Urban Scene Parsing via Low-Rank Texture Patches

by

Cyril Lan
S.B., EECS, MIT, 2011

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology May, 2012 ©2012 Massachusetts Institute of Technology All rights reserved

Author: Cyril Lan
Department of Electrical Engineering and Computer Science
May 21, 2012

Certified by: Dr. Yi Ma
Microsoft Research Asia
VI-A Company Thesis Supervisor

Certified by: Professor William T. Freeman
Computer Science and Artificial Intelligence Laboratory
MIT Thesis Supervisor

Accepted by: Professor Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee
Urban Scene Parsing via Low-Rank Texture Patches

by

Cyril Lan

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 2012

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in
Electrical Engineering and Computer Science

ABSTRACT

Automatic 3-D reconstruction of city scenes from ground, aerial, and satellite imagery is a difficult problem that has seen active research for nearly two decades. The problem is difficult because many algorithms require salient areas in the image to be identified and segmented, a task that is typically done by humans. We propose a pipeline that detects these salient areas using low-rank texture patches. Areas in images such as building facades contain low-rank textures, which are an intrinsic property of the scene and invariant to viewpoint. The pipeline uses these low-rank patches to automatically rectify images and detect and segment out the patches with an energy-minimizing graph cut. The output is then further parameterized to provide useful data to existing 3-D reconstruction methods. The pipeline was evaluated on challenging test images from Microsoft Bing Maps oblique aerial photography and produced an 80% recall and precision with superb empirical results.
Acknowledgements

I would like to express my gratitude to those who have taught me, guided me, supported me, and without whom this thesis would not be complete.

First, I would like to thank my mentor, Yi Ma, for imparting onto me many golden nuggets of wisdom. I would also like to thank my groupmates, Zhengdong Zhang and David Wipf, for assisting me with their technical prowess.

Second, I would like to thank my supervisor, Bill Freeman, for his enthusiasm in my work, and his generosity and patience in giving me feedback.

I would also like to thank my Mom, my Dad, and my sister Janice for their continued support during my half a year of research abroad.

I would like to thank Barbara, my true soulmate, who believed in me as I hacked away night and day.

Finally, I would like to thank my friends at MIT, a truly amazing group of people, who have given left me with experiences that I will never forget.
## Contents

1 Introduction ................................. 11

2 Related Research ........................... 12

3 Low-rank Texture Patches ................. 15

4 Automatic Image Rectification .......... 17
   4.1 Algorithm overview .......................... 18
   4.2 Importance mask ............................... 18
   4.3 TILT sampling ................................. 20
   4.4 Evaluation .................................... 21
      4.4.1 Missing rectifications ................. 22
   4.5 Possible extensions ......................... 23

5 Facade Parsing ............................... 24
   5.1 Branch-and-bound Rank Minimization ...... 24
      5.1.1 Graph cut model ........................... 25
      5.1.2 Patch priors ................................ 25
      5.1.3 Neighbor distances ......................... 28
      5.1.4 Graph construction .......................... 29
      5.1.5 Conflict resolution ........................ 30
      5.1.6 Optimizing over $\lambda$ ................. 30
   5.2 Brute force TILT ............................ 32
      5.2.1 Graph model ............................... 33
      5.2.2 Probability mass function of $\tau$ .... 34
      5.2.3 Ad-hoc distance using geometry and texture 34
      5.2.4 Graph cut .................................. 36
      5.2.5 Optimizing over $\lambda$ and $k_2$ ....... 37
5.3 Evaluation and Discussion ........................................ 39
   5.3.1 Branch-and-bound parser .................................... 39
   5.3.2 Brute force TILT parser .................................... 42

6 Facade Parameterization into Structures 46

7 Applications 47
   7.1 3-D modeling and viewpoint transitions ...................... 49
   7.2 Image compression ............................................. 49

