_j^{q-1}$ and $w(e_q) \leq MInt(R_i^{q-1}, R_j^{q-1})$, then $\mathbf{R}^q$ is obtained by merging $R_i^{q-1}$ and $R_j^{q-1}$, otherwise $\mathbf{R}^q = \mathbf{R}^{q-1}$.
➢ Return $\mathbf{R} = \mathbf{R}^m$.
```

In Equation 3.4, the penalty is relatively small if two adjacent pixels belong to the same segment R. There are two reasons to take advantage of the segmentation result. First, by construction, a segment is a group of adjacent pixels with similar colors. Figure 3.3 shows an image with its segmentation result. The pixels which belong to the white board are now in the same segment after applying image segmentation. Second, integrating information from the result of image segmentation can prevent frequent switching among the input images when compositing the final background. According to Equation 3.1, each pixel in the background image is selected from one of the input images. When image segmentation is applied, pixels in the same segment from the same image will be selected more favorably. Without this bias, adjacent pixels could be selected from different images, which will create undesirable seams. Additionally, using segmentation results makes the estimated background less sensitive to illumination change that appears among input images by biasing to select pixels from the same segment. Hence, in Equation 3.4, we set β to be 0.01 to reflect a low penalty when a pair of adjacent pixels are in the same segment.

The measurement described above encourages the result to be visually smooth along the x-direction. Along the y-direction, we use the information from the previous scanline to measure the smoothness. In particular, it is defined as follows.

$$S_y\Big(I_i(x_i, Y)\Big) = \sum_{k=r,g,b} |I_i^k(x, Y) - I_f^k(x, Y-1)| \tag{3.11}$$

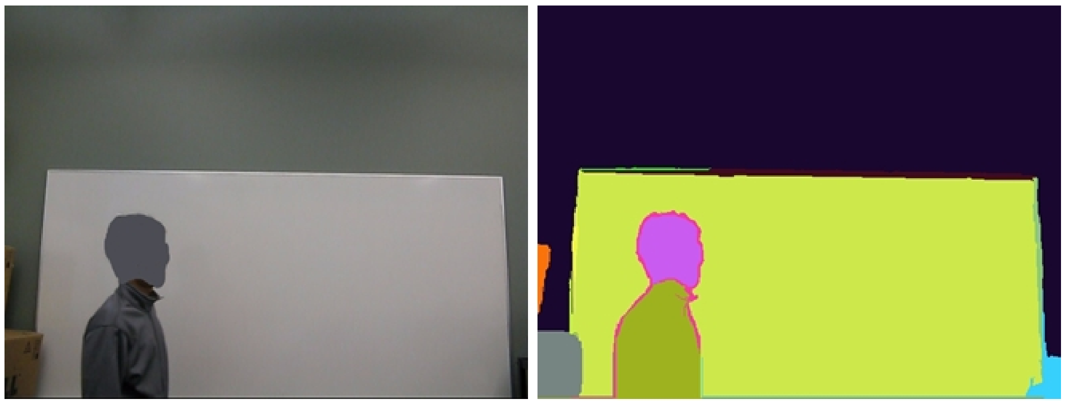


Figure 3.3: Left: Input image. Right: Segmentation result by applying the method proposed in [23]

where i denotes the i^{th} input image. It is defined based on the information from the previous scanline that has just been processed, which is the $(Y-1)^{th}$ scanline in Equation 3.11. For the first scanline of the image, the smoothness term is initialized to 0 since there is no previous scanline to it. On the $(Y-1)^{th}$ scanline, f is selected to minimize the aggregated cost. The cost minimization is discussed in Section 3.1.4.

3.1.3 Stationary Coefficient

We propose a stationary coefficient to satisfy the second assumption, that is, background objects are more likely to appear than transient ones. The coefficient ρ is defined to be:

$$\rho\Big(I_i(x,Y), I_j(x-1,Y)\Big) = \Big(1-\frac{NUM(I_i(x,Y))}{N}\Big)\cdot\Big(1-\frac{NUM(I_j(x-1,Y))}{N}\Big) \tag{3.12}$$

where N is the number of input images and $1 \leq i,j \leq N$. $NUM(I_i(x,Y))$ denotes the number of pixels from $\{I_1(x,Y),...,I_i(x,Y),...,I_N(x,Y)\}$ that are similar to $I_i(x,Y)$. In our experiments, $I_m(x,y)$ is defined to be similar to $I_i(x,y)$ if $|I_m(x,y)-I_i(x,y)| < 0.1\cdot I_i(x,y)$ for all three color channels, where $1 \leq m \leq N$. The stationary coefficient implies that if two adjacent pixels appear frequently, then the corresponding assigned cost is small.

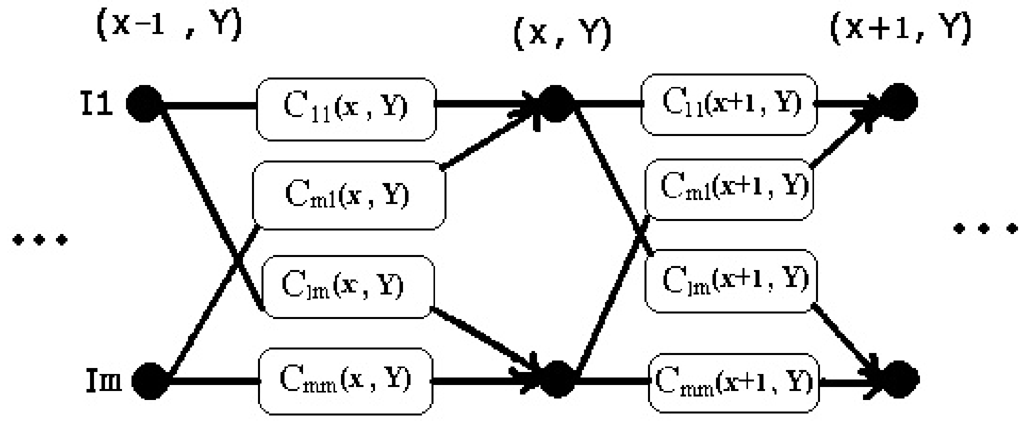


Figure 3.4: Cost assignment.

3.1.4 Cost Minimization

Figure 3.4 shows how the cost is assigned to two neighboring pixels. To simplify illustrations, only two input images are shown. To extend to more images is straightforward. The figure shows only the Y^{th} scanline. In this figure, I_1 represents image 1, $C_{1m}(x+1, Y)$ is the proposed cost for selecting $I_1(x, Y)$ and $I_m(x+1, Y)$.

After a cost is assigned to every pair of adjacent pixels on a scanline, DP is applied to minimize the aggregated cost for each scanline. The formulation is similar to the one introduced in [15] and is defined as follows.

$$E_Y = \min_m \Big(E_Y(L, m)\Big) \tag{3.13}$$

$$E_Y(x, m) = \begin{cases} 0 & \text{if } x = 1 \\ \min\Big(E_Y(x-1, 1) + C_{1m}(x, Y), ..., & \\ \quad E_Y(x-1, m) + C_{mm}(x, Y), ..., & \\ \quad E_Y(x-1, N) + C_{Nm}(x, Y)\Big) & \text{if } x > 1 \end{cases} \tag{3.14}$$

where $1 \leq m \leq N$. L is the length of a scanline and $1 \leq x \leq L$. DP is applied to one scanline at a time. In Equation 3.13, E_Y is the minimum cost of the Y^{th} scanline. In Equation 3.14, $E_Y(x, m)$ denotes the aggregated cost of the m^{th} row, the x^{th} column on the Y^{th} scanline. C_{1m} denotes the cost assigned to the adjacent pixels $I_1(x, Y)$ and $I_m(x-1, Y)$, which is the same as that shown in Figure 3.4. The minimum cost is calculated after going through the whole scanline. During backtracking, a pixel at each x coordinate along a scanline is selected from one of the input images. The selected input image is indexed as f, which is the same as the one used in Equation 3.11, and then the corresponding α_f is set to be 1.

3.1.5 Implementation

In this method, obvious seams could exist in the output image because different regions are selected from different input images. The difference in illumination among input images is the main cause of the artifact. Therefore, blending is needed to suppress this problem.

Linear blending is applied to removed the artifact in the result. In order to combine information from multiple input images, a weighting function $\theta(x, y) = w(x)w(y)$ is assigned to each image, where $w(x)$ varies linearly from 1 at the center of the image to 0 at the edge. Then a weighted sum of the image intensities is computed using:

$$O^{linear}(x, y) = \frac{\sum_{m=1}^{N} I_m(x, y)\theta_m(x, y)}{\sum_{m=1}^{N} \theta_m(x, y)} \tag{3.15}$$

where $1 \leq m \leq N$ and $|I_m(x, y) - I_f(x, y)| < t_H$, $I_f(x, y)$ is introduced in Equation 3.11 and t_H is a threshold specified by the user according to the illumination condition. Typically t_H is set to be $0.2 \cdot I_f(x, y)$ for all three color channels. It should be larger when the illumination change is large among input images.

3.1.6 Parameter t_H

We notice that there is a parameter t_H in the first method when linear blending is applied to the datasets taken from different viewpoints. Therefore, the following is an alternative method to eliminate this parameter.

After each input image is transformed to a global coordinate system, all the possible combination of images are calculated using linear blending. For example, if there are three input images $\{I_1, I_2, I_3\}$, then there are seven possible combination of the images when applying linear blending, which are $\{I_1\}$, $\{I_2\}$, $\{I_3\}$, $\{I_1, I_2\}$,

$\{I_1, I_3\}$, $\{I_2, I_3\}$, $\{I_1, I_2, I_3\}$. As a result, there are $2^3 - 1 = 7$ blended images. The Equation 3.15 is applied for linear blending and there is no constraint on $I_m(x, y)$. Figure 3.5 shows the linear blending result. The top row are two input images and the bottom row is the blended image.

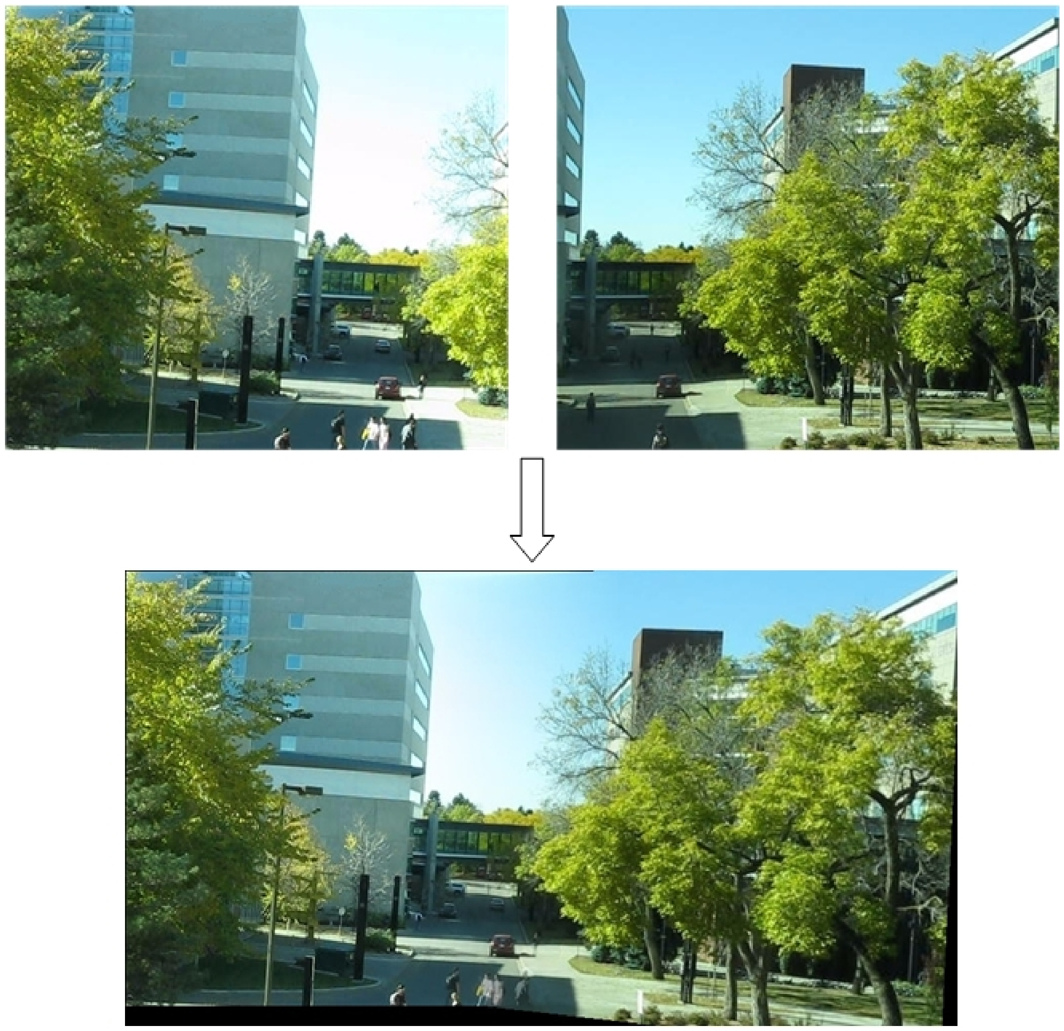


Figure 3.5: Linear blending.

When all the blended images are calculated, they are used as the input images, then the cost is assigned to each pair of adjacent pixels and DP is applied for cost minimization. Figure 3.6 is the flow chart of this alternative method. Compared to the flow chart shown in Figure 3.2, the difference is that linear blending is performed in the first step of this alternative method. As a result, the parameter t_H can be eliminated.

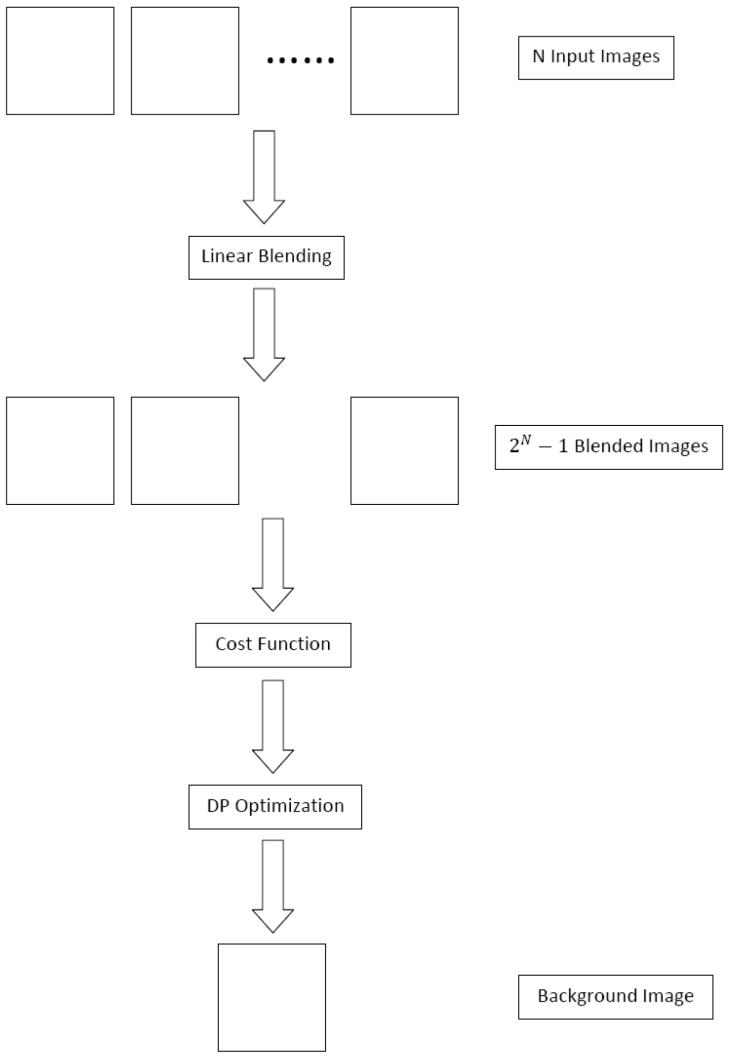


Figure 3.6: Flow chart of our alternative method.

In Figures 3.9 and 3.10, we demonstrate that this alternative method can get results similar to our original method for some datasets. However, there are differences between these two methods. The advantage of this alternative method is that it has no parameter for linear blending. One of its drawback is that it is slower. Suppose we have N input images, then our original method takes them as input images. But the alternative method applies linear blending to all of them, which produces $2^N - 1$ images, which are used as input images. Generally, the alternative method is not suitable for generating background from a large number of input images. The second drawback is that it is not practical when there are many transient objects in the input images. The reason is that applying linear blending lowers the probability of the background pixels to be visible. The problem is illustrated in Figure 3.7. The top row in Figure 3.7 are three synthesized input images. Assume that the white part is the background and the color blocks are the foreground objects. In the center of each input image, there is a black dot, and it indicates that the background pixel shows up twice in that location among these three input images. The result of linear blending is shown in the bottom row. We can see that the background pixel at the same location shows up three times, which is in the second, fourth and sixth image if indexing from left to right. In other words, linear blending lowers the probability of this particular background pixel from 2/3 to 3/7.

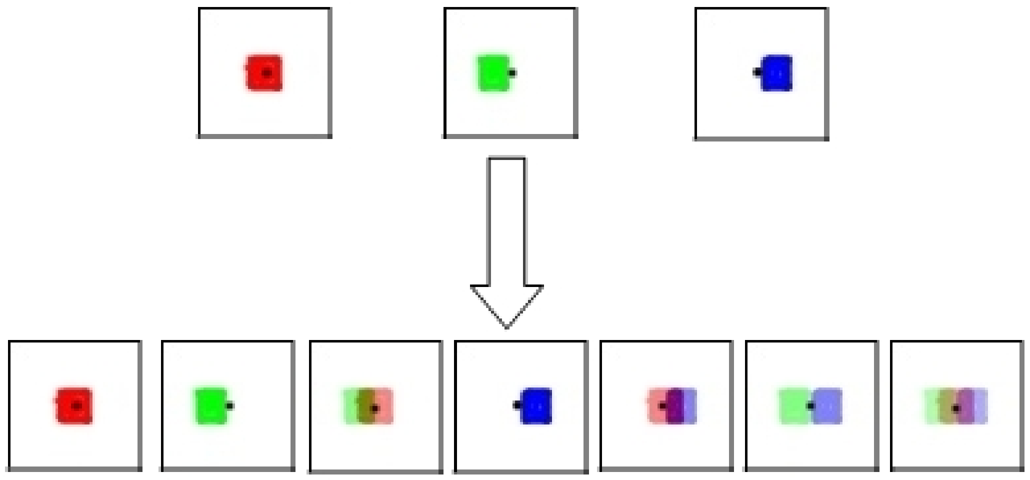


Figure 3.7: An example to illustrate how linear blending lower the probability of the background pixels. Top row: the input images. Bottom row: the linear blending results.

3.1.7 Experimental Results

In our experiments, we generate several datasets and they are divided into different categories. Most of the input images are captured using a digital camera on a tripod. Some datasets generated by other researchers are also used for comparison.