8 Conclusion 50

References 52

A Additional successful results from facade parsers 56
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Examples of low-rank textures</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Low-rank textures in real-world images</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Autocorrelation for importance mask</td>
<td>19</td>
</tr>
<tr>
<td>4.2</td>
<td>TILT importance sampling</td>
<td>20</td>
</tr>
<tr>
<td>4.3</td>
<td>Automatic rectification process</td>
<td>21</td>
</tr>
<tr>
<td>4.4</td>
<td>Histogram of TILT results</td>
<td>22</td>
</tr>
<tr>
<td>4.5</td>
<td>Automatic rectification example</td>
<td>23</td>
</tr>
<tr>
<td>4.6</td>
<td>Missing rectification example</td>
<td>23</td>
</tr>
<tr>
<td>5.1</td>
<td>Distribution of $</td>
<td>\theta^*</td>
</tr>
<tr>
<td>5.2</td>
<td>Conflict resolution</td>
<td>31</td>
</tr>
<tr>
<td>5.3</td>
<td>Precision and recall of branch-and-bound scene parser</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>Precision and recall of brute force TILT scene parser - $\lambda$</td>
<td>38</td>
</tr>
<tr>
<td>5.5</td>
<td>Precision and recall of brute force TILT scene parser - $k_2$</td>
<td>38</td>
</tr>
<tr>
<td>5.6</td>
<td>Branch-and-bound scene parser success examples</td>
<td>40</td>
</tr>
<tr>
<td>5.7</td>
<td>Branch-and-bound scene parser failure examples</td>
<td>41</td>
</tr>
<tr>
<td>5.8</td>
<td>Brute force TILT scene parser success examples</td>
<td>43</td>
</tr>
<tr>
<td>5.9</td>
<td>Brute force TILT scene parser failure examples</td>
<td>44</td>
</tr>
<tr>
<td>6.1</td>
<td>Convolution kernel</td>
<td>47</td>
</tr>
<tr>
<td>6.2</td>
<td>Building-finding kernel methodology</td>
<td>48</td>
</tr>
<tr>
<td>6.3</td>
<td>Kernel-based building detection results</td>
<td>48</td>
</tr>
</tbody>
</table>
List of Tables

5.1  Edge weights for branch-and-bound parser graph cut                        30
5.2  Edge weights for brute force TILT graph cut                               36
6.1  Kernel scoring table                                                   46
Urban Scene Parsing via Low-Rank Texture Patches

1 Introduction

Three-dimensional modeling of cities has been an active area of research for nearly twenty years. Fueled by the development of sophisticated sensing and imaging techniques, the exponential rise in computing power, and the advent of crowdsourced projects, 3-D modeling software and platforms have become widely used and now are an integral part of today’s digital world. Some of these popular platforms include Google Maps and Microsoft Bing Maps, which provide users with aerial oblique-view models as well as ground-level panoramas.

The 3-D models used in Google Maps and Microsoft Bing Maps are crowdsourced and are manually built using publicly available software [14] [25]. Naturally, one may wonder whether it is possible for a computer algorithm to automatically create 3-D models of urban areas given image data. The areas of scene understanding, image segmentation, and 3-D reconstruction have all seen very quick advancement. A variety of algorithms have been invented, from point clouds to semantic analysis, to process data from an equally diverse pool of data sources. This thesis explores the various research projects on automated 3-D reconstruction, the methods used, and the types of available data. A new 3-D reconstruction method is proposed that requires almost zero human interaction, an improvement over many current 3-D reconstruction methods today which require some form of human input. Its results and performance are thoroughly analyzed.

The motivation for this project originates from two papers that used internet photo collections to create 3-D models of Rome [9] as well as of various landmarks in the United States [21]. Using hundreds of thousands to millions of geo-tagged images from the internet, photos of the subjects were analyzed to recreate an accurate computer model. These 3-D
models are valuable as tools for educators, city planners, and artists. Granted, these two research papers operate in a slightly different realm from the 3-D city reconstruction methods explored here. Large, user-generated images were used in [9] and [21], while most urban modeling software use specialized data, such as laser altimetry, aerial photos, or LiDAR [27]. But despite these differences, successful 3-D reconstruction projects should serve as an inspiration for what can be made possible with today's advances in computer vision as well as the types of people and organizations that can benefit from using them.