Figure 3.8 shows some results when we apply our method to four datasets with different features, the comparison between our results and the ones produced by [1] is also given. The input images in the first row have a smooth background and smooth foreground. In this dataset, there is a region where the background appears

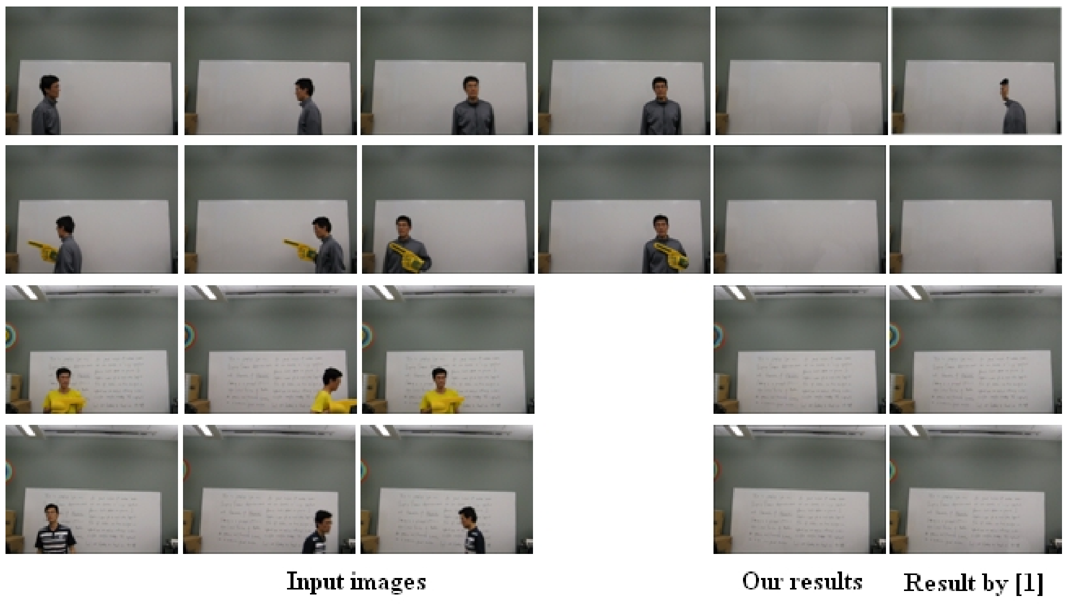


Figure 3.8: Background estimation from different datasets with various features.

only once. Our method can estimate the background correctly. However, there are error regions in the result using the method in [1]. The background of the input in the second row is smooth, but the foreground is not. The background regions in the input images of the third row and the bottom row are the same. But one has a smooth foreground and the other has a complex foreground. In this figure, we demonstrate that our method can produce good results under various background/foreground conditions.

Figure 3.9 shows the results when applying our method to four input images which are taken from different viewpoints, with cars and people as foreground objects. The background and foreground are both complex in this dataset. Figure 3.9(b) shows our result and Figure 3.9(c) the result after applying the alternative method. In this dataset, we show that our method can provide correct results even if the illumination changes significantly among input images. Figure 3.9(d) shows the panoramic image generated by using the automatic image stitching system [11]. The red arrow and the blue arrow each point at the error regions. There are artifacts because there is no background estimation in [11]. Figure 3.9(e) shows the result using the background estimation method proposed in [1]. We can see that there are still some transient objects in this image which are indicated by the arrows.

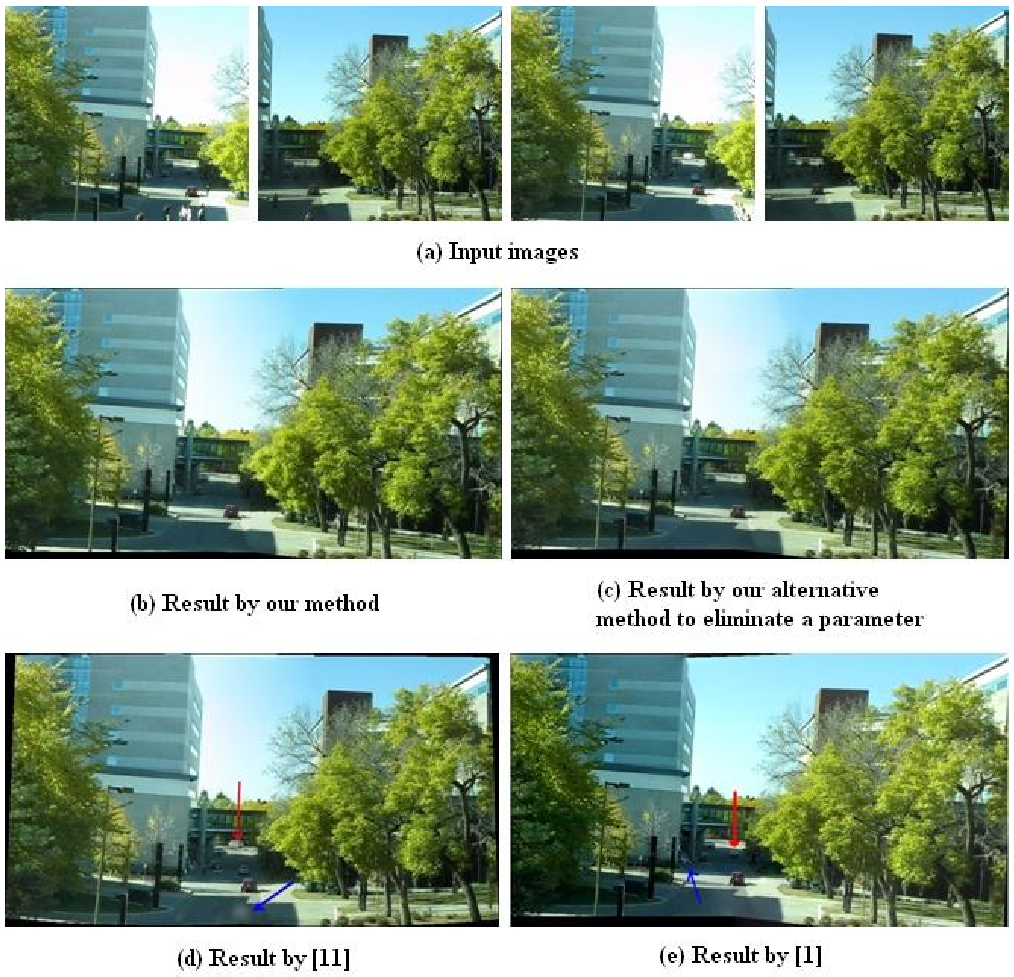


Figure 3.9: (a) Four input images from different viewpoints, with transient objects. (b) Estimated background by our proposed method. (c) Estimated background by the alternative method. (d) Result produced by the image stitching system [11]. (e) Result by applying [1].

We demonstrate the importance of integrating segmentation information in Figure 3.10. There are four input images taken by a moving camera. The window frames are considered to be the foreground objects because they are moving relative to the camera, and the far away buildings are the background since their movements relatively to the camera can be ignored. Figure 3.10(b-c) are the results by applying our method and the alternative method. For comparison, we give the result produced by [1]. We also demonstrate the importance of incorporating segmentation information in this dataset. Figure 3.10(e) is the result by changing our smoothness measure along the x-direction to be without using any segmentation information.

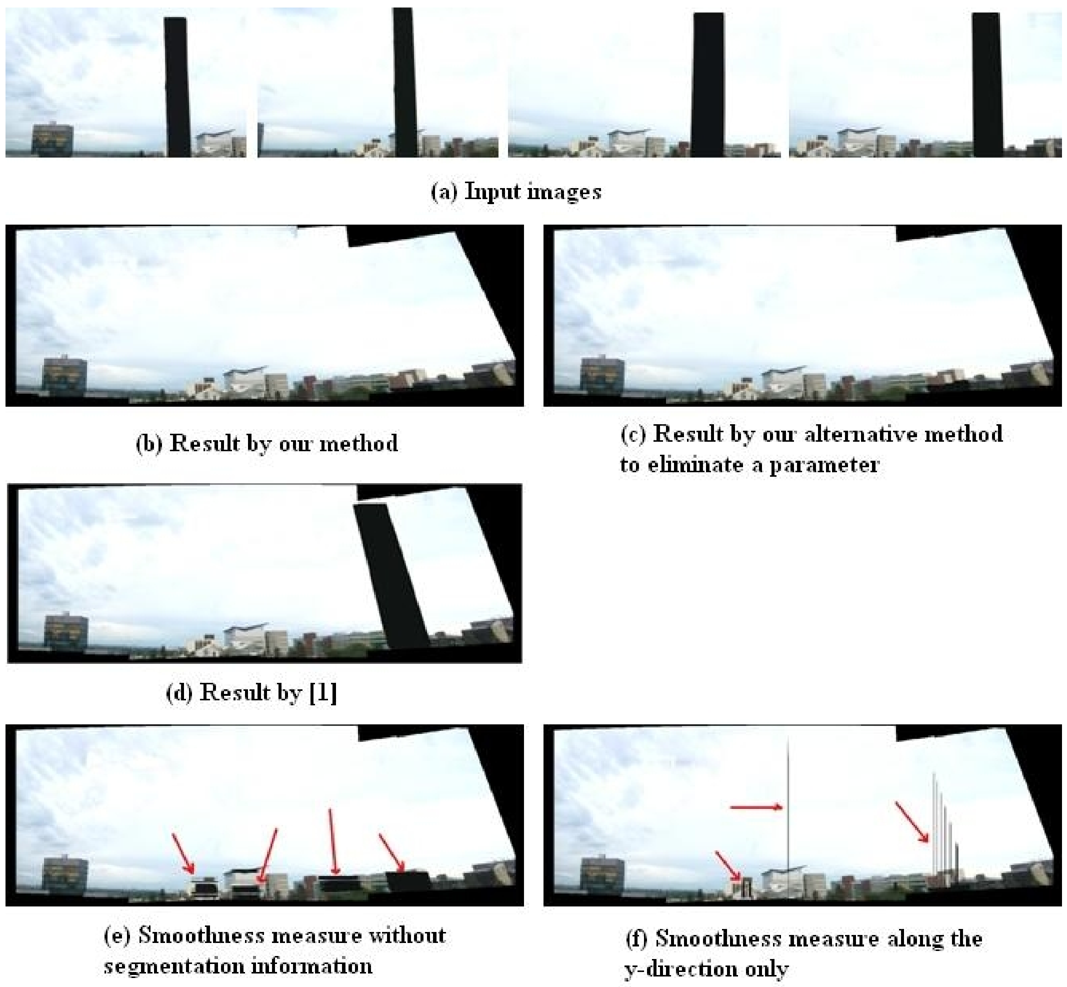

(a) Input images
(b) Result by our method
(c) Result by our alternative method to eliminate a parameter
(d) Result by [1]
(e) Smoothness measure without segmentation information
(f) Smoothness measure along the y-direction only

Figure 3.10: (a) Multiple view images with window frames as foreground objects. (b) Estimated background by our method. (c) Estimated background by our alternative method to eliminate the threshold. (d) Estimated background by applying [1]. (e) Result when using the smoothness measure without segmentation information. (f) Result with smoothness measure along y-direction only.

That is, the cost between two neighboring pixels is simply the sum of the absolute differences of their colors. Figure 3.10(f) shows the result with the smoothness measure along the y-dimension only. The red arrows in Figures 3.10(e) and 3.10(f) point to the regions that are wrong. Figure 3.10 shows that our algorithm can create a large field of view with transient foreground objects removed.

In Figure 3.11, several images with a fence as the foreground object are given as the input images to our method. Figure 3.11(b-f) show the results with different number of input images. We can see that with the number of input images increases, the fence gradually disappears in the result. Figure 3.11(g) shows the central part of the result by using the whole image sequence. We can see that the fence is removed in the central part of the composite image. We compare our result with the result by applying the methods proposed in [1] and [11].

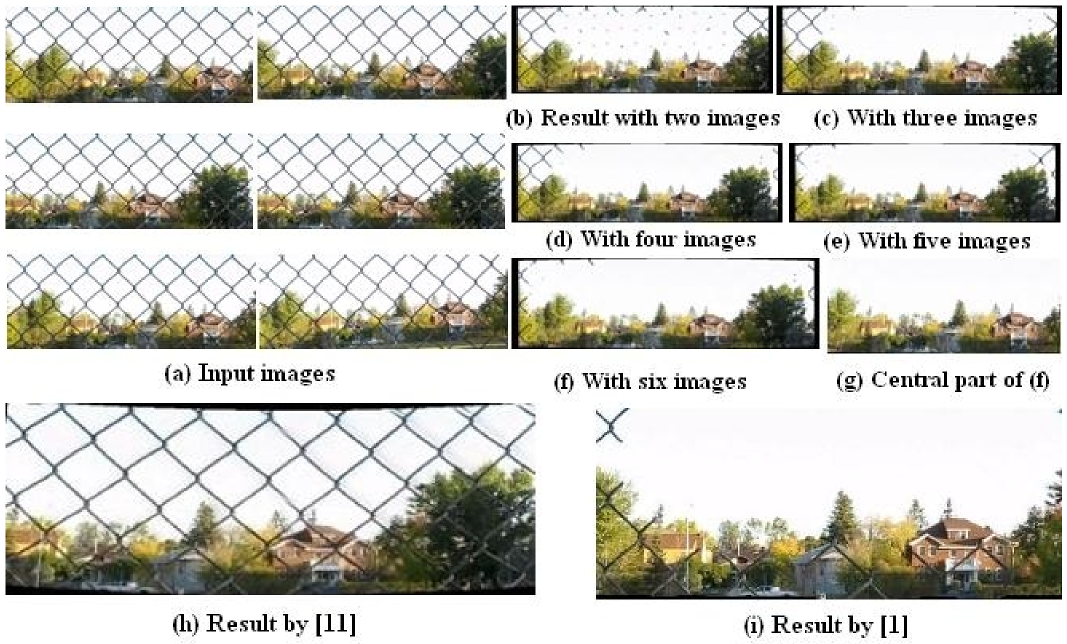


Figure 3.11: Multiple image de-fencing. (a) Six input images with a fence as foreground object. (b-f) Results with different number of input images. (g) The central part of (f). (h) Applying [11] to the six images. (i) Applying [1] to the six images.

Next, we apply our method to some datasets that are generated by other researchers. Figure 3.12(a) is the input image sequence from the *Cathedral* scene which is originally used in [1]. Figure 3.12(b) is the estimated background by applying our method. The red arrow points at the error region in our result. Figure 3.12(c) shows the result produced by [1]. The yellow rectangle region encloses undesirable foreground objects.

Figure 3.13 shows the result of our alternative method. As we discussed in Section 3.1.6, when the scene has many foreground objects such as the one shown in Figure 3.12(a), linear blending makes the background being covered more times, and that is the reason why there are error regions in Figure 3.13. Hence, this alternative method is not practical when the input images have many transient objects in the scene.

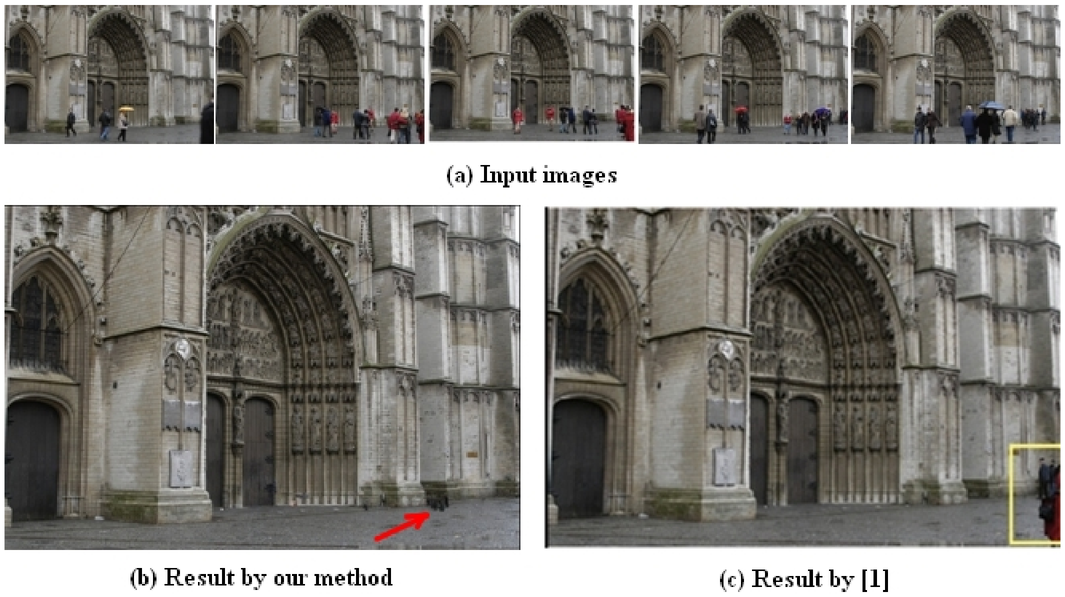


Figure 3.12: (a) Five input images from *Cathedral* scene. (b) Estimated background by our method. (c) Estimated background by [1].

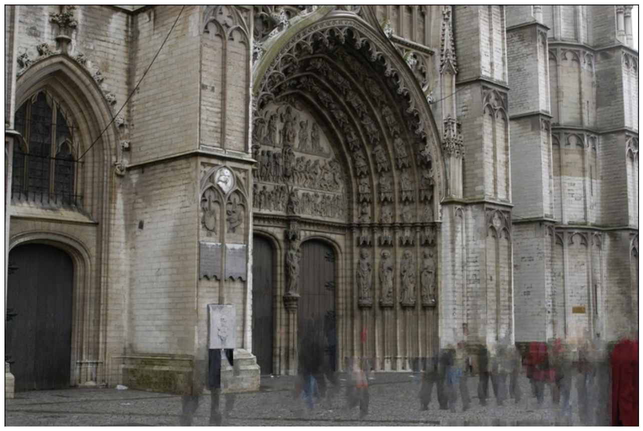


Figure 3.13: Result by applying our alternative method to the dataset shown in Figure 3.12(a).

Figure 3.14(a) shows a subset of 21 images which are originally used in [3]. Figure 3.14(b) is the estimated background by our method, and Figure 3.14(c) is the result by applying the method in [1]. Since the scene is complex, we can see that there are error regions in both our result and the result produced by [1].

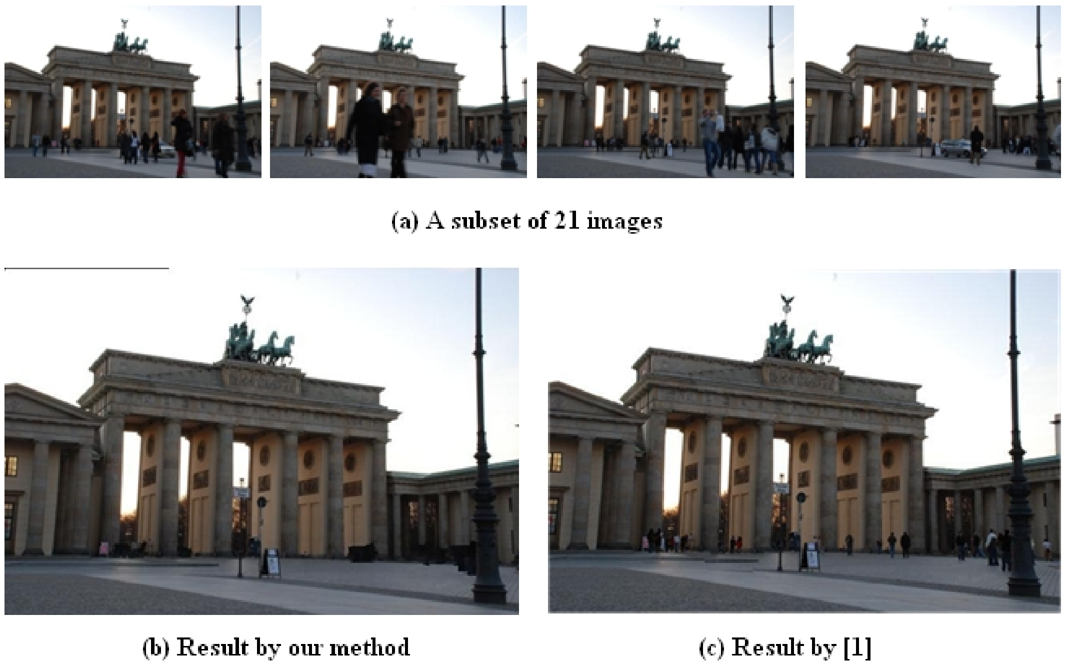


Figure 3.14: (a) A subset of 21 images from [3]. (b) Estimated background by our method. (c) Estimated background by [1].