Section 2 discusses related research and the most difficult challenges faced by fully automated 3-D reconstruction. Section 3 explains the concept of low-rank texture patches and how they are used. Section 4 outlines a method for automatic image rectification. Section 5 details the process through which facades in an image are identified. Section 6 talks about how to parameterize the facade parsing results into a data type that is useful for multi-view stereo reconstruction. Section 7 shows possible applications. Section 8 wraps up the project and suggests directions for future research.

2 Related Research

Over the past two decades, two main types of 3-D city modeling have emerged [15]. The first type creates coarse models and operates on large-scale data, typically from aerial, imagery, satellite imagery, or laser altimetry. They focus on reconstructing the "footprint" or the outline of the building, modeling rooftop features, and output coarse, polyhedral meshes. Techniques such as point clouds are often used to find the edges of the building, parameterize its structure, and represent it with polygons. [31] shows an example of a coarse 3-D modeling scheme that uses top-down aerial images and elevation data. In that method, buildings are represented as compositions of rectangles, leaving only the outline of the building to be segmented out. These methods generally require less human work.
The second type of 3-D city modeling creates very fine models, usually over a smaller region on the ground, and uses a combination of low-altitude oblique aerial imagery, ground-based LiDAR, or street-level imagery. These types of data are chosen because they offer higher resolution and better texture quality. These methods are able to produce 3-D models with much finer detail, such as doors, windows, and other facade features. However, they often require large amounts of human input before the computer algorithm can take over. For instance, [37] uses the low-rank rectification of building facades in multiple images to compute very precise camera calibrations, but identification of the low-rank areas still remains an unsolved problem. Another paper [29] proposes a grammar-based method which creates very detailed 3-D building models once the algorithm has been initialized with a segmentation, but the algorithm must be hand-fed the outline of each building.

Earlier methods of fine 3-D urban reconstruction often required even larger amounts of human input for the reconstruction of just a single structure. In [30], segmentation of the different facades of a building from the background is accomplished by having the user click on every single boundary. The user also manually initiates multiple-view matching of images taken from different angles. In [20], a method is proposed for combining textures from ground-based images with shapes from aerial images. The ground-based image is rectified automatically, but the segmentation of facades is again achieved only with user input. Some 3-D reconstruction methods such as [17] are completely reliant on human inputs, making them more similar to 3-D drawing software.

Some progress has been made on the segmentation of street-level images, which typically contain scenes that are topologically simpler (i.e., when driving along a street, it is assumed that nearby building facades are oriented similarly and usually bordered by ground on the bottom and sky on top). Many methods have been put forward for segmenting and re-
constructing street-level images, typically with decent to excellent results [16] [28] [32] [33] [34]. Relative few segmentation methods exist for segmenting more complex scenes, such as what may be produced by oblique aerial photography. Indeed, most ground-level scene segmentation methods make strong assumptions about the structure of the scene, making them difficult to extend to oblique scenes. The only segmentation method without these assumptions is [3], and although it shows a bit of promise, the method lacks extensive testing and its performance on larger or more complex scenes is uncertain.

There are some interesting 3-D reconstruction systems that don’t require salient areas to be segmented and identified. One such example is [1], which poses the problem of 3-D modeling as a model fitting problem. The locations of the ground plane and appropriate vertical planes are found, and the entire image is projected onto a backbone consisting of the ground plane and vertical planes. In [24], the heights of each pixel above the ground plain are solved to create a 3-D model. Lines detected in the image are used to impose geometric constraints on the heights of each pixel, and the best such point cloud is found with dynamic programming. Neither of these methods requires segmentation or rectification of the image, but the tradeoff is that their outputs are not parameterized, meaning that they can produce coarse 3-D models at best. Another dynamic programming method similar to [24] is presented by [11].

Several other noteworthy 3-D reconstruction methods were found. In [12], regions of the image are classified as planar or non-planar. The regions are used to smooth and simplify the rough point cloud that is produced by an epipolar-geometry-based multi-view stereo algorithm. Lafarge and Mallet [19] presented a parameterized point cloud method, which first identifies points that comprise man-made structures and then further decomposes the structures into geometric primitives. An energy minimization problem is solved to achieve the segmentation of man-made structures from foliage and background. Finally, [26] uses symmetry detection to perform both the segmentation of salient areas and the rectification
and reconstruction of buildings. An algorithm first scans the image for repetitive textures, and those textures are used to identify the locations and orientations of building facades.