3.1.8 Application

In this final experiment, we apply our background estimation method to some high-quality video sequences with their depth maps provided by the Microsoft Research Group [61]. There are two video sequences provided online, the “Breakdancing” sequence and the “Ballet” sequence. There are eight different views for each sequence, and each sequence contains 100 color images with 100 corresponding depth maps.

We apply our method to the depth maps and their corresponding color images simultaneously. For each sequence, we only show results of images from two viewpoints. Figure 3.15(a) shows a color image and its corresponding depth map. The whole image sequence contains 100 color images and 100 depth maps. Figure 3.15(b) shows the estimated background by applying our method. In this video sequence, the four people standing in the far end have slight movements. Therefore, the result is blurred in those regions due to linear blending. Figure 3.15(c) shows the background image which is used in [37]. Figures 3.15(b) and (c) clearly show that our result is smoother, and has less noise, especially on the floor. We believe that there could be significant improvements to the results in [37] if our background estimation results were used. Figure 3.16 shows the result by applying our method to another image sequence from a different viewpoint. The top row shows a color image and its corresponding depth map, and the bottom row the estimated background for both color images and depth maps. The input images shown in Figures 3.15 and 3.16 are taken from the “Breakdancing” sequence. In Figure 3.17, the images on the top row and the bottom row are from different viewpoints, and they are all from the “Ballet” sequence. The two columns on the left side show the input images and the other two columns on the right side show the estimated background by applying our method. In this dataset, the people standing on the right-hand side is moving slightly. Therefore, the result is blurred in that particular region.

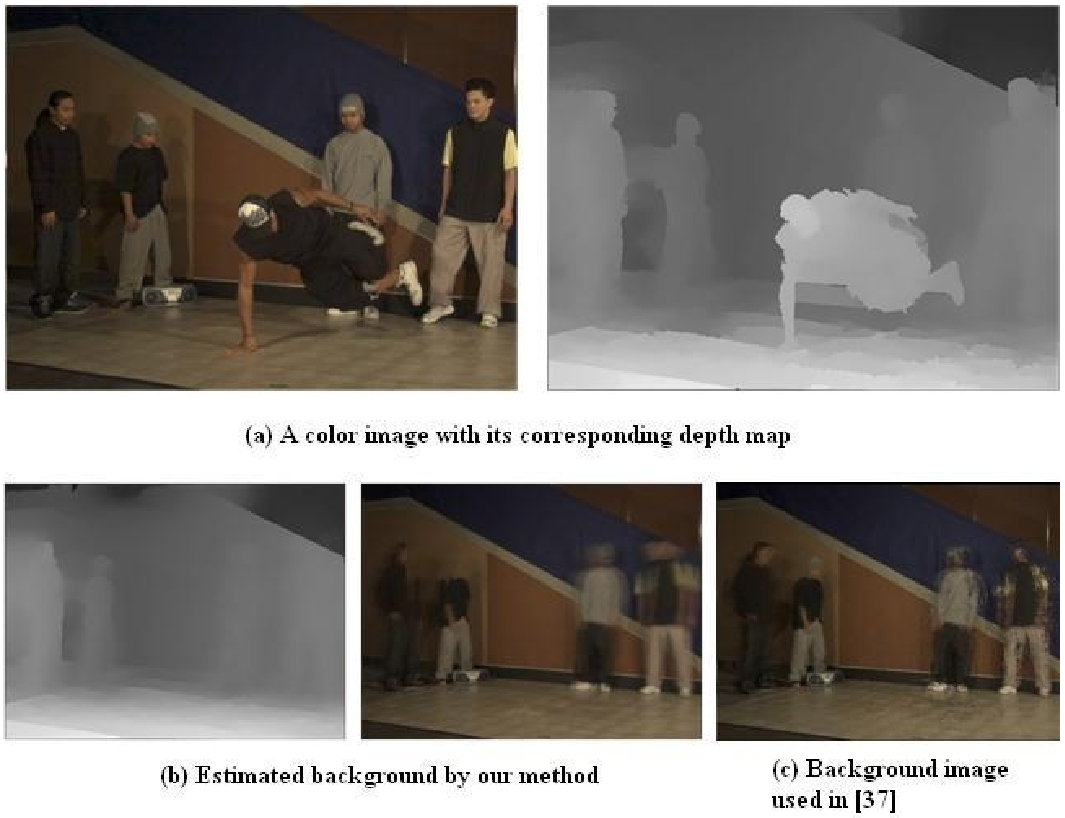

(a) A color image with its corresponding depth map
(b) Estimated background by our method
(c) Background image used in [37]

Figure 3.15: (a) A color image and its corresponding depth map. (b) Estimated background by our method. (c) The background image used in [37]

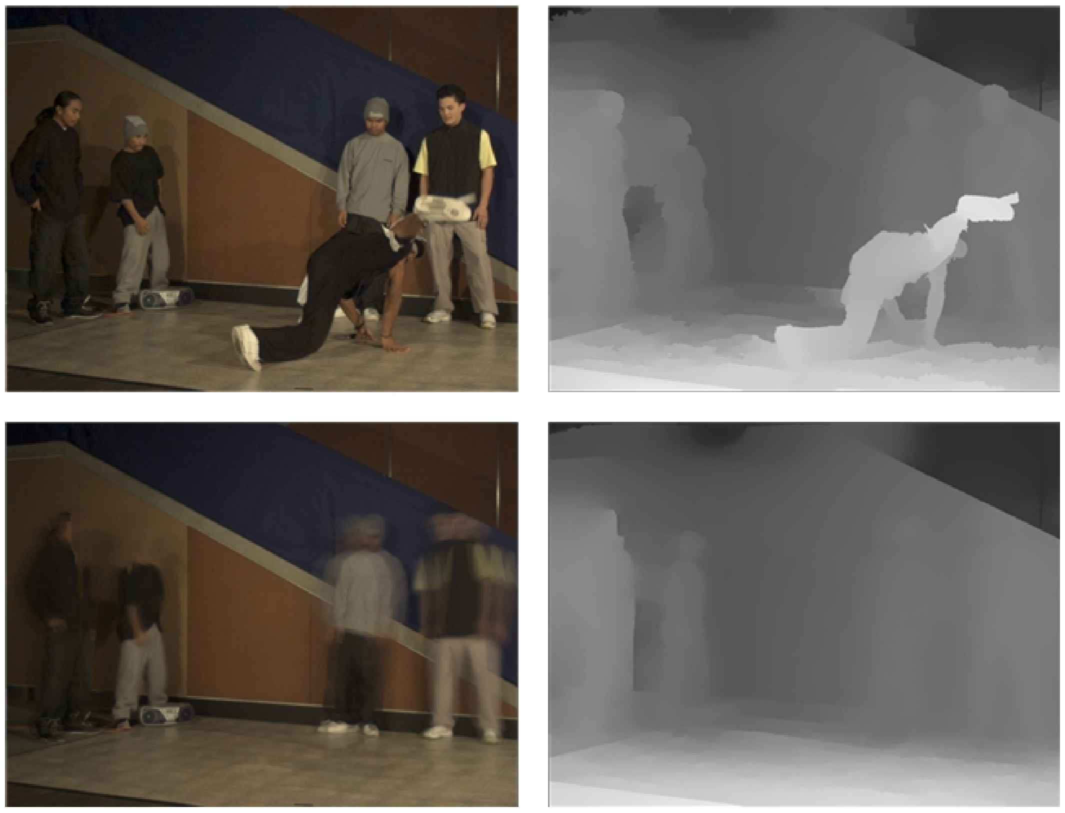


Figure 3.16: Top row: A color image and its corresponding depth map. The viewpoint is different from the ones in Figure 3.15. Bottom row: Estimated background.

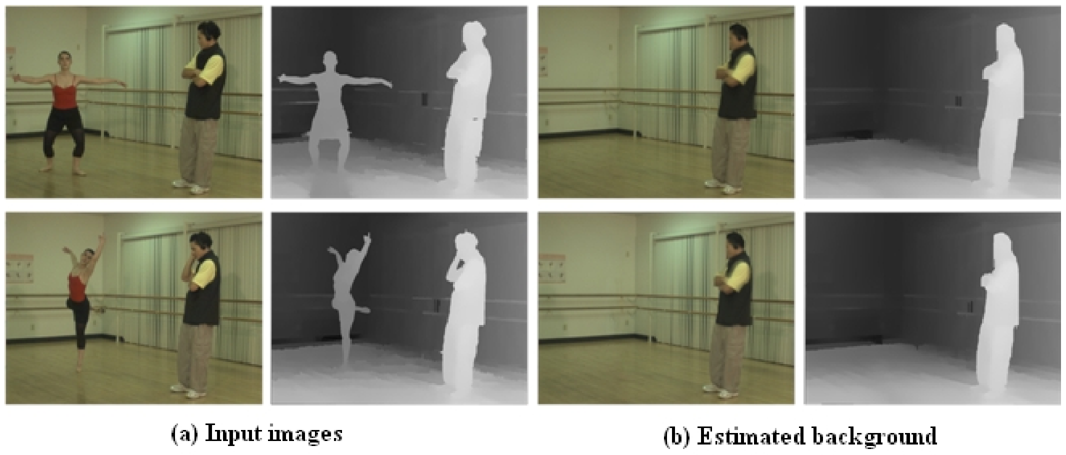


Figure 3.17: Apply our method to the "Ballet" sequence. (a) The color images with their corresponding depth maps. (b) Estimated background by our method.

3.2 Sampling-Based Background Estimation

3.2.1 Motivation

The background estimation results produced by the method described above are promising. However, errors still exist in some particular datasets such as the one shown in Figure 3.12. Also, our alternative method in Section 3.1.6 is not applicable when the scene has many transient objects (Figure 3.13).

As pointed out by Cohen [13], background estimation can be regarded as a labeling problem. That is, each pixel in the background image is copied from one of the input images. The pixel-labeling problem is represented in terms of energy minimization. Szeliski et al. [54] compare the solution quality and runtime of several commonly used energy minimization algorithms: graph cuts, loopy belief propagation and tree-reweighted message passing, in addition to an older iterated condition mode algorithm.

There are two graph cuts algorithms introduced in [10], namely, the swap move algorithm and the expansion move algorithm. Both algorithms repeatedly compute the global minimum of a binary labeling problem in their inner loop. This process converges quickly and results in a strong local minimum. According to the results provided in [54], the expansion algorithm produces better results than the other energy optimization methods in general.

Motivated by the work in [58] [13] [1] [28], we consider background estimation as a labeling problem as well. The information from multiple input images is combined to form an output image. In this method, we propose a cost function and use graph cuts to minimize the cost. The novelty of this method lies in a new predicted term which is used to predict the color value in the regions where the background is not visible. Our method incorporates a simple image inpainting technique and assigns higher costs to the labels that are more different from the predicated result. Figure 3.18 is the flow chart of this method. In the final step, gradient domain fusion is applied to remove any visible seam only if there are significant illumination changes among input images.

3.2.2 Energy Function

Similar to our previous methods, we assume that the input images have been transformed to a global coordinate if they are taken from different viewpoints, and our goal is to estimate the background. That is, for each pixel, we need to find an input image in which the background is visible so that we can copy this pixel to construct

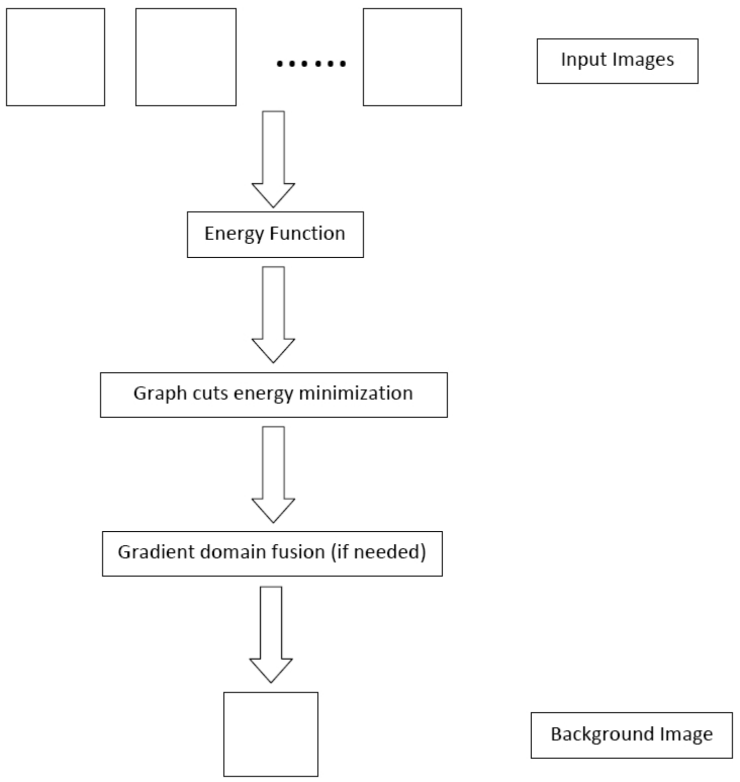

Input Images
Energy Function
Graph cuts energy minimization
Gradient domain fusion (if needed)
Background Image

Figure 3.18: The flow chart of this method.

the background. Each pixel is assigned a label with a frame number and a cost is assigned to each possible labeling. Our method obtains a labeling for all the pixels which can minimize the cost.

Formally, suppose we are given a set of N input images $\{I_1, I_2, ... , I_N\}$, and let $\mathcal{P}$ be the set of pixels in an image. Let $I_m(p)$ be the color value at position p of the m^{th} image. We denote $\mathcal{L}$ as the label space with $\mathcal{L} = \{1, 2, ..., N\}$, representing the image index. Let f_p be a label of pixel p and $f_p \in \mathcal{L}$. A labeling f is to assign a particular label f_p to pixel $p \in \mathcal{P}$, and a corresponding cost is assigned to this labeling based on the energy function that is defined below. With the above definitions, our problem is to find a labeling f^* to construct the background I_B = I_{f^*}, such that the labeling f^* has the minimum cost.

We define our energy function based on the MRF (Markov Random Field) formulation:

$$E(f_p) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q) \tag{3.16}$$

where $D_p(f_p)$ denotes the data term which defines the cost of assigning label f_p to pixel p. $V_{p,q}(f_p, f_q)$ denotes the smoothness term that defines the cost of assigning labels f_p and f_q to the pair of neighboring pixels p and q. In the above equation, $\mathcal{N}$ is the set of neighboring pixels in $\mathcal{P}$. The details of both terms are discussed in the following sections.

3.2.3 Data Term

Our data term consists of two parts which are the stationary term D^S and the predicted term D^{PR}. It is defined as follows.

$$D_p(f_p) = D_p^S(f_p) + D_p^{PR}(f_p) \tag{3.17}$$

The stationary term is defined based on the observation that in most part of the image, the background shows up more times than the foreground, although the background may appear only once in some particular region in the whole image sequence. As a result, our stationary cost is defined based on the color similarity at p of all the input frames. In particular,

$$D_p^S(f_p) = \sum_{i=1}^{N} |I_{f_p}(p) - I_i(p)| \tag{3.18}$$

In the following sections, we use the term $|I_i(p) - I_j(p)|$ to represent the sum of the absolute color differences between $I_i(p)$ and $I_j(p)$ of the R, G and B channels. As an example of the stationary cost, if the same background pixel p appears N times in the input images, then the cost is zero.

There is a predicted term in our energy function which applies an image inpainting technique. Image inpainting is to recover the lost or corrupted part of the image data. Figure 3.19 is an example of image inpainting. The left is an image with scratch, and the middle is a mask. The mask indicates that the red regions should be filled in. In order to fill in the corrupted part, a typical image inpainting method [6] uses neighborhood information of the red regions. Then the image on the right side is the result.

We apply a simple image inpainting technique which is described as follows. If a certain region is stable, which means that the background pixels show up N

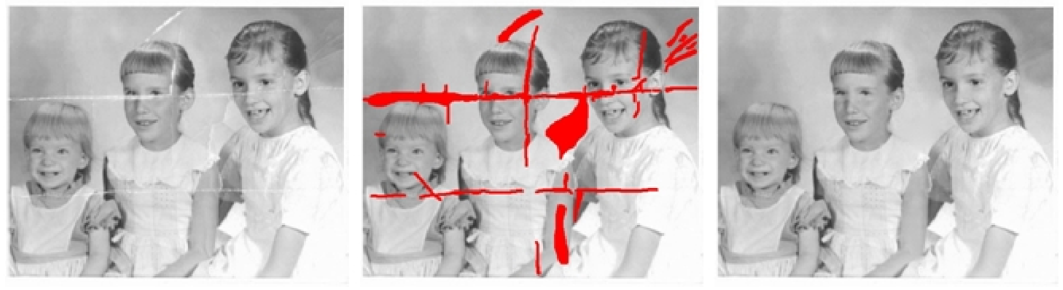


Figure 3.19: An image inpainting example.

times in that region, then the region is retained in the final image. In other words, the predicted cost is assigned to the regions that contain occlusions (unstable regions). Therefore, the stable regions in the input image sequence is detected first. Since some of the background regions are never occluded, there always exists useful information from stable regions. To detect stable regions, we set a threshold and examine the color value for $p \in \mathcal{P}$ to get the stable vector $\overline{S}$. Specifically,

$$\overline{S}(p) = \begin{cases} k & \text{if } |I_i(p) - I_k(p)| < t_H, \text{ for all } i = 1, ..., N \text{ but } i \neq k \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

where $k \in [1, N]$ and t_H is a threshold. For each color channel, the threshold is set to be either 15 if $I_k(p) = 0$ or $0.2 \cdot I_k(p)$ otherwise. Once we obtain the stable vector, we can construct the stable image I_{ST} for the entire image sequence. That is,

$$I_{ST}(p) = \begin{cases} I_{\overline{S}(p)}(p) & \text{if } \overline{S}(p) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

Figure 3.20 shows the stable image of the input images which are used in Figure 3.12. The black regions in I_{ST} represent the unstable regions. As described above, the stationary term takes advantage of temporal information. The predicted term which is discussed below uses spatial information from the stable regions. Our predicted cost is defined for the unstable pixels only. That is,

$$D_p^{PR}(f_p) = \begin{cases} 0 & \text{if } \overline{S}(p) > 0 \\ |I_{f_p}(p) - I_{PR}(p)| & \text{otherwise} \end{cases} \tag{3.21}$$

where $I_{PR}(p)$ is the predicted color value for the unstable pixel p in I_{ST} by applying an image inpainting technique. By defining the predicted term in this way, it means that if the color of the candidate pixel I_{f_p} is close to the predicted value $I_{PR}(p)$, then the predicted cost is low.

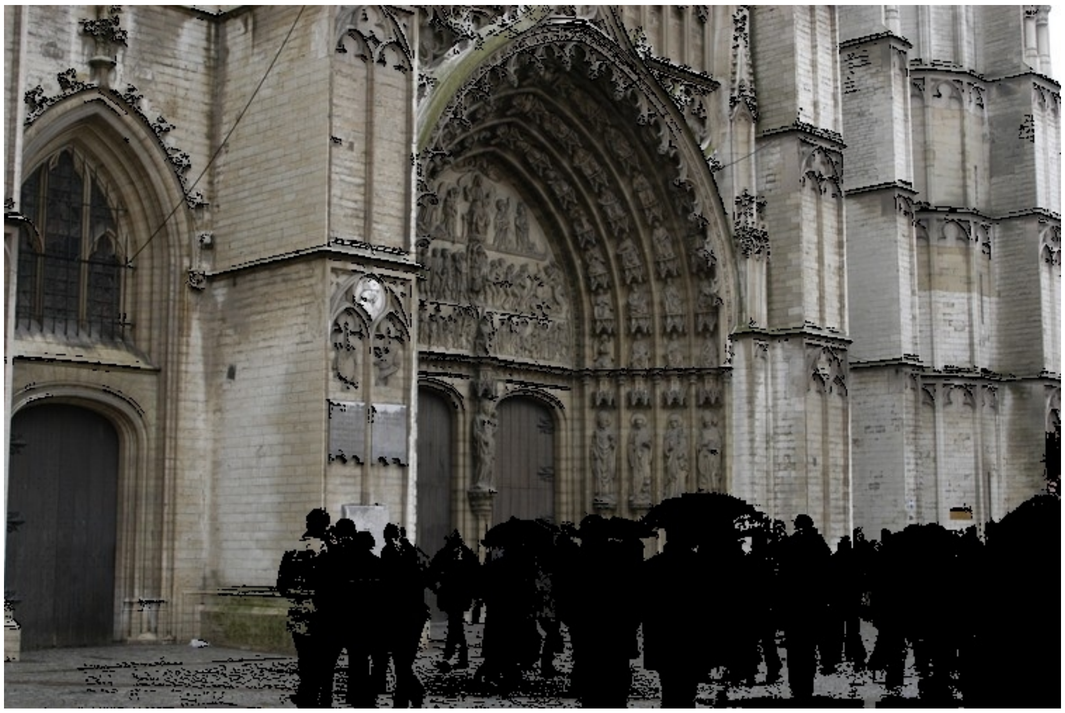


Figure 3.20: The stable image I_{ST} for the input image sequence shown in Figure 3.12.

The image inpainting technique that is applied is based on local and global color sampling. The neighborhood information is used for local color sampling. For each unstable pixel p in I_{ST}, we define a window centered at p and apply bilinear interpolation inside the window to compute the predicted color value based on stable pixels. Therefore, the predicted value by local color sampling is the same for all labels $f_p \in \mathcal{L}$. Only the stable pixels inside the window are used because they are reliable background pixels. However, the unstable regions could be huge sometimes, which reduces the number of stable pixels inside the window. In this case, we apply global color sampling as follows. For the candidate pixel $I_{f_p}(p)$, bilinear interpolation is applied to all the pixels in I_{ST} that have similar color values with $I_{f_p}(p)$. Since the color value $I_{f_p}(p)$ varies for different input images, the predicted color value for p by global sampling varies as well. The global sampling relies on the observation that a background pixel is likely to appear more than once at different locations. Formally, the predicted color value is defined as follows:

$$I_{PR}(p) = \frac{\sum\limits_{p' \in \mathcal{S}} I_{ST}(p') \cdot \left(1 - \frac{|p - p'|}{|WI|}\right)}{\sum\limits_{p' \in \mathcal{S}} \left(1 - \frac{|p - p'|}{|WI|}\right)} \tag{3.22}$$

where $\mathcal{S}$ denotes the set of sampled pixels defined below. The definition of $|WI|$ and $\mathcal{S}$ are given in the following equations.

$$|WI| = \begin{cases} |W| & \text{if } \frac{\text{\# of stable pixels in } W}{\text{\# of pixels in } W} > 0.5\% \\ |I_{ST}| & \text{otherwise} \end{cases} \tag{3.23}$$

$$\mathcal{S} = \begin{cases} \{p' | p' \in W, \overline{S}(p') > 0\} & \text{if } |WI| = |W| \\ \{p'' | p'' \in I_{ST}, \overline{S}(p'') > 0, I_{ST}(p'') \simeq I_{f_p}(p)\} & \text{otherwise} \end{cases} \tag{3.24}$$

In the above equations, W is the window centered at p with the default size of 100 $\times$ 100 pixels and $|W|$ denotes the window size. The experimental results that justify this particular window size is shown in the Appendix C. $|I_{ST}|$ is the size of image I_{ST} and $|p - p'|$ denotes the distance between p and p'. $I_{ST}(p'') \simeq I_{f_p}(p)$ means that $|I_{ST}(p'') - I_{f_p}(p)| < t_H$ for each of the R, G and B channels.

The pseudocode shown in Algorithm 3 illustrates the steps to calculate the predicted value for an unstable pixel p. And in Figure 3.21, we show the image after all the unstable regions in Figure 3.20 are filled in by our proposed local and global sampling method.

3.2.4 Smoothness Term

In order to achieve visual smoothness in the final output image, we adopt the smoothness term which is proposed in [13]. That is,

$$V_{p,q}(f_p, f_q) = \frac{||I_{f_p}(p) - I_{f_q}(p)||_2 + ||I_{f_p}(q) - I_{f_q}(q)||_2}{2} \tag{3.25}$$

The smoothness term gives a high cost if f_p and f_q are not matched with each other well. However, if background regions are available in both f_p and f_q, then they are ideal locations to switch copying from one frame to the other. As illustrated in [13], the smoothness cost can be high in an area containing a moving textured

Algorithm 3 Predicted term computation for an unstable pixel p

```
$\mathcal{S} = \emptyset$
Examine the window $W$ centered at $p$
if $\frac{\text{\# of stable pixels in } W}{\text{\# of pixels in } W} > 0.5\%$ then
    $|WI| = |W|$
    for each pixel $p^{'} \in W$ do
        if $\overline{S}(p^{'}) > 0$ then
            $\mathcal{S} = \mathcal{S} \cup \{p^{'}\}$
        end if
    end for
else
    $|WI| = |I_{ST}|$
    for each pixel $p^{''} \in I_{ST}$ do
        if $\overline{S}(p^{''}) > 0$ and $I_{ST}(p^{''}) \simeq I_{f_p}(p)$ then
            $\mathcal{S} = \mathcal{S} \cup \{p^{''}\}$
        end if
    end for
end if
Apply Equation 3.22 to compute $I_{PR}(p)$
```

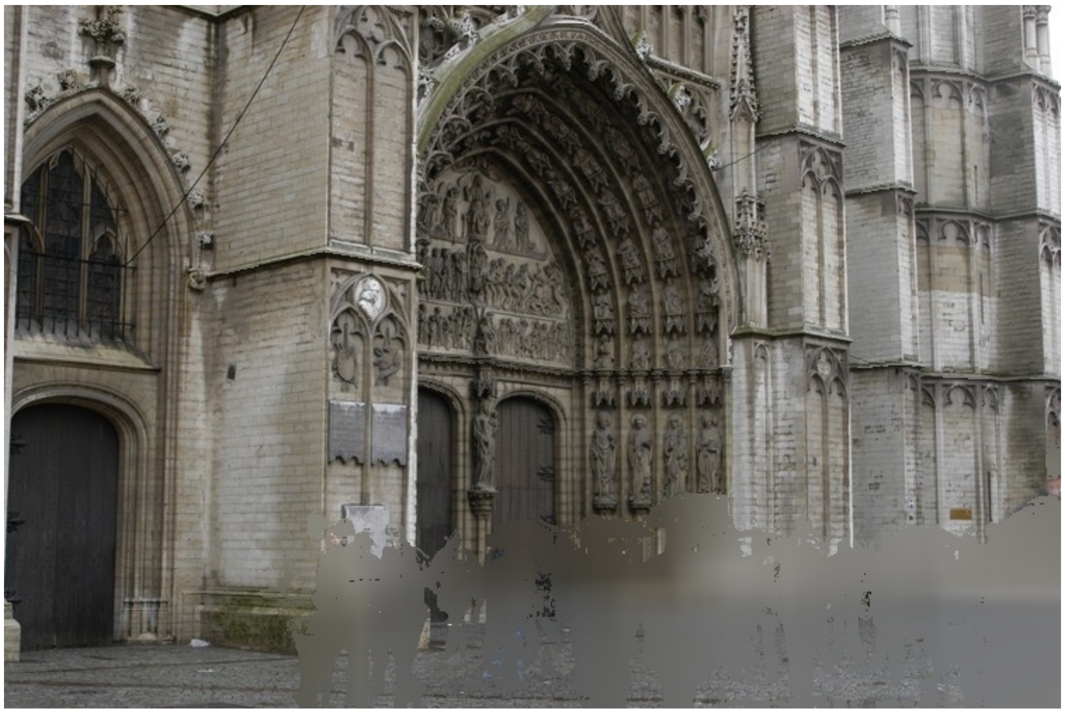


Figure 3.21: Result when the unstable regions in Figure 3.20 are filled in by predicted values.

object. By incorporating such a smoothness term, gradient domain fusion is applied after the energy minimization only when the images are taken from different viewpoints.

3.2.5 Implementation

We minimize the energy E using the framework that is publicly available [54]. This framework is developed under Windows XP using C++ programming language. In particular, we apply the expansion move algorithm because it produces better results than other energy minimization methods in general. The implementation of this algorithm uses the max-flow algorithm which is originally implemented by Boykov et al. [9]. The expansion move algorithm is specially designed for the graphs in vision applications. Shown in [9], this algorithm performs particularly well for those graphs.

When applying the global color sampling in the image inpainting technique, our method searches the entire stable image I_{ST} in order to find similar pixels for bilinear interpolation. However, this process can be extremely time-consuming. When the size of the input images is big such as 1024×768, and if the unstable regions are large in I_{ST}, then there will be many pixels inside the unstable regions that require global color sampling. In order to reduce the processing time, we use histogram for acceleration. The histogram of the color of stable image I_{ST} is constructed first. Then we search the pixels which are similar to the candidate pixel $I_{f_p}(p)$ in the histogram. Since we only search for the similar pixels in the histogram, the information of the distance between the candidate pixel and its similar pixels is lost. As a result, averaging instead of bilinear interpolation is applied to similar pixels. And finally the average value is used as the predicted color value by global color sampling. Formally, the equations to compute the predicted value by using histogram is shown below.

$$I_{PR}(p) = \frac{\sum\limits_{p'' \in \mathcal{S}} I_{ST}(p'')}{\text{\# of pixels in } \mathcal{S}} \tag{3.26}$$

$$\mathcal{S} = \{p'' | p'' \in I_{ST}, \overline{S}(p'') > 0, I_{ST}(p'') \simeq I_{f_p}(p)\} \tag{3.27}$$

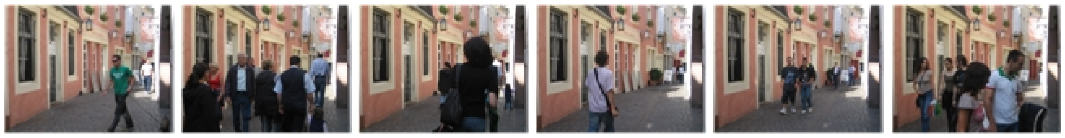
Specifically, instead of searching pixels p'' that have similar value to I_{f_p} in I_{ST}, p'' are obtained from the histogram of I_{ST}. Although the process is slightly different from bilinear interpolation, we found that the results produced by both methods are very similar to each other. However, using histogram acceleration reduces the processing time significantly. The results produced by both approaches and the improvement in speed is discussed in the following section.

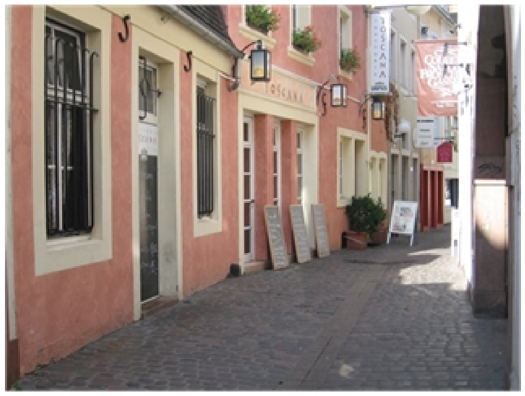
3.2.6 Experimental Results

In our experiments, we apply the proposed method to many datasets to demonstrate that our method is robust. The program is running on a desktop with 2 dual-core 2.2GHz AMD Opteron Processors with 4GB of memory.

We first apply the method to two datasets with natural scenes used in [28]. The first dataset is the *Toscana* scene shown in Figure 3.22(a). It has six input images each with size 800×600 pixels. There are many background regions occluded in the input sequence. It takes around 300 seconds to compute the background for this dataset by using histogram for acceleration. However, it takes more than

15 minutes without this acceleration approach. Figure 3.22(b) shows the result of our method. We can see that all the occluders have been removed from the scene, leaving only the background objects in the final result. For comparison, we apply the background reconstruction method proposed in [1], and the result is shown in Figure 3.22(c). The red rectangle in the image highlights the error region.


(a) Input images


(b) Estimated background by our method

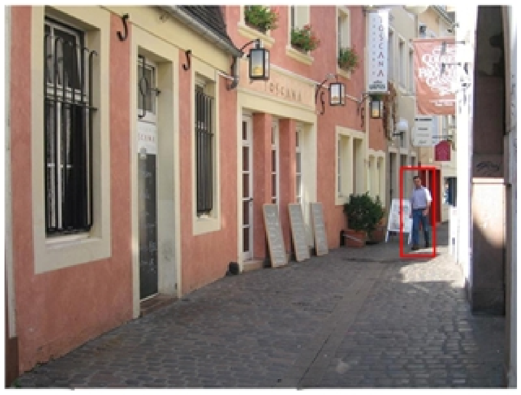
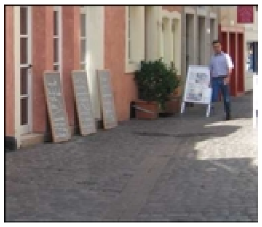
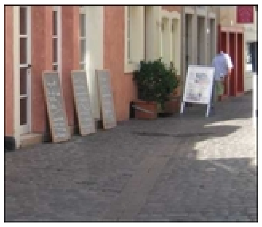
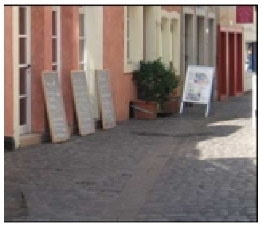

(c) Estimated background by [1]

Figure 3.22: (a) A natural image sequence with six input images. (b) Estimated background by applying our method. (c) Estimated background by applying the method proposed in [1].


(a) $D^s + V$


(b) $D^s + V + D^L$


(c) $D^s + V + D^{GL}$

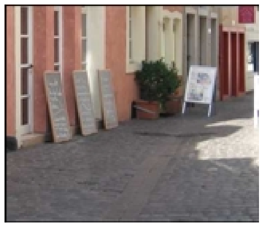

(d) $D^s + V + D^L + D^{GL}$

Figure 3.23: (a) Result with the stationary term and the smoothness term. (b) Result with the stationary term, the smoothness term and the predicted term only by local color sampling. (c) Result with the stationary term, the smoothness term and the predicted term only by global color sampling. (d) Result with the stationary term, the smoothness term and the predicted term by both local and global color sampling.

In order to show the importance of our predicted term, the results of applying different combinations of terms to this dataset are shown Figure 3.23. Figure 3.23(a) shows the result without the predicted term. Figures 3.23(b) and (c) show the results with the predicted term computed by local color sampling or global color sampling, respectively. And finally we show the result with our proposed predicted term in Figure 3.23(d). We demonstrate that the result is significantly improved with our predicted term.

Figure 3.24(a) shows another natural scene with eight input images which is named the *Market* scene. Figure 3.24(b) is the result by applying our method and Figure 3.24(c) is the result by applying the background reconstruction method proposed in [1]. The foreground object inside the red rectangle region shown in Figure 3.24(c) should not exist. Also, the sky region in our result appears more realistic than the one by [1].

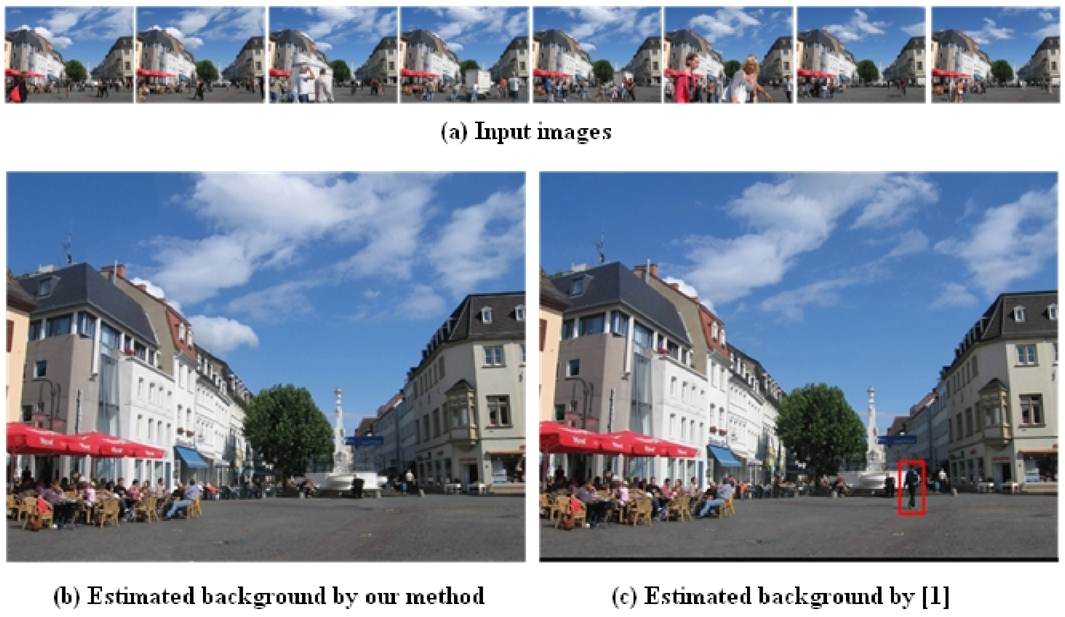

(a) Input images
(b) Estimated background by our method
(c) Estimated background by [1]

Figure 3.24: (a) Eight input images of the *Market* scene. (b) Estimated background by applying our method. (c) Estimated background by applying the method proposed in [1].

In our implementation of this method, we use histogram in order to reduce the processing time. Specifically, we perform averaging instead of linear interpolation on the pixels which are similar to the candidate pixel. Figure 3.25(a) shows the result without the acceleration approach and Figure 3.25(b) the result with it. The comparison indicates that the result without acceleration is slightly better. However, applying histogram acceleration can significantly reduce the processing time.