However, segmentation of more complex scenes remains an unsolved problem, and most 3-D reconstruction algorithms rely on knowing the locations of salient areas before they can commence. This project approaches segmentation of complex scenes using a new method that utilizes low-rank texture patches. Low-rank textures generally only appear on building facades and are useful for identifying those salient areas.

3 Low-rank Texture Patches

This project makes extensive use of low-rank texture patches to identify where low-rank textures exist. This section discusses the mechanics behind low-rank textures and how they are useful.

Textures are typically represented as a matrix of pixel values. If the texture contains repetitive horizontal and vertical lines, its matrix of pixel values will be low rank. Figure ?? shows several examples of textures that are low-rank and that are not low-rank. If a transformation such as a rotation or a skew is applied to a low-rank texture, the resulting texture and matrix of pixel values will typically not be low rank. This is usually the case when the texture is viewed from an angle, which causes a projective transformation to be applied to it. Convex optimization can recover the perspective transformation that returns the texture to its low-rank state [35]. The algorithm is described in full detail in that paper. The perspective transformation that is computed contains information about the relative orientation between the camera and the plane containing the texture.

One critical property of low-rank textures is that they are an intrinsic property of the 3-D
scene and are thus invariant to the transformation that is applied to them. The transformation is applied when a 2-D image is made from a 3-D scene when the camera is at an oblique position. No matter what angle the low-rank texture is viewed from, a projective transformation corresponding to the position of the camera will restore the texture to its low-rank state.

Let $I$ be an $m \times n$ be a low-rank texture, so that $\text{rank}(I) \ll m, n$. Let $\tau$ be the projective transformation on $I$ that results from viewing $I$ from the camera’s oblique position. The texture captured in the 2-D image, $I_1 = I \circ \tau$, is no longer low rank. The transformation $\tau^{-1}$ to restore $I_1$ to its low-rank state can be computed via convex optimization. Figure 3.2 illustrates the results of such a transformation. Notice that regardless of the applied transformation $\tau$, the low-rank texture $I$ remains the same because it is an intrinsic property of the scene. Additionally, the projective transformation that is recovered is unique up to a translational and scaling factor.

In typical images of city scenes, buildings, which contain the most edges, corners, and regular patterns, will contain most of the low-rank textures in the image. The buildings in a single image also tend to be oriented in the same direction, so that their planar facades are pointing in the same direction. This means that very few projective transformations are necessary to rectify many parts of the image. For a rectangular building, only two such transformations
Figure 3.2: Low-rank textures in real-world images. We refer to non-low-rank textures (a) as "unrectified", and low-rank (b) textures as "rectified".

are necessary, one for the frontal face, and one for the side face. This fact is exploited in the methods for automatic image rectification and facade parsing. The image is broken down into small patches, and the low-rank-inducing transformation is found for that patch. By clustering these transformations, the regions of the image that contain low-rank textures can be identified.

4 Automatic Image Rectification

A common model that applies to most urban scenes is the Manhattan World model, which has been explored by many papers such as [2], [6], [7], and [10]. The Manhattan World model assumes that structures in urban settings are primarily oriented in two orthogonal directions. This strong assumption simplifies many computer vision tasks such as vanishing point detection, calculating camera pose, and reconstructing objects in 3-D. In this project, we propose a new algorithm for computing the vanishing points of a Manhattan World
image without needing human input using a RANSAC-based approach with low-rank texture patches. The results of this algorithm are subsequently used by facade parsing algorithms in the pipeline.

4.1 Algorithm overview

The automatic facade rectification process works by randomly selecting patches in the image and computing the low-rank transformation using TILT [35]. Since TILT computes the planar homography that transforms the patch into a low-rank texture, it will yield two of the three vanishing points of the Manhattan World scene. Depending on the part of the image that is selected, each sample returns either the left vanishing point and vertical vanishing point, or the right vanishing point and vertical vanishing point. The results are accumulated on a 2D histogram to identify the most promising vanishing point candidates. Once a candidate reaches a certain confidence, the patches that contributed to that candidate are selected and simultaneously rectified with a batch TILT operation for increased accuracy. The algorithm loops twice to compute the two dominant rectifications in an image.