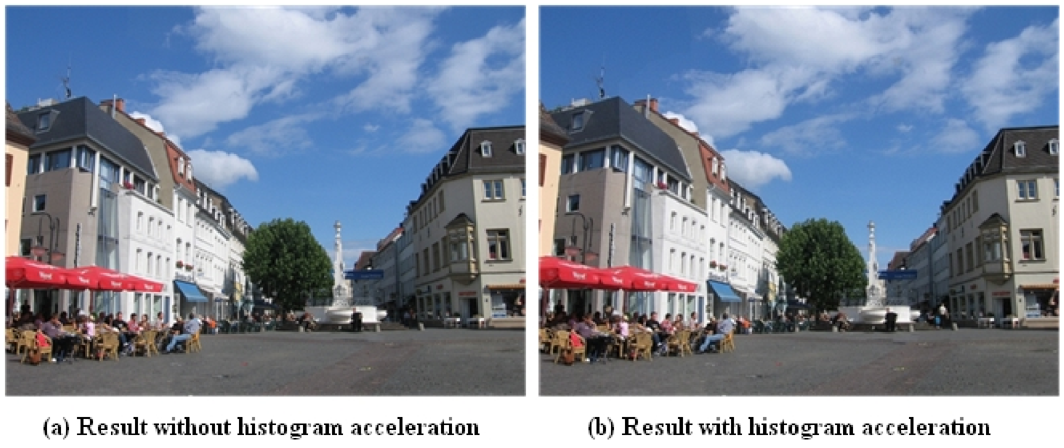

(a) Result without histogram acceleration
(b) Result with histogram acceleration

Figure 3.25: Comparison of the results with and without our histogram acceleration.

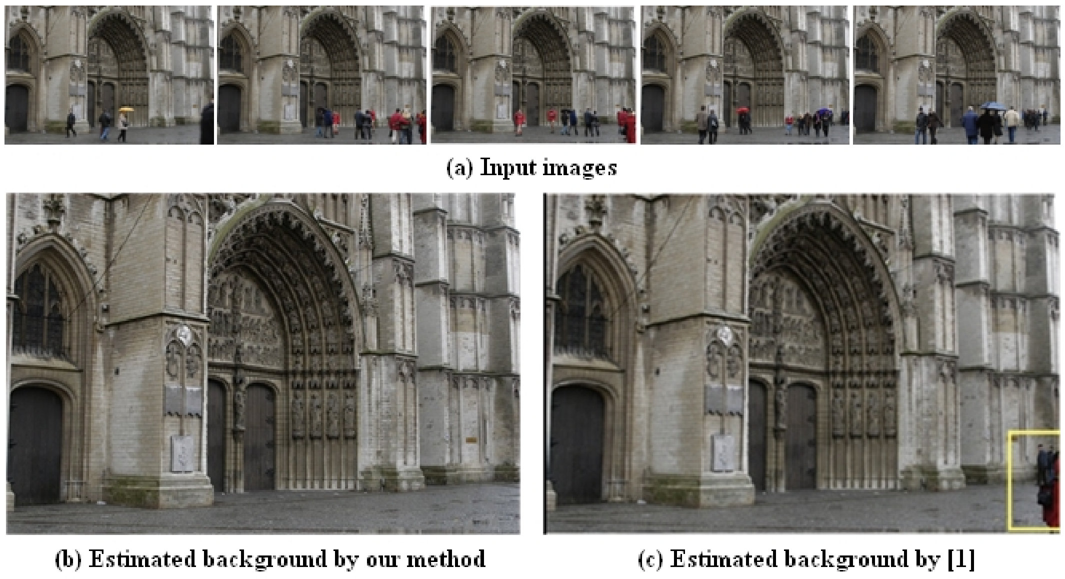

(a) Input images
(b) Estimated background by our method
(c) Estimated background by [1]

Figure 3.26: Applying our method to *Cathedral* scene. (a) Five input images. (b) Estimated background by applying our method. (c) Estimated background by [1].

We also apply our method to the *Cathedral* scene, which is originally used in [1]. There are five input images in the sequence, as shown in Figure 3.26(a). Then our result is shown in Figure 3.26(b). Again, errors exist in the red rectangle region in Figure 3.26, which is the result by applying [1]. By comparing our result to the result produced by the method in [1], we show that our method is better than theirs.

Next, our method is applied to a complex scene which is originally used in [3]. Figure 3.27(a) shows a subset of this image sequence which contains 21 images. Figure 3.27(b) is our result and Figure 3.27(c) the result using the method in [1]. Because this scene is extremely complicated, some of the transient objects remain in the results that are produced by both our method and the one in [1].

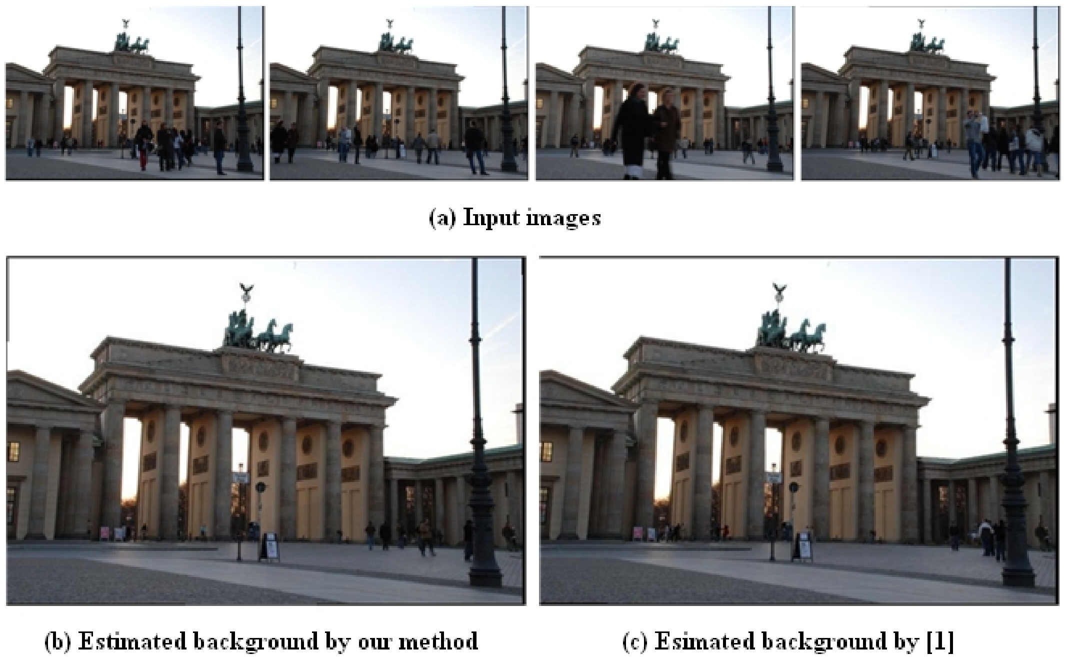

(a) Input images
(b) Estimated background by our method
(c) Esimated background by [1]

Figure 3.27: Applying our method to the *Brandenburg gate* scene. (a) 6 of 21 input images. (b) Estimated background by applying our method. (c) Estimated background by [1].

As we know, traffic surveillance is an important practical problem. An important first step is to estimate the background. In order to demonstrate that our method is practical in the real world, we apply it to a dataset that is captured by a traffic surveillance camera. The dataset is taken from [13], which has 25 images in total. To conserve space, we show only four of them. Figure 3.28(b) is our result and Figure 3.28(c) is the result produced using a median filter. This dataset shows that our method is much better than median filtering.

In the following experiment, we apply our sampling-based method to images taken from different viewpoints. The input images in Figure 3.29 are taken by a

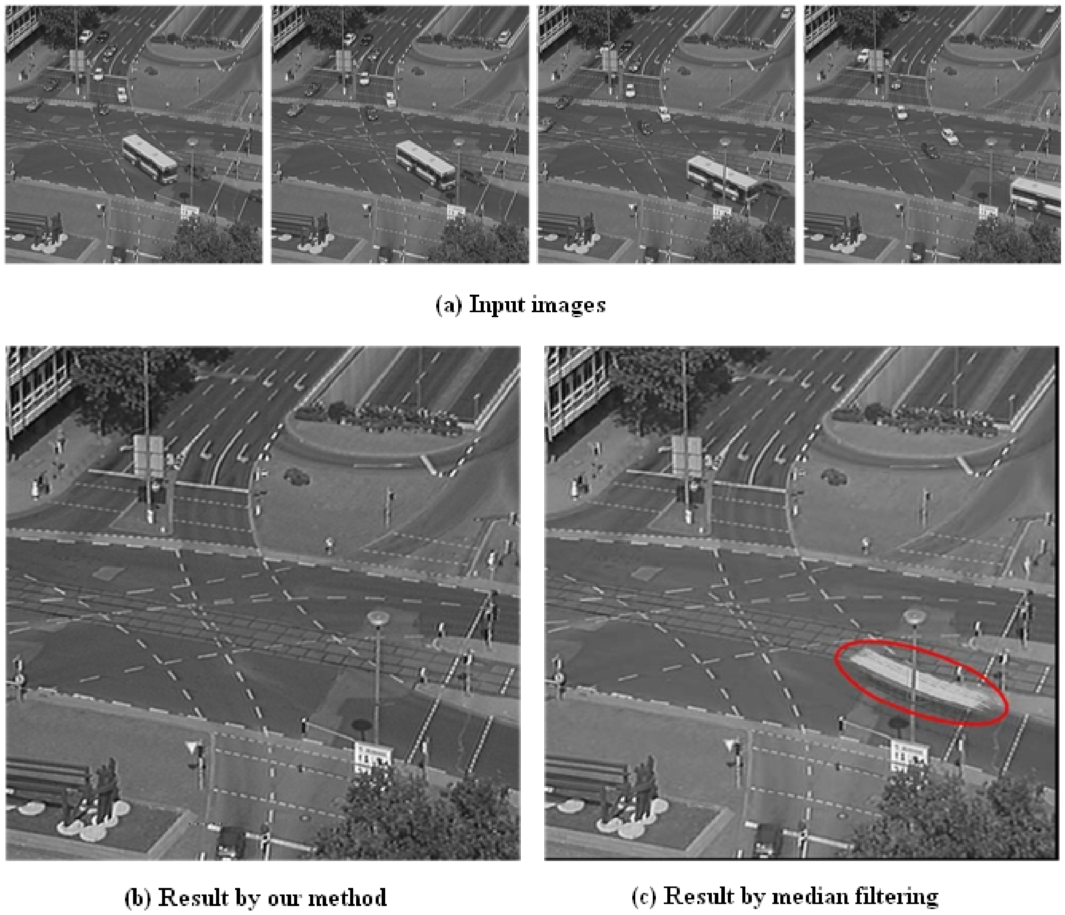


Figure 3.28: Applying our method to images taken by a traffic surveillance camera. (a) 4 of 25 input images. (b) Estimated background by applying our method. (c) Estimated background by a median filter (from[13]).

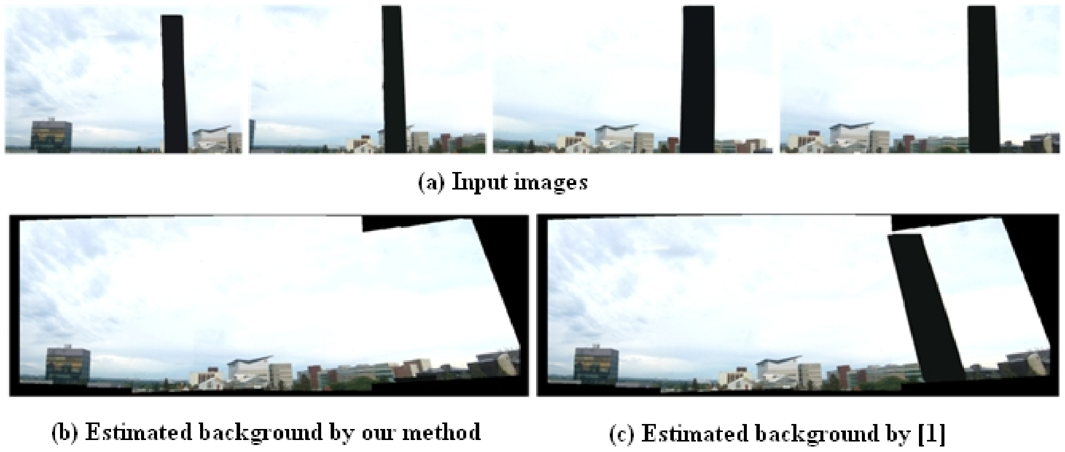

(a) Input images
(b) Estimated background by our method
(c) Estimated background by [1]

Figure 3.29: (a) Four images with a window frame as the foreground object. (b) Estimated background by applying our method. (c) Estimated background by [1].

moving camera and with window frames as the foreground objects. This dataset shows that our method can get correct result, which means that the window frames are removed in the background image. However, the result by applying [1] still has the window frame.

Figure 3.30(a) are four images taken from different viewpoints and with significant illumination changes. In Figure 3.30(b) and (c) the results by our method and by [1] are shown. The arrows in Figure 3.30(c) point out the error regions. In this dataset, gradient domain fusion is applied to remove some visual seams. However, we can see that there are some small foreground objects exist in our result. The result indicates that our sampling-based method cannot work perfectly on datasets with significant illumination changes.

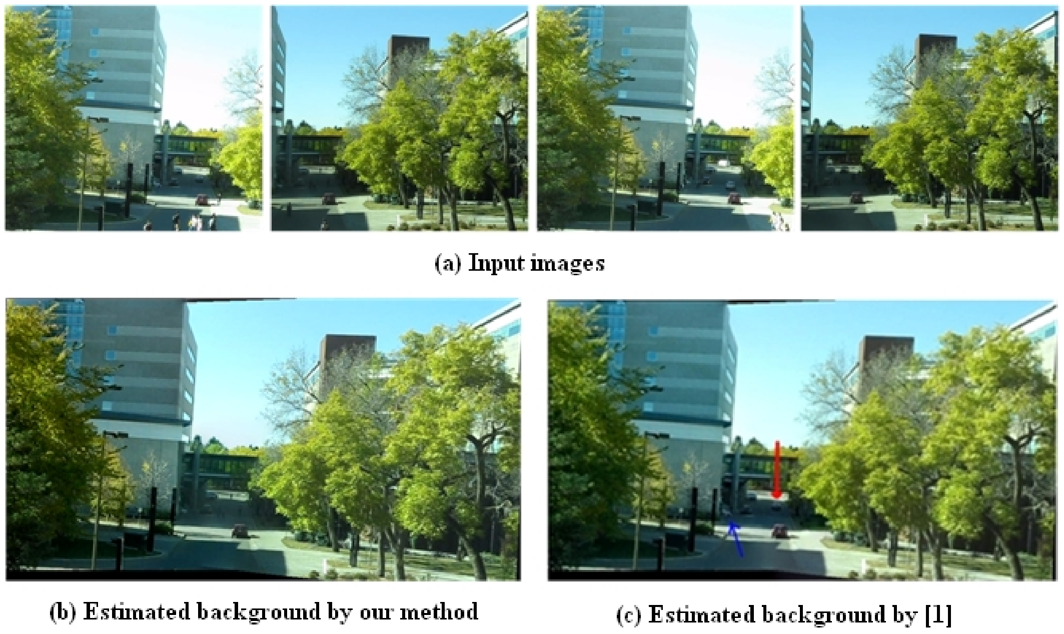

(a) Input images
(b) Estimated background by our method
(c) Estimated background by [1]

Figure 3.30: (a) Four images with large illumination changes. (b) Estimated background by applying our method. (c) Estimated background by [1].

3.2.7 Application to High Dynamic Range Images

In the final experiment, we apply our sampling-based method to a high dynamic range image sequence. The input images are captured under different exposure settings. The dataset is taken from [28], which is shown in Figure 3.31. The corresponding exposure times are also given in the figure. It is clear that the pixel values vary significantly in the same location among different images. Therefore, some pre-processing is required before we can apply our algorithm.

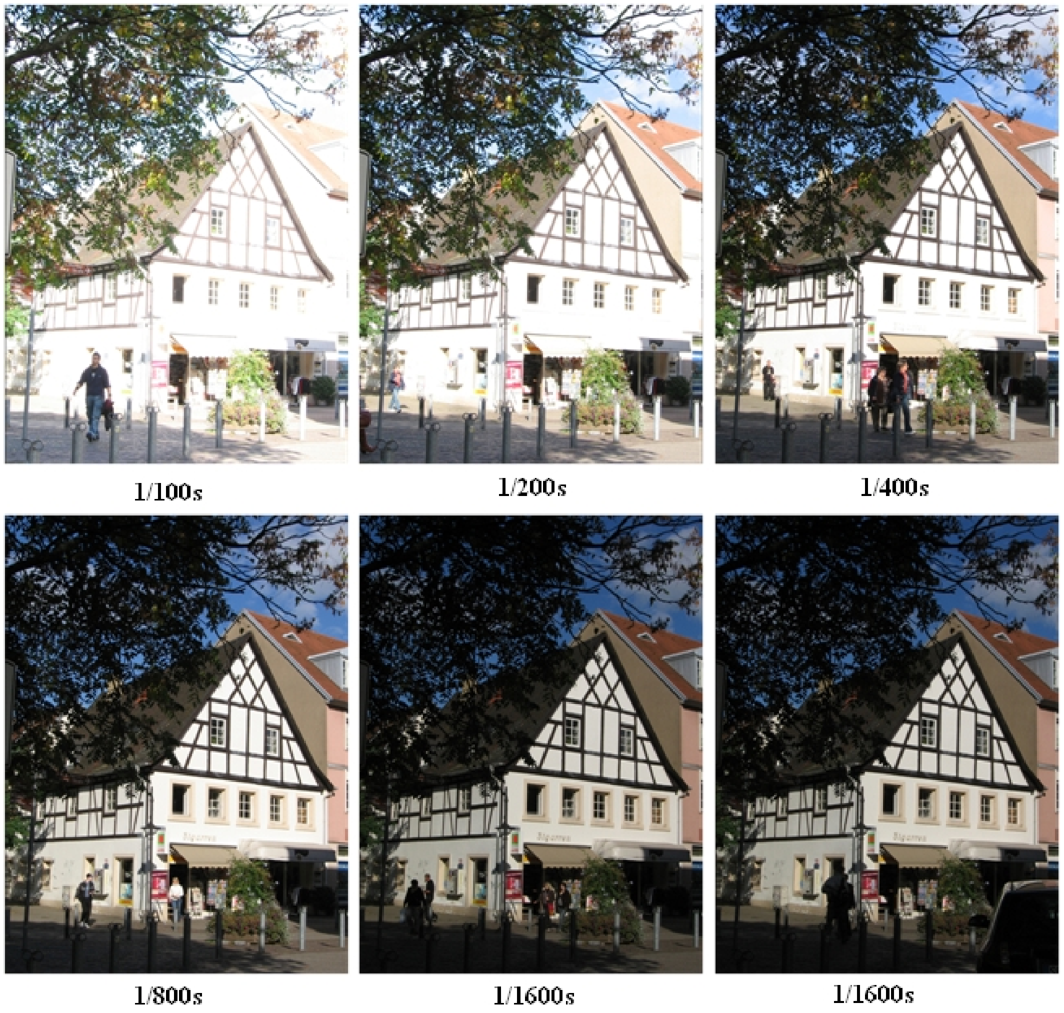


Figure 3.31: Input images with different exposure times.

In order to combine the input images, we use the algorithm proposed in [20] to recover the camera response function, and then we construct the radiance map for each image. Immediately after that, the “dodging-and-burning” method, which is proposed in [48], is applied to obtain the tone mapped image. A tone map or tone reproduction refers to the process of mapping a high dynamic range image back to low dynamic range. In our implementation, we use the global operators when applying the method in [48]. This is because the results provided by global operators are smoother in general. Moreover, the processing time can be significantly reduced by applying global operators than local operators. Figure 3.32 shows two tone-mapped input images. We can see that the illumination change is less significant than the original images in the input sequence.