4.2 Importance mask

Images of urban scenes usually do not contain solely low-rank textures since foliage, roads, rooftops, ground clutter, and sky take up a considerable part of the image. The low-rank transformations of these objects are generally meaningless and do not yield information about the geometry of the scene. To speed up automatic rectification, salient regions that contain only low-rank textures are identified using autocorrelation before TILT sampling is started.

First, we compute the Laplacian of Gaussian of a grayscaled version of the input image. Next, we compute the cross-correlation of each part of the image with its adjacent regions. For any given pixel \((x, y)\) in the image and a radius \(R\) around it, we take \((x \pm R, y \pm R)\)
as the kernel, and \((x \pm 3R, y \pm 3R)\) as the region. The region and kernel are normalized to have zero mean and unit variance. We then compute the 2-D correlation of the kernel over the region. Large values in the 2-D correlation indicate the presence of repetitive structures. If the maximum value in the convolution exceeds a threshold, \((x, y)\) is marked as a salient region. This 2-D convolution is computed every 10 pixels in the \(x\) - and \(y\) - directions. This allows us to determine whether or not each \(10 \times 10\) pixel region is part of a facade.

**Figure 4.1:** Maximum autocorrelation of each region with its neighbors, computed using the Laplacian of Gaussian of the image. A small kernel region is cut out and convolved with a larger surrounding region.

Figures 4.1 and 4.2 demonstrate the use this importance mask. The maximum autocorrelation of each patch with its neighbors is displayed in Figure 4.1, with lighter areas indicating a higher autocorrelation. These light colored areas are thresholded and applied to the original image (Figure 4.2a), restricting the random sampling space to regions that contain low-rank textures (Figure 4.2b).
Figure 4.2: Importance mask of an image. Only patches of (a) the original image that contain low-rank textures are sampled using TILT. Those patches are highlighted by the importance mask in (b).

4.3 TILT sampling

The result from TILT for a single patch gives us an estimate of the two vanishing points that describe the orientation of the patch. This estimate, however, is not very precise because of the small size of the patch. Even a half-pixel offset in a $20 \times 20$ patch translates to roughly a $1.4^\circ$ error in deducing the location of the vanishing points. To improve our estimate, many patches are sampled and the results are accumulated on a histogram, and the final result is computed from the histogram. If we assume that the two vanishing points are very far away, we can approximate each TILT result as an affine transformation instead of a homography, which decreases the dimensionality of the TILT result from 4 to 2 and makes it easier to cluster. In other words, $H_{3,1}$ and $H_{3,2}$ of the homography matrix $H$ are small and the vanishing points depend only on the two affine terms, $H_{1,2}$ and $H_{2,1}$.

For each patch and TILT result, $H_{1,2}$ and $H_{2,1}$ are extracted and bucketed into a 2-D histogram. When the size of a single bucket exceeds a threshold, all the TILT samples from that bucket are selected for a batch TILT run to produce a single, very accurate rectifica-
Figure 4.3: Patches in the image are sampled randomly. Green boxes show the homography produced by TILT. When enough patches produce similar homographies, they are selected for a single batch TILT operation.

4.4 Evaluation

Automatic rectification was tested on a dataset of 112 images of buildings satisfying the Manhattan World assumption. Each image can be rectified in two ways. Out of the 224 total rectified images, the algorithm correctly returned 132, for an overall accuracy of 59%. Each image took on average 2.4 minutes to process.

Images that have adequate amounts of low-rank textures for both rectifications typically were correctly processed. However, if buildings in the image have noisy, weak, or 1-dimensional
Figure 4.4: Histogram of TILT results: patches that have similar vanishing points are clustered together. When the peak surpasses a threshold, all the corresponding patches are selected for a batch TILT operation.

low-rank textures, the rectifications may come out incorrectly. An example is shown in Figure 4.5. The original image of the two Manhattan-world buildings is displayed in Figure 4.5a. There are two dominant rectifications for this image, one that rectifies the left sides of both buildings, and one that rectifies the right sides of both buildings. The left side rectification was computed incorrectly 4.5b, and the right side rectification was computed correctly 4.5c. Notice that low-rank texture areas are much noisier on the left side of the tall building, and almost non-existent on the left side of the short building.