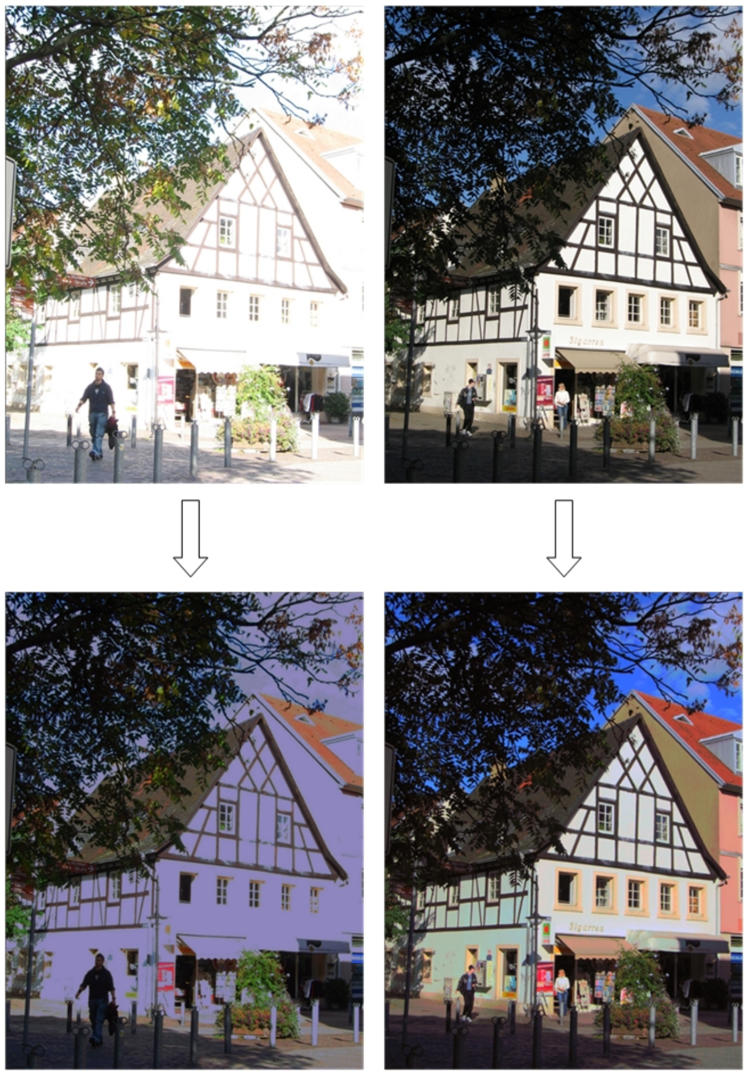


Figure 3.32: Corresponding tone mapped input images. Top row: original input images. Bottom row: tone mapped images.

Our method is applied directly to the tone-mapped images. When calculating the stable image I_{ST} for this dataset, we change the threshold t_H which is defined in Equation 3.19. This is because the illumination change is larger than a normal dataset such as that shown in Figure 3.26. t_H is changed to 25 for each channel if $I_1(p) = 0$ or $0.3 \cdot I_1(p)$ otherwise. Figure 3.33(a) shows the result of our method.

For comparison, the result by the "dodging-and-burning" method is shown in Figure 3.33(b). In particular, a weighting function [20] is applied when constructing the high dynamic range radiance map. Hence, artifacts are present in the result shown in Figure 3.33(b). In particular, all the transient objects from the input images can be seen in this image. However, these artifacts are completely removed using our method.

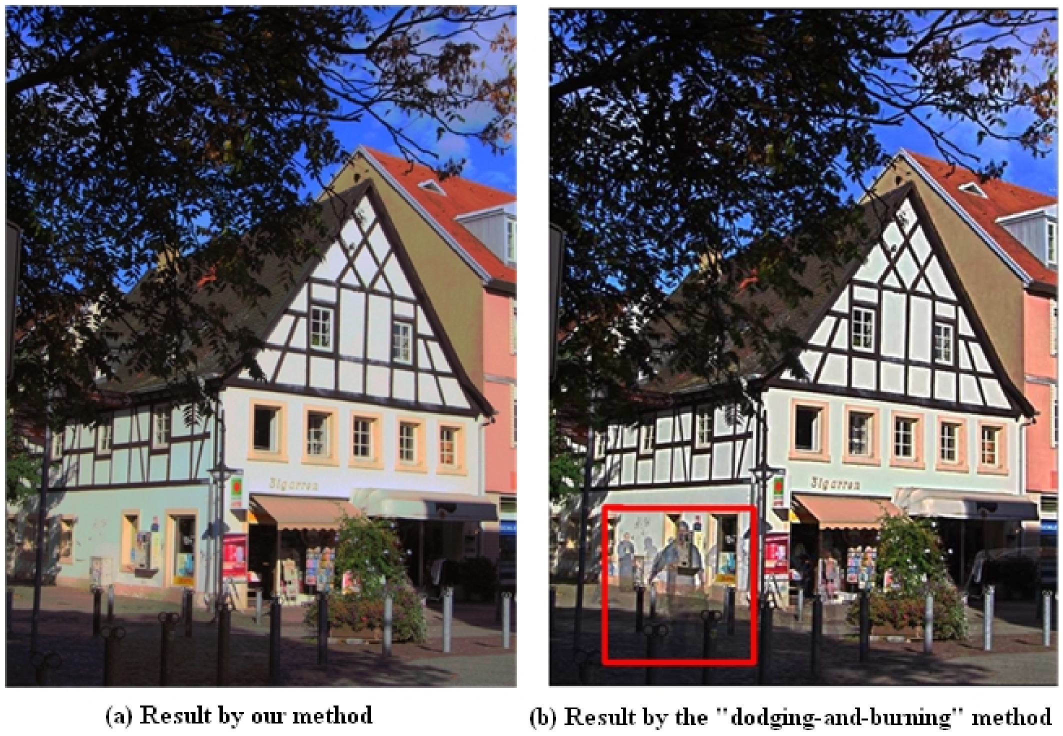


Figure 3.33: (a) Estimated background by our method. (b) The result produced by applying the "dodging-and-burning" method.

3.3 Conclusion

We have presented two novel methods for background estimation. Comparing the quality of the results, our sampling-based method is better than segmentation-based method in general. It can be indicated by the comparison between Figure 3.12(b) and Figure 3.26(b). However, the segmentation-based method performs much better in terms of processing time. When both methods are applied to the image sequence shown in Figure 3.12(a), it takes only 10 seconds for the segmentation-based method but about 100 seconds for the other one. The main reason is that we apply local and global color sampling for the predicted term. And this process can be time-consuming even with histogram acceleration. In real world applications, our segmentation-based method is suitable when the background is not too complicated such as the one shown in Figure 3.9. However, if the background is extremely complex and the foreground objects occupy a large portion of the scene (Figure 3.24(a)), then our sampling-based method is a more reasonable choice.

Chapter 4

GPU Speed Up

In our segmentation-based method, we apply an image segmentation algorithm first and linear blending last. In this chapter, we give a GPU implementation of both methods for speedup.

4.1 Overview

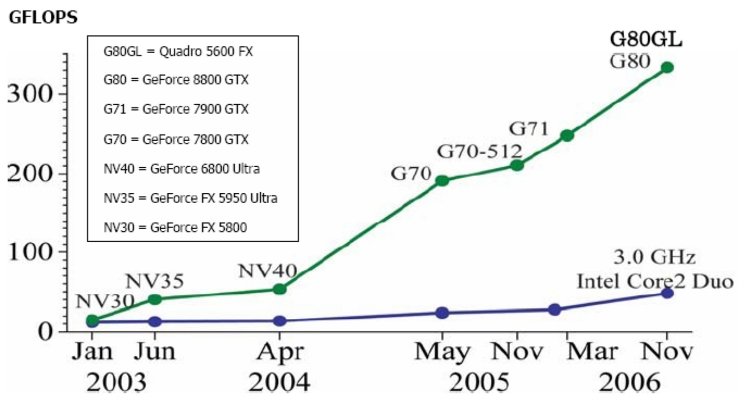


Figure 4.1: Floating-point operations per second for the CPU and the GPU. (from [46])

GPU, which stands for graphics processing unit, has successfully evolved into an indispensable resource in a computing system during the last few years driven by demands in the 3D computer game industry. Figure 4.1 shows the comparison of improvements in computational power between the CPU and the GPU. It shows that the GPU increases in performance at a rate much faster than the CPU. As a matter of fact, the most recent GPU chips have the computational power with nearly 1800

Gflops. Nowadays, GPUs offer incredible computing resources for both graphics and non-graphics processing as powerful parallel processors. Therefore, users are focusing on general-purpose computation using the GPU (GPGPU) during the last few years. As a matter of fact, image processing has become a popular topic for acceleration on the GPU as well. This is because many image processing methods have sections that consist of a common computation over many pixels. Ahn et al. [2] develop an image processing toolkit on the GPU which contains several techniques such as image segmentation and image enhancement. The image segmentation algorithms that implemented are isoperimetric graph partition, normalized cut and active contour. However, the method proposed in [23] and linear blending has not been implemented on the GPU before.

4.1.1 GPU Architecture

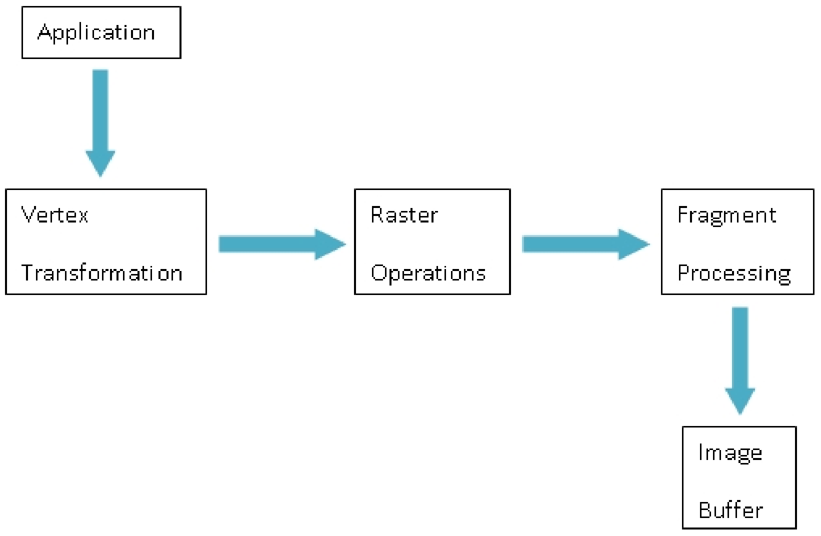


Figure 4.2: A simplified diagram of a programmable graphics pipeline.

The diagram of a typical programmable graphics pipeline is shown in Figure 4.2. The main function of the GPU is to use the geometrical information of a 3D scene, which is normally represented as vertices, which are then transformed to a 2D image composed of colored pixels. The graphics pipeline can be divided into several stages as follows.

➢ Application: This stage usually resides on the CPU rather than the GPU. It handles high-level operations. For example, sending the 3D geometry data in the form of vertex coordinates and the necessary commands to be performed.

➢ Vertex Transformation: The vertex processor is responsible for transformation and lighting. In transformation, it transforms 3D triangle vertices to 2D coordinates on an image plane according to a given camera model. In lighting, the vertex processor takes information such as the position and intensity of the light sources to compute the color of the vertex.

➢ Raster Operations: The primary function of this stage is to map a triangle into a set of screen pixels. It can be divided into two main stages. The first stage is to determine which pixels are part of the triangle. The second is the interpolation of vertex attributes such as color.

➢ Fragment Processing: This stage is to process the information from the rasterizer and to compute the final color of each pixel. The output of this stage is the 2D image buffer that is displayed on screen.

4.1.2 CUDA

GPGPU programming using a graphics API such as OpenGL or DirectX can simplify the use of the GPU. However, it has limitations as pointed out in [46]. For example, graphics APIs impose a high-learning curve for non-graphics users, and also incur overhead when the application is wrapped with graphics API calls.

CUDA, which is short for Compute Unified Device Architecture, is introduced by NVIDIA [46] to address the above mentioned limitations of GPGPU programming. The syntax of CUDA programming is similar to the C programming language. For example, the programmer can use *cudaMalloc* for allocating memory on the GPU which is very close to *malloc* in the C programming.

When programming using CUDA, the GPU is viewed as a compute device which could execute a high number of threads in parallel. Figure 4.3 illustrates the thread batching model of CUDA. In this figure, a grid includes all the kernels on the GPU for running a CUDA program. A block contains the kernels that run concurrently on one GPU multiprocessor. The threads in a block cooperate with each other through a shared memory.

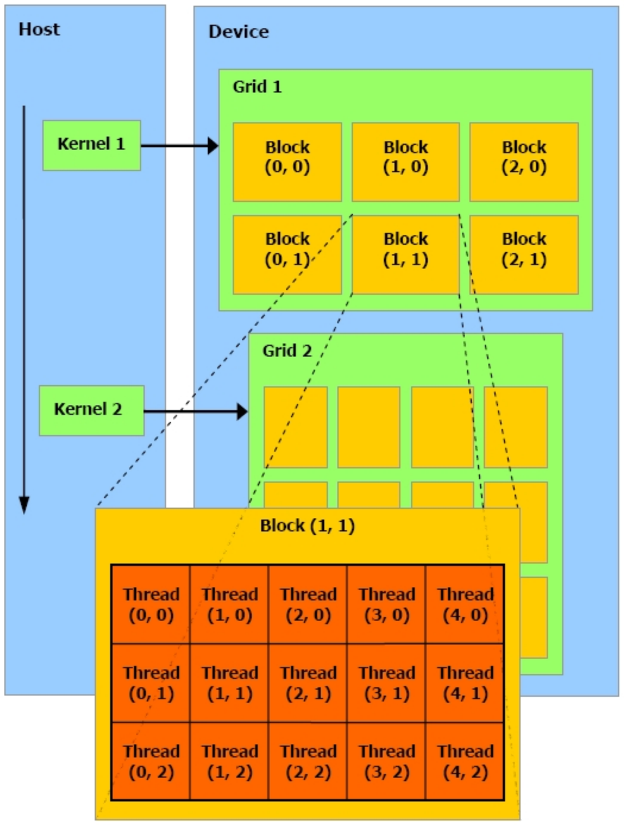


Figure 4.3: Thread-batching model of GPU. (from [46])

4.2 GPU Image Segmentation

Algorithm 2 in Section 3.1.2 is a step-by-step description of the image segmentation algorithm that we apply in our segmentation-based background estimation method. There are two parts in this algorithm that can be executed in parallel.

Given an image, the first step is to transform it into a graph which is then served as input to Algorithm 2. Figure 4.4 shows a graph which represents an image. In this graph, each node such as v corresponds to a pixel in the image. The weight of the edge e connecting a pair of adjacent pixels is computed by Equation 3.5. In the CPU implementation, the weight of each edge in the graph is computed sequentially. This process is parallelized in our GPU implementation. Because the threads on the GPU are executed in parallel, we assign each thread on the GPU to

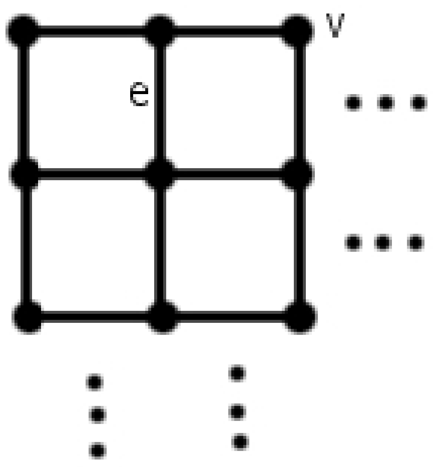


Figure 4.4: A graph: the representation of an image.

be in charge of an edge in the graph. Assigning one thread for one edge does not effect the result because when computing the weight for an edge, it only requires the intensity information of the adjacent pixels connected by this edge, which is stored in a shared memory. The pseudocode for building a graph is given in Algorithm 4. The CUDA code for each part that is ported to the GPU is provided in the Appendix B.

The second part that can be ported to the GPU is the third step of Algorithm 2, which is the main step to construct the segments. In this step, the algorithm processes each edge sequentially, and determines whether or not the two adjacent vertices that are connected by this edge should be merged. In our GPU implementation, we also assign one thread to process one edge. However, the results by our implementation can be different from the CPU version. A minimum spanning tree (MST) is maintained for each segment in the CPU version of this algorithm. When an edge is processed, the MST is updated. That is, whether two nodes connected by an edge should be merged or not is determined by Equation 3.6, which is defined based on the MSTs. When we port this part to the GPU, the MST is still maintained for each segment. However, since the edges are processed in parallel, the MST for each segment is different from the one on the CPU. A simple example is given as follows. Suppose the program is currently processing edge e on the CPU. Let v_i,

Algorithm 4 Building a graph for an image on GPU

```
// define the structure of an edge
struct {
    float w;    //the weight of the edge
    int a, b;   // the indices of two adjacent nodes connected by this edge
} edge;
Input: An array containing the color information of each pixel in the image
for each thread on the GPU do
  assign the indices of two adjacent nodes to edge.a and edge.b
  calculate the weight using the color information and assign it to edge.w
end for
synchronize all the threads
Output: A group of edges
```

v_j be the two adjacent pixels connected by e and $v_i \in R_i$, $v_j \in R_j$, where R_i and R_j are two separate segments. To determine whether or not v_i and v_j should be merged, the program uses the information of the MSTs from both R_i and R_j, as well as the information of the size of each segment. In the CPU version, when e is processed, both segments may already have some nodes in them, which means that their size can be large. However, when the threads are running in parallel on the GPU, the segments R_i and R_j may have only one node in each segment, that is, v_i in R_i and v_j in R_j. Since the size and the MST of each segment differ between the GPU and the CPU, the value of $P(R_1, R_2)$ could be different, and hence, different results are produced. We give the pseudocode of this part in Algorithm 5.

We compare the results produced by our GPU implementation and the result by the CPU version and then evaluate the performance.

Comparison: We conduct two experiments in order to show that our implementation can obtain results similar to the CPU implementation. First, we process an input image on both the CPU and the GPU and then compare the results, which should be similar to each other. The input image for this experiment is shown in Figure 4.5(a). We apply the image segmentation algorithm proposed in [23] and our

Algorithm 5 Segment a graph

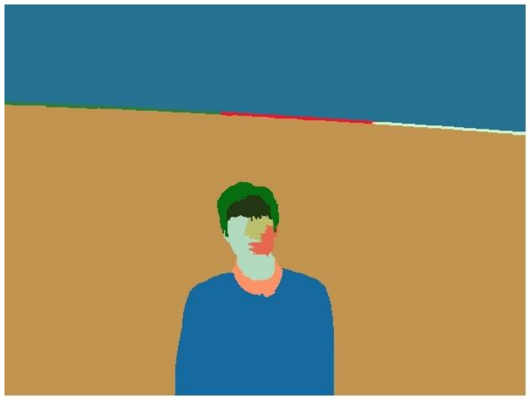
```
Input: A group of edges representing a graph
for each thread on the GPU do
    compute $P(R_1, R_2)$ in Equation 3.6 for each edge $e$ connecting two segments
    $R_1$ and $R_2$
    if $P(R_1, R_2) = true$ then
        merge $R_1$ and $R_2$
    else
        do nothing
    end if
end for
synchronize all the threads
Output: A group of separate segments
```

GPU implementation to this image. The image in Figure 4.5(b) is the segmentation result produced by [23] and Figure 4.5(c) is the result by our GPU implementation. The comparison indicates that our GPU implementation is able to produce result which is similar to the one by the original author.