4.4.1 Missing rectifications

A few of the images had low-rank regions that were so weakly textured that they were not picked up by the importance sampling mask (see section 4.2). If every facade oriented in that particular direction had weak textures, then none of those facades would be sampled
Figure 4.5: Example of an image that produced one correct and one incorrect rectification

Figure 4.6: Examples of an image that only produced a single rectification as a result of an inadequate importance mask

and no information about that rectification would be available. An example is shown in Figure 4.6. There are two left-facing facades in the Figure 4.6a, but one of them does not have a strong low-rank texture and the other one is dimly lit and noisy. As a result, the importance mask only picked up part of one facade (Figure 4.6b), which was not sufficient to produce a high enough peak in the vanishing point histogram. In the end, only the strong, right-facing rectification was found in Figure 4.6c.

4.5 Possible extensions

A possible addition, which would immensely help with missing rectifications, is to use the Hough Transform to identify candidate regions that the importance sampling mask may have missed. [2], [16], [20], and [32] all employ some sort of Hough transform to detect lines and vanishing points in their images. It would be interesting to see if the Hough transform,
combined with a clustering heuristic, can recover rectifications that are unrecoverable using only low-rank textures.

5 Facade Parsing

After an input image has been rectified, a facade parsing algorithm segments out regions of the image that contain rectified facades. A general outline of the algorithm is as follows: First, the image is divided into patches and a prior is computed for each patch to determine its likelihood of belonging to a facade. Second, we use the prior information and an energy-minimizing graph cut to produce a segmentation. These two steps are repeated for each of the two dominant rectifications of an image. The two segmentations are then post-processed to resolve conflicts, smooth edges, and remove noise.

Two variations of this algorithm are described in the proceeding sections. The Branch-and-bound Rank Minimization algorithm uses probability priors and a neighbor-distance-weighted graph cut, as carried out in [4]. The Brute Force TILT method uses the TILT output of a patch as a prior and the Markov Random Field Ising model, which was inspired by [36].

5.1 Branch-and-bound Rank Minimization

A new image segmentation model was proposed by Boykov and Funka-lea in [4]. Suppose that we have a background region and a foreground region. Each pixel has an intensity that allows us to compute the prior probability of that pixel belonging to the background or foreground. Additionally, we wish to segment the foreground so that neighboring pixels separated by the boundary do not have similar intensities. For the Branch-and-bound Rank Minimization algorithm, we use the same graph cut model but work with image patches instead of pixels and feature vectors instead of intensities.
5.1.1 Graph cut model

We will briefly summarize the graph cut model from [4].

Suppose that we have an energy function \( E(A) \) that we wish to minimize, where \( A = (A_1, A_2, \ldots, A_{|p|}) \) is a binary vector denoting a segmentation and \(|p|\) is the number of patches in the image. \( A_i = 0 \) if patch \( i \) is background, and \( A_i = 1 \) if patch \( i \) is foreground. Let us define an energy function \( E(A) \) as follows:

\[
E(A) = \lambda \cdot \sum_{p \in P} R_p(A_p) + \sum_{(p,q) \in N} B_{(p,q)} \cdot \delta(A_p, A_q)
\]

\[
\delta(A_p, A_q) = \begin{cases} 
1 & \text{if } A_p == A_q \\
0 & \text{otherwise}
\end{cases}
\]

where \( R_p(A_p) \) is a cost of assigning segmentation \( A_p \) to patch \( p \) and \( B_{(p,q)} \) is the cost of having a boundary drawn between patches \( p \) and \( q \). \( R_p(A_p) \) is computed from prior probabilities of patches belonging to a facade or non-facade region, and \( B_{(p,q)} \) is computed from the probability of two neighboring patches belonging to different regions. \( N \) is the set of all neighbor relations, so that the boundary cost is only be summed over neighboring pixels. We use a 4-neighbor system to solve this problem. \( \lambda \) is simply a weighting factor to emphasize either the importance of patch priors or boundary conditions.

5.1.2 Patch priors

The patch prior term, \( R_p(A_p) \), gives the cost of assigning region \( A_p \) to patch \( p \). If the patch has a rectified low-rank texture, \( R_p(1) \) should be low and \( R_p(0) \) should be high. If the patch has an unrectified texture or contains noisy or smooth textures, \( R_p(0) \) should be low and \( R_p(1) \) should be high.
For each patch, we perform a branch-and-bound test to determine if the patch's texture can be further reduced in rank. Three transformations are tested: rotation, $x$-direction skew, and $y$-direction skew. Rotation is tested from $-\pi/9$ to $+\pi/9$ in 20 equal steps, and skew is tested from $-0.5$ to $+0.5$ in 20 equal steps. Each transformation returns two numbers: the best rotation or $x$-skew or $y$-skew that results in a patch of the lowest rank, as well as the variation in rank between the highest rank patch and the lowest rank patch.

For rotation:

$$
\theta^* = \min_{\theta} |I \circ \tau(\theta)|_* \text{ where } \tau(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \text{ and } \theta = \left[ -\frac{\pi}{9} : \frac{\pi}{9} : \frac{\pi}{9} \right]
$$

$$
\theta^\gamma = \max_{\theta} |I \circ \tau(\theta)|_* \ldots
$$

$$
R_\theta = |I \circ \tau(\theta^*)|_* - |I \circ \tau(\theta^\gamma)|_*
$$

$R_\theta$ is the difference between the "best" rank achievable and the "worst" rank achievable.

For $x$-skew:

$$
s_x^* = \min_{s_x} |I \circ \tau(s_x)|_* \text{ where } \tau(s_x) = \begin{bmatrix}
1 & 0 & 0 \\
s_x & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \text{ and } s_x = [-0.5 : 0.05 : 0.5]
$$

$$
s_x^\gamma = \max_{s_x} |I \circ \tau(s_x)|_* \ldots
$$

$$
R_{s_x} = |I \circ \tau(s_x^*)|_* - |I \circ \tau(s_x^\gamma)|_*
$$
The case for $y$-skew follows similarly. $|I \circ \tau|_*$ indicates the nuclear norm of $I \circ \tau$, which is a good substitute for rank as the nuclear norm is a convex function but the rank is not [35].

Using $F = [\theta^*, R_\theta, s_x^*, R_{sx}, s_y^*, R_{sy}]$, a feature vector is created for each patch. If a patch is part of a rectified facade, it will likely have a strong texture and be fully rectified, in which case $\theta^*$, $s_x^*$, and $s_y^*$ will be zero and $R_\theta$, $R_{sx}$, and $R_{sy}$ will be large. If a patch is not part of a facade, i.e. it is from a non-rectified region, then $\theta^*$, $s_x^*$, and $s_y^*$ will not be zero. Likewise, if a patch is part of a noisy or untextured region, $R_\theta$, $R_{sx}$, and $R_{sy}$ will be small.

Now, we would like to compute $\Pr('facade'|F)$ and $\Pr('not facade'|F)$. An additional training data set was built to model the distributions of $F'|'facade'$ and $F'|'not facade'$. A total of 123k patches were analyzed to create this distribution. A histogram of values of $|\theta^*|$ illustrates how facades, which contain rectified textures, produce much smaller values of $\theta^*$ than non-facades, which contain unrectified textures or noise. Figures 5.1a and 5.1b show this histogram.
Bayes’ formula gives us

\[
\begin{align*}
\Pr('facade'|F) &= \Pr(F|'facade') \cdot \Pr('facade') \\
\Pr('not facade'|F) &= \Pr(F|'not facade') \cdot \Pr('not facade')
\end{align*}
\]

followed by a normalization. At this point, a simplification is necessary due to the large sample space of \( F \) and relatively small number of samples of \( F \) from the training data set. We add the assertion that each component \( F_i \) of \( F \) has an independent distribution, otherwise, our obtained samples of \( F \) may be too sparse to produce an accurate measurement of \( \Pr(F|'facade') \) or \( \Pr(F|'not facade') \). This simplification yields the following

\[
\begin{align*}
\Pr('facade'|F) &= \prod_i \Pr