In our second experiment, we use the segmentation results produced by our GPU implementation and redo the experiments of applying segmentation-based method. The images on the left column in Figure 4.6 shows the background estimation results by using our GPU segmentation results, and images on the right column are CPU image segmentation results. The input images for Figure 4.6(a) are from the same viewpoint and Figure 4.6(b) are from different viewpoints. The images on the left column and the right column demonstrates the similarity.

Performance: After the correctness is verified, the performance of our GPU implementation is evaluated. We apply our implementation to seven sets of images with different sizes and compare the running time on the GPU to the CPU. Table 4.1 shows the comparison of the running time and the speedup. Figure 4.7 shows that the curve of running time versus image size for the GPU stays much lower than that of the CPU. This means that the GPU can be used to process much larger size

(a) Input image


(b) Result produced by [17]

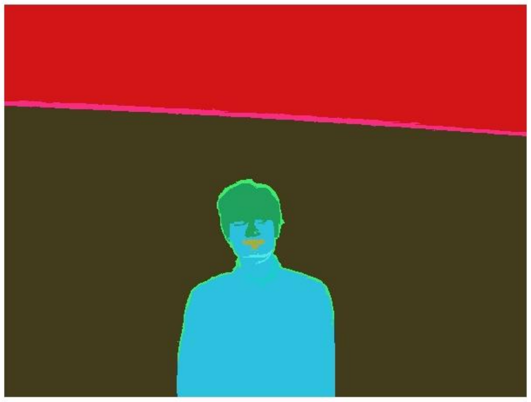

(c) Result produced by our GPU implementation

Figure 4.5: (a) An input image to the image segmentation algorithm. (b) The segmentation result produced by [23]. (c) The result by our GPU implementation.

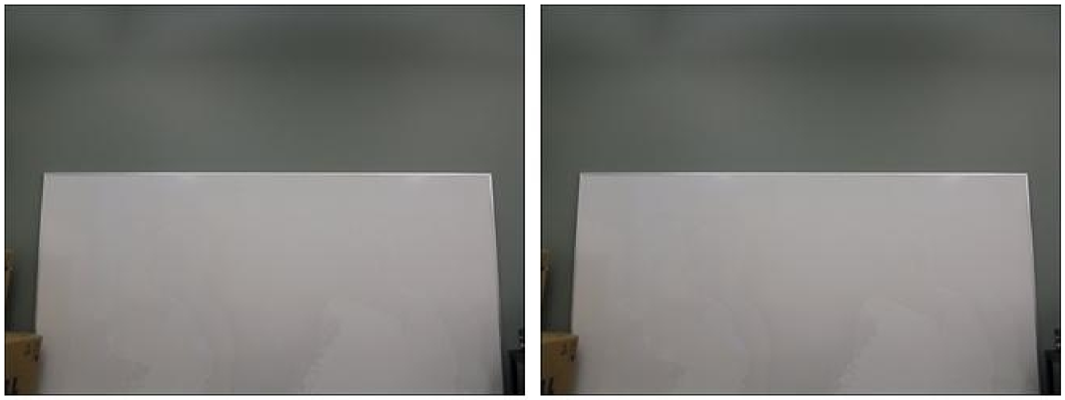
images.

Image Resolution	CPU	GPU	Speedup
1200 × 900	30.13s	1.49s	20
1632 × 1224	57.49s	2.98s	19
2308 × 1732	118.68s	5.57s	21
3264 × 2448	247.42s	11.71s	21

Table 4.1: Comparison between the running time for image segmentation on the CPU and the GPU.

4.3 GPU Linear Blending

As mentioned above, linear blending is the last step in our segmentation-based method. We also give a parallel implementation on the GPU for linear blending.


(a) Result of the dataset on the first row of Figure 3.8

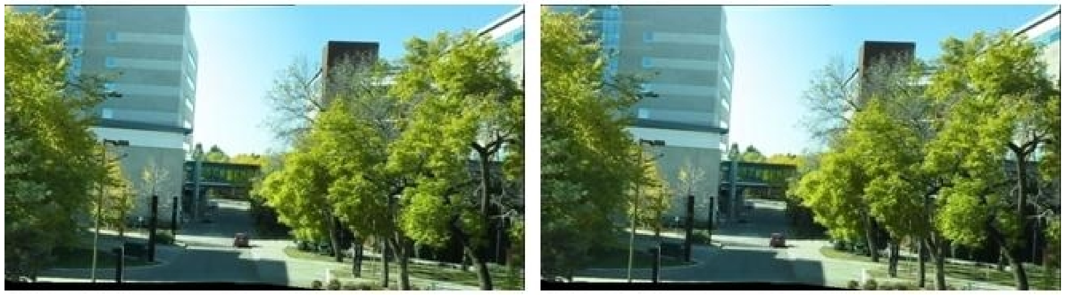

(b) Result of the dataset shown in Figure 3.9

Figure 4.6: Comparison of the background estimation results with the image segmentation results by both the GPU and the CPU implementation. (a) The estimated background of the dataset on the first row in Figure 3.8. (b) The estimated background of the dataset shown in Figure 3.9. Left column: with GPU image segmentation results. Right column: with CPU image segmentation results.

The description of linear blending is given in Section 3.1.5. The following is a simple example for linear blending. Given two images f_1 and f_2 from the same viewpoint with different illumination conditions. The algorithm processes pixels in one coordinate (x, y) at a time. In particular, it computes the weight w_1 for pixel p_1 in f_1, and w_2 for p_2 at the same coordinate in f_2, then outputs the result as $\frac{p_1 \times w_1 + p_2 \times w_2}{w_1 + w_2}$. Hence, to process the pixel at (x, y) does not require any information from other pixels. Therefore, it can be easily parallelized by assigning a thread on the GPU to compute the value of one pixel in the result, and the threads on GPU can execute concurrently. The pseudocode for GPU linear blending is given below as well.

In our experiments, we compare the results by GPU and CPU linear blending, and also evaluate the performance of our GPU implementation.

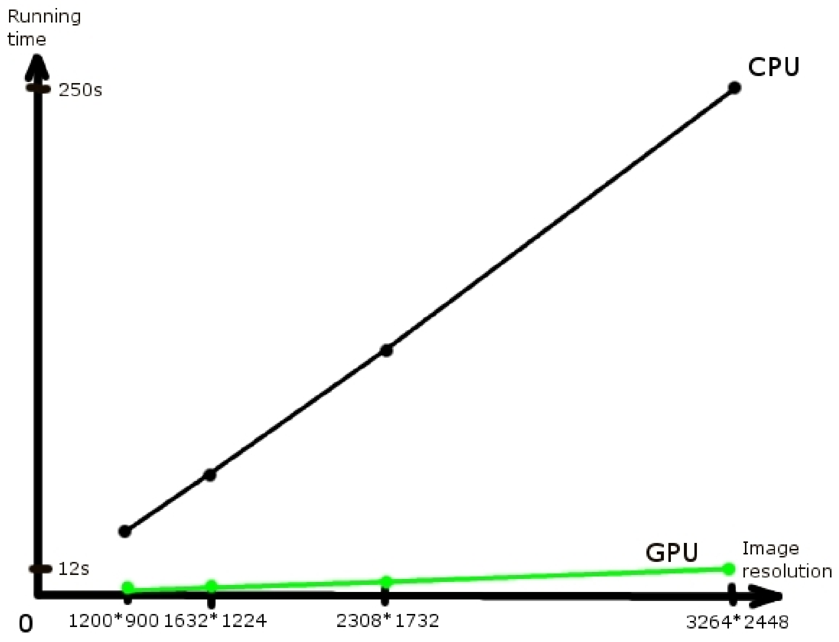


Figure 4.7: Running time of the image segmentation algorithm on the CPU and the GPU.

Comparison: We use two images shown in Figure 4.8(a), which are taken from the same viewpoint but with illumination changes between them. Then Figure 4.8(b) and (c) show the results of linear blending on the GPU and on the CPU, respectively.

We use Equation 4.1 to measure the similarity between Figure 4.8(b) and (c). That is, we compute the sum of the absolute difference on three color channels and then take the average as the measure. In this equation, N is the total number of pixels in an image, and I^k_{n,f_1} is the value of color channel k for pixel n in image f_1. We apply this equation to the images shown in Figure 4.8(b) and (c), and $Diff = 2.836705$. That is, the average difference for each pixel in the image is less than 3, and that is the sum of differences on the three color channels. This experimental result of measure indicates that the difference of two implementation results is sufficiently small so that there is no visual difference.

$$Diff = \frac{\sum\limits_{n=1}^{N} \sum\limits_{k=R,G,B} |I^k_{n,f_1} - I^k_{n,f_2}|}{N} \tag{4.1}$$

Besides the above comparison, we also compare the background estimation re-

Algorithm 6 GPU linear blending

```
Input: Two arrays $p_1$, $p_2$ with the same size, holding the color value for two input
        images and
        An array $r$ with the same size as $p_1$, and all elements in it are initialized
        to be 0.
for each thread $t$ on the GPU do
    compute the weight $w_1[t]$ for pixel $p_1[t]$ and $w_2[t]$ for $p_2[t]$
    $r[t] = \frac{p_1[t]*w_1[t]+p_2[t]*w_2[t]}{w_1[t]+w_2[t]}$
end for
synchronize all the threads
Output: The array $r$
```

sult by linear blending on the CPU and on the GPU. The results are shown in Figure 4.9. The input datasets used in this experiment are the same as those used in Figure 4.6. The comparison indicates that our GPU linear blending results are comparable to the ones by the CPU version.

Performance: To evaluate the performance of our GPU linear blending, we apply it to images with different resolutions. Table 4.2 shows the running time and the speedup. Similar to Figure 4.7, the curves of the running time for the CPU and for the GPU are shown in Figure 4.10. The figure demonstrates that our GPU implementation has significant improvement on the processing speed.

Image Resolution	CPU	GPU	Speedup
1200×900	0.52s	90ms	6
1632×1224	0.94s	105ms	9
2308×1732	1.81s	140ms	13
3264×2448	3.80s	203ms	19
4032×3024	5.89s	266ms	22

Table 4.2: Comparison between the running time on the CPU and on the GPU for linear blending.

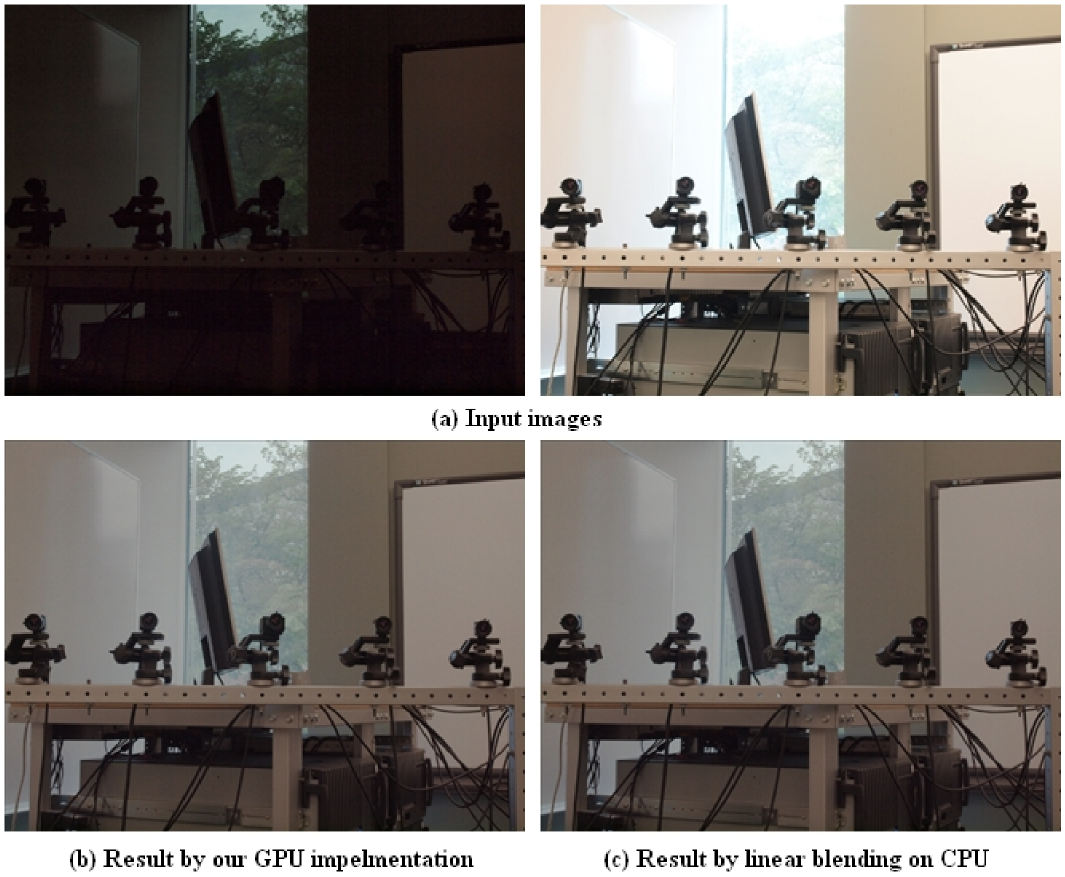

(a) Input images
(b) Result by our GPU impelmentation
(c) Result by linear blending on CPU

Figure 4.8: Linear blending on the CPU and on the GPU. (a) The input images for linear blending. (b) Result by GPU linear blending. (c) Linear blending result on the CPU.

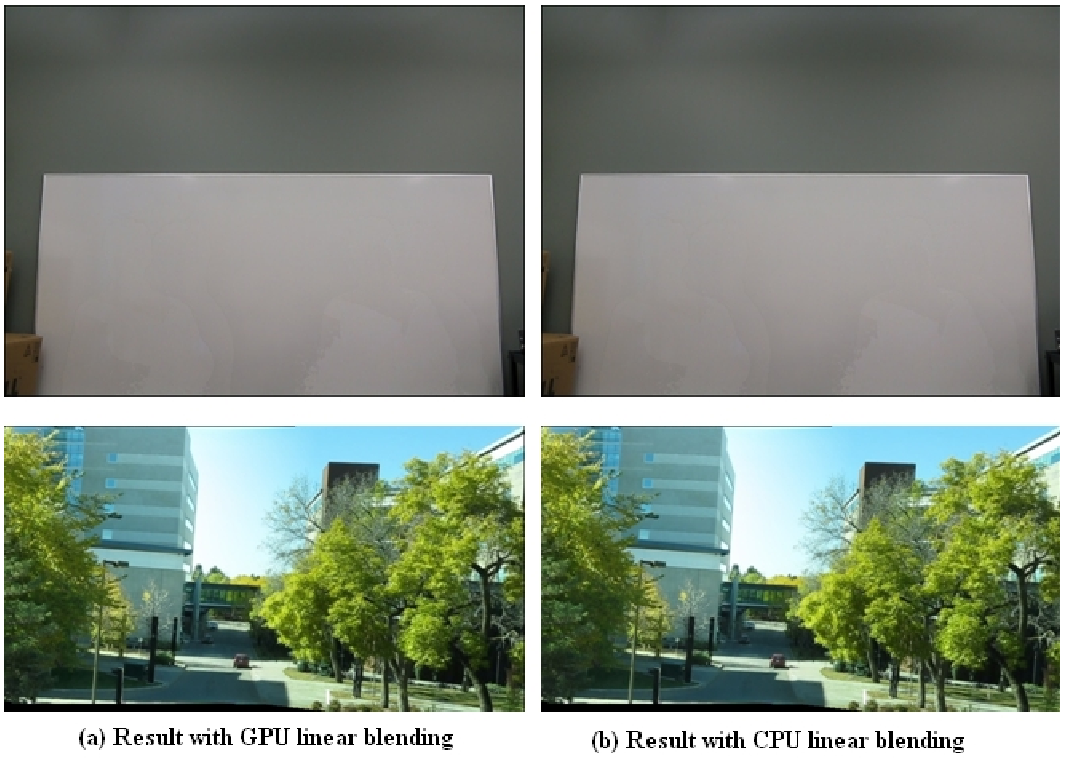

(a) Result with GPU linear blending
(b) Result with CPU linear blending

Figure 4.9: Comparison of the background estimation results with linear blending on the CPU and on the GPU. Left column: with linear blending on the CPU. Right column: with linear blending on the CPU.

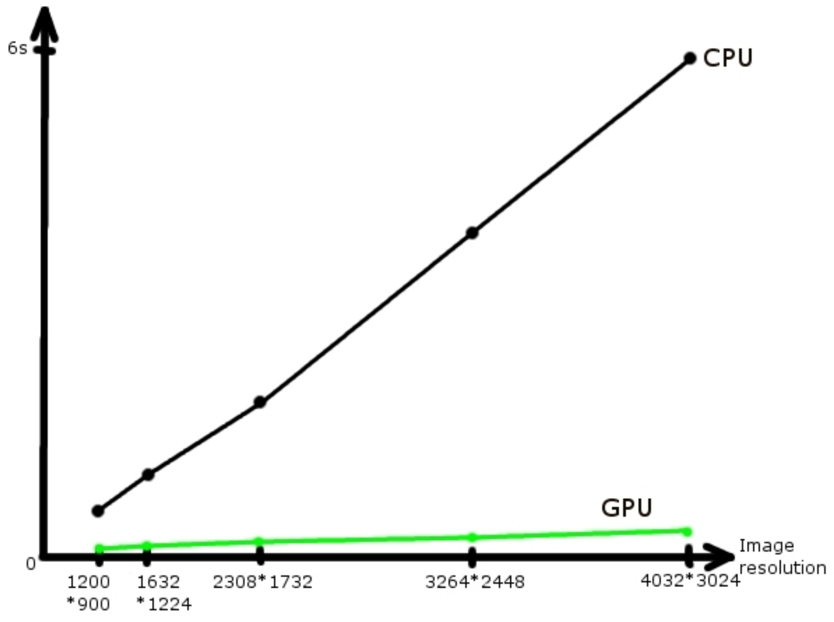


Figure 4.10: Running time of linear blending on the CPU and on the GPU.

Chapter 5

Conclusions and Future Work

5.1 Contributions

In this thesis, we present two novel methods for background estimation and also provide GPU speedup.

Segmentation-based method

The novelty of this method is to integrate the information of image segmentation. The cost function defined in this method includes a smoothness measure and a stationary coefficient. When defining the smoothness measure, we take advantage of the image segmentation results to make it more reasonable. A dynamic programming framework is applied to choose the candidate pixels which could minimize the aggregate cost along each scanline. Since significant illumination variation may be present among input images, linear blending is applied to remove any visible seams after candidate pixels are selected on each scanline. We use this method to process several datasets including images with similar illumination condition and with significantly illumination changes, and the results are promising.

In order to eliminate the parameter used in linear blending, we propose an alternative method, which is to apply linear blending first with the blended results served as input images. After that, the cost function is defined based on the blended results and the candidate pixels selected by the DP framework are the final results.

During our experiments, we also discover the limitations of the blending-fist method. Since the number of input images increases, the method is much slower than the blending-last method. When the scene has many transient objects such as

Figure 3.14(a), there will be error regions in the result (Figure 3.14(b)). Although the blending-fist method is able to eliminate one parameter, it actually decreases the probability of background pixels in the input sequence. As a result, the stationary coefficient makes the background pixels less likely to be selected.

Sampling-based method

In this method, the background estimation is cast into a labeling problem. We define an energy function based on the MRF formulation and use graph cuts to minimize it. The energy function includes a data term and a smoothness term. When the data term is defined, we integrate an image inpainting technique which has never been used in background estimation. In particular, we detect the stable regions among the input images, and then apply image inpainting to fill in the unstable regions. A predicted term is defined based on the filled-in regions by the image inpainting technique. Finally, a smoothness term is defined to make sure that the results are visually smooth. Therefore, gradient-domain fusion is applied only when the images are taken with different illumination conditions. We apply this method to datasets from the same viewpoint and from different viewpoints as well. The comparison indicates that our method is able to produce better results than other methods [1] [13] [58]. In our experiments, we demonstrate the importance of the integrated image inpainting method. In the final experiment, we also show that the sampling-based method can be applied to high dynamic range images.

The limitation of this method is on processing time. First of all, the graph cuts minimization technique is slower than DP in general. The second reason is due to the sampling used in the image inpainting technique. The global color sampling (Section 3.2.3) can be time consuming if the image resolution is high and the image set has huge unstable regions. Even with our histogram acceleration, the processing time is still much longer than our segmentation-based method.

GPU speedup

Besides the two novel background estimation methods that described above, we port the image segmentation algorithm and linear blending from the CPU to the GPU. We take advantage of the features of the GPU. In particular, the CUDA programming facility and the parallel execution of threads on the GPU. By porting the methods into the GPU, the speedup can be as high as 20 times.

5.2 Future Work

In our segmentation-based method, we set a threshold when performing linear blending and the threshold needs to be changed based on the illumination of the input images. We propose an alternative method to eliminate the threshold, which results in another constraint. That is, the alternative method cannot be applied to images with many transient objects in the scene. An interesting research topic is to eliminate the parameters that we have, and to make the method completely automatic.

In our sampling-based method, we integrate a simple image inpainting technique. As is known to all, there are many methods proposed in this area [6] [4] [5] [7]. In our method, we use only color information for sampling. It is possible to apply different image inpainting techniques for comparison. For example, we can use structure and texture information for inpainting [7], in addition to color information. However, it may take more time for processing if we apply more sophisticated inpainting methods.

Another possible research direction is to reduce the processing time in the sampling-based method. The global color sampling can be time consuming if the resolution of the input images is high and the unstable regions are large. Therefore, a faster image inpainting technique that can produce results with the same quality is worthy for further investigation. One avenue along this direction is to exploit using the GPU.

Bibliography

[1] A. Agarwala, M. Dontcheva, M. Agrawala, S. M. Drucker, A. Colburn, B. Curless, D. Salesin, and M. F. Cohen. Interactive digital photomontage. volume 23, pages 292–300, 2004.

[2] I. Ahn, M. Lehr, and P. Turner. Image processing on the gpu. White paper, University of Pennsylvania, February 2005. Available online.

[3] M. Alexa. Extracting the essence from sets of images. In *Proceedings of the Eurographics Workshop on Computational Aesthetics*, pages 113–120, 2007.

[4] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera. Filling-in by joint interpolation of vector fields and gray levels. *IEEE Transactions On Image Processing*, 10(8):1200–1211, 2001.

[5] M. Bertalmio, A. Bertozzi, and G. Sapiro. Navier-stokes, fluid-dynamics and image and video inpainting. In *Proceedings of the Eighth IEEE International Conference on Computer Vision*, pages 355–362, 2001.

[6] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of ACM SIGGRAPH 2000*, pages 417–424, 2000.

[7] M. Bertalmo, L. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting. *IEEE Transactions On Image Processing*, 12(8):882–889, 2003.

[8] T. Boult, R. Micheals, X. Gao, P. Lewis, C. Power, W. Yin, and A. Erkan. Frame-rate omnidirectional surveillance and tracking of camouflaged and occluded targets. In *Proceedings of Second IEEE Workshop on Visual Surveillance*, pages 48–55, Fort Collins, Colorado.

[9] Y. Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, 26(9):1124–1137, September 2004.

[10] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, 23(11):1222–1239, November 2001.

[11] M. Brown and D. G. Lowe. Recognising panorama. In *Proceedings of the 9th International Conference on Computer Vision (ICCV 2003)*, pages 1218–1225, Nice, France, 2003.

[12] P. J. Burt and R. J. Kolczynski. A multiresolution spline with applications to image mosaics. *ACM Transactions on Graphics*, 2(4):217–36, 1983.

[13] S. Cohen. Background estimation as a labeling problem. In *Proceedings of the 10th IEEE International Conference on Computer Vision (ICCV 2005)*, pages 1034–1041. IEEE Computer Society, 2005.

[14] A. Colombari, A. Fusiello, and V. Murino. Background initialization in cluttered sequences. In *Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*, pages 197–202, 2006.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. The MIT Press, Massachusetts, 1990.

[16] A. Criminisi, G. Cross, A.Blake, and V. Kolmogorov. Billayer segmentation of live video. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006)*, pages 53–60, New York City, NY, 2006.

[17] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts and shadows in video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1337–1342, 2003.

[18] R. Cutler and L. Davis. View-based detection and analysis of periodic motion. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, pages 495–500, Brisbane, Australia, 1998.

[19] J. Davis. Mosaics of scenes with moving objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 354–360, Santa Barbara, 1998.

[20] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97 Conference Proceedings*, pages 369–378, August 1997.

[21] A. Elgammal, D. Harwood, and L. Davis. Non-parametric model for background subtraction. In *Proceedings of the Sixth European Conference on Computer Vision*, volume 2, pages 751–767, London, UK, 2000.

[22] R. Fattal, D. Lischinski, and M. Werman. Gradient domain high dynamic range compression. *ACM Transactions on Graphics*, 21(3):249–256, 2002.

[23] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[24] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 175–181, San Francisco, CA, 1997.

[25] X. Gao, T. Boult, F. Coetzee, and V. Ramesh. Error analysis of background adaption. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 503–510, Hilton Head Isand, SC, 2000.

[26] B. Gloyer, H. K. Aghajan, K.-Y. Siu, and T. Kailath. Video-based freeway monitoring system using recursive vehicle tracking. In *Proceedings of SPIE*, pages 173–180, 1995.

[27] N. Gracias, A. Gleason, S. Negahdaripour, and M. Mahoor. Fast image blending using watershed and graph cuts. In *Proceedings of British Machine Vision Conference*, pages 469–478, 2006.

[28] M. Granados, H. Seidel, and H. P. A. Lensch. Background estimiation from non-time sequence images. In *Graphics Interface*, pages 33–40, 2008.

[29] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 22–29, 1998.

[30] D. Gutchess, M. Trajkovic, E. Cohen-Solal, D. Lyons, and A. K. Jain. A background model initialization algorithm for video surveillance. In *Proceedings of the International Conference on Computer Vision (ICCV 2001)*, pages 733–740, 2001.

[31] G. Halevi and D. Weinshall. Motion of disturbances: Detection and tracking of multi-body non-rigid motion. In *Machine Vision and Applications*, pages 122–137, 1999.

[32] J. Heikkila and O. Silven. A real-time system for monitoring of cyclists and pedestrians. In *Proceedings of Second IEEE Workshop on Visual Surveillance*, pages 246–252, Fort Collins, Colorado.

[33] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Proceedings of the second European Workshop on Advanced Video Based Surveillance Systems*, 2001.

[34] K.-P. Karmann and A. Brandt. Moving object recognition using an adaptive background memory. In *Time-Varying Image Processing and Moving Object Recognition*, pages 289–307, 1990.

[35] D. Koller, J. Weber, and J. Malik. Robust multiple car tracking with occlusion reasoning. Technical report, EECS Department, University of California at Berkeley, 1994.

[36] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, 26(2):147–159, February 2004.

[37] E. S. Larsen, P. Mordohai, M. Pollefeys, and H. Fuchs. Temporally consistent reconstruction from multiple video streams using enhanced belief propagation. In *Proceedings of the International Conference on Computer Vision*, pages 1–8, Rio de Janeiro, Brazil, 2007.

[38] D.-S. Lee, J. Hull, and B. Erol. A bayesian framework for gaussian mixture background modeling. In *Proceedings of IEEE International Conference on Image Processing*, Barcelona, Spain.

[39] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *Eighth European Conference on Computer Vision (ECCV 2004)*, pages 377–389, 2004.

[40] Y. Liu, T. Belkina, J. H. Hays, and R. Lublinerman. Image de-fencing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[41] B. P. L. Lo and S. A. Velastin. Automatic congestion detection system for underground platforms. In *Proceedings of 2001 International symposium on intelligent multimedia, video, and speech processing*, pages 158–161, Hong Kong, 2001.

[42] W. Long and Y.-H. Yang. Stationary background generation: An alternative to the difference of two images. *Pattern Recognition,*, 23:1351–1359, 1990.

[43] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the 7th International Conference on Computer Vision (ICCV'99)*, pages 1150–1157, Corfu, Greece, 1999.

[44] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[45] N. McFarlane and C. Schofield. Segmentation and tracking of piglets in images. *Machine Vision and Applications,*, 8(3):187–193, 1995.

[46] NVIDIA. Nvidia programming guide. Technical report, NVIDIA, 2008.

[47] P. W. Power and J. A. Schooness. Understanding background mixture models for foreground segmentation. In *Proceedings Image and Vision Computing New Zealand*, pages 267–271, Auckland, New Zealand.

[48] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. In *Proceedings of ACM SIGGRAPH 2002*, pages 267–276, 2002.

[49] P. Remagnino, A. Baumberg, T. Grove, D. Hogg, T. Tan, A. Worrall, and K. Baker. An integrated traffic and pedestrian model-based vision system. In *Proceedings of the Eighth British Machine Vision Conference*, pages 380–389, 1997.

[50] H.-Y. Shum and R. Szeliski. Construction of panoramic mosaics with global and local alignment. *International Journal of Computer Vision*, 36(2):101–30, 2000.

[51] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the 1999 Conference on Computer Vision and Pattern Recognition (CVPR 1999)*, pages 2246–2252, 1999.

[52] C. Stauffer and W. E. L. Grimson. Learning patterns of activity using real-time tracking. pages 747–757, 2000.

[53] J. Sun, W. Zhang, X. Tang, and H.-Y. Shum. Background cut. In *Proceedings of European Conference on Computer Vision*, pages 628–641, 2006.

[54] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization

method for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, pages 1068–1080, 2008.

[55] M. F. Tappen and W. T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003)*, pages 900–907, 2003.

[56] M. Uyttendaele, A. Eden, and R. Szeliski. Eliminating ghosting and exposure artifacts in image mosaics. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, pages 509–516, Kauai, Hawaii, 2001.

[57] C. Wren, A. Azabayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 780–785, 1997.

[58] Xun Xu and Thomas S. Huang. A loopy belief propagation approach for robust background estimation. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008)*, Anchorage, Alaska, June 24-26 2008. IEEE Computer Society.

[59] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. Technical report, Mitsubishi Electric Research Laboratories, January 2002.

[60] Q. Zhou and J. Aggarwal. Tracking and classifying moving objects from videos. In *Proceedings of IEEE Workshop on Performance Evaluation of Tracking and Surveillance*, 2000.

[61] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics*, 23(3):600–608, 2004.

Appendix A

Homography

In computer vision, two images are related by a homography only if:

➢ They are viewing the same plane from different angles.

➢ They are taken by the same camera but from different angles.

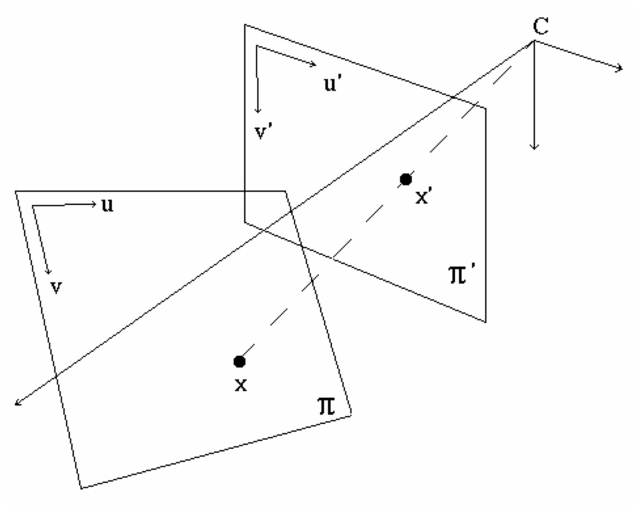


Figure A.1: Projective transformation

Figure A.1 illustrates the geometry of an homography transform. $x = (u, v, 1)$ and $x^{'} = (u^{'}, v^{'}, w^{'})$ are two matching points in different images, and their coordinates are related by a 3×3 homography matrix. In particular, $x^{'} = Hx$, where H is the 3×3 matrix. It leads to the following equation.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u^{'} \\ v^{'} \\ w^{'} \end{bmatrix} \tag{A.1}$$

where $H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$ is the homography matrix. When solving the above equation, h_{33} is normally scaled to be 1. As a result, there are 8 unknowns in total. Therefore, we need 4 pairs of corresponding points to obtained the 8 unknown elements in H.

Appendix B

GPU Segmentation

B.1 Building the graph

```
//
// parameter list:
// ed_d: an array of edges. The structure of an edge is given in Algorithm 5.
// img_d: an array containing color information of pixels.
// width: the width of the input image.
// height: the height of the input image.
// Channels: number of color channels of the input images.
// num_edges: the total number of edges in the graph.
//
__global__ void build(edge* ed_d, uchar* img_d, int width, int height, int Channels,
int num_edges)
{
    // we operate on each edge try to set a, b and w for each edge
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    if (idx < num_edges) {
        int row, col;
        int tempa, tempb, tempc, index1, index2;
        // if the edge is on the last row of the image
        if (idx >= (2*width-1)*(height-1)) {
            row = height-1; col = idx-row*(2*width-1);
            tempa = row*width+col;
            index1 = tempa*Channels;
            ed_d[idx].a = tempa; ed_d[idx].b = tempa+1;
            tempa = img_d[index1]-img_d[index1+3];
            tempb = img_d[index1+1]-img_d[index1+4];
            tempc = img_d[index1+2]-img_d[index1+5];
            ed_d[idx].w = sqrt((float)(tempa*tempa + tempb*tempb + tempc*tempc));
        }
        //if the edge is on the last column of the image
        else if ((idx+1)%(2*width-1) == 0) {
            row = (int)idx/(2*width-1); col = width-1;
            tempa = row*width+col; tempb = (row+1)*width+col;
            index1 = tempa*Channels; index2 = tempb*Channels;
            ed_d[idx].a = tempa; ed_d[idx].b = tempb;
            tempa = img_d[index1]-img_d[index2];
            tempb = img_d[index1+1]-img_d[index2+1];
            tempc = img_d[index1+2]-img_d[index2+2];
            ed_d[idx].w = sqrt((float)(tempa*tempa + tempb*tempb + tempc*tempc));
        }
        else {
            // if the edge is in the middle of the image
            row = (int)idx/(2*width-1); col = (int)(idx-row*(2*width-1))/2;
            if ((idx-row*(2*width-1))%2 == 0) {
                tempa = row*width+col;
                index1 = tempa*Channels;
                ed_d[idx].a = tempa; ed_d[idx].b = tempa+1;
                tempa = img_d[index1]-img_d[index1+3];
                tempb = img_d[index1+1]-img_d[index1+4];
                tempc = img_d[index1+2]-img_d[index1+5];
                ed_d[idx].w = sqrt((float)(tempa*tempa + tempb*tempb + tempc*tempc));
            }
```

```
            else {
                tempa = row*width+col; tempb = (row+1)*width+col;
                index1 = tempa*Channels; index2 = tempb*Channels;
                ed_d[idx].a = tempa; ed_d[idx].b = tempb;
                tempa = img_d[index1]-img_d[index2];
                tempb = img_d[index1+1]-img_d[index2+1];
                tempc = img_d[index1+2]-img_d[index2+2];
                ed_d[idx].w = sqrt((float)(tempa*tempa + tempb*tempb + tempc*tempc));
            }
        }
    }
    __syncthreads();
}
```

B.2 Segmentation

```
//
// parameter list:
// ed_d: an array of edges.
// m_d: an array with the same size of ed_d, representing whether two adjacent
// nodes connected by a certain edge should be merged or not.
// size_d: the size of each segment.
// th_d: an array stores the information of Int(R) + τ(R), which is
// shown in equation 3.8.
// num_edges: the total number of edges in the graph.
// c: the same as the parameter k in equation 3.10.
//
__global__ void segment(edge* ed_d, uchar* m_d, int* size_d, float* th_d, int num_edges,
float c)
{
    // we operate on each edge, that is, each element in the array ed_d[].
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    if (idx < num_edges) {
        int a = ed_d[idx].a;
        int b = ed_d[idx].b;
        float w = ed_d[idx].w;
        if (a ! = b) {
            //if the predicate in equation 3.6 is false, which indicates the two nodes
            //should be merged.
            if ((w <= th_d[a]) && (w <= th_d[b])) {
                //set the flag to be 1.
                m_d[idx] = 1;
                //change the size of the segment
                size_d[a] = size_d[a]+size_d[b];
                size_d[b] = size_d[a];
                // use the th_d to maintain a MST.
                // refer to the sequential code
                th_d[a] = w + (c/size_d[a]);
                th_d[b] = th_d[a];
            }
        }
    }
    __syncthreads();
}
```

Appendix C

Sampling-based Method with Various Window Size

In the sampling-based method, we define a window with default size of 100 $\times$ 100 pixels. In this experiment, we compare the results with different window size. In particular, the sampling-based method is applied to the input dataset shown in Figure 3.14(a). However, when the method is applied, the size of the window is changed to get different results for comparison. In this experiment, the size of window varies from 10 $\times$ 10 pixels to 200 $\times$ 200 pixels. The results are shown in the following figure. In the figure, we can see that when the window size is around 100 $\times$ 100 pixels, the results are similar. However, the results are worse if the size is either much smaller or much larger.

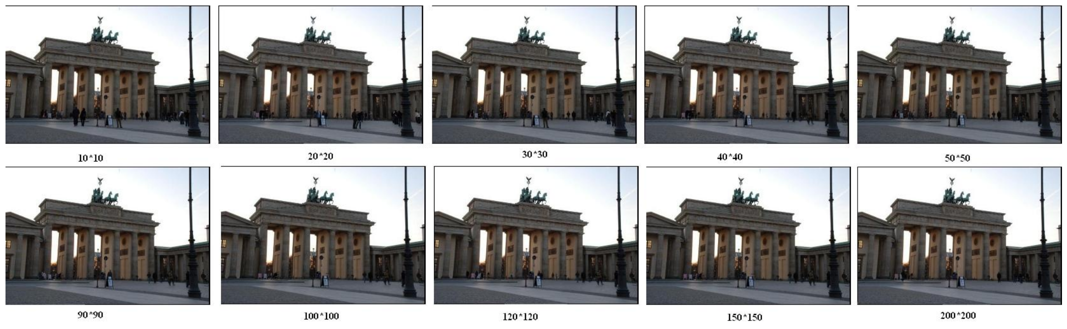


Figure C.1: Comparison of the results by the sampling-based method with different window size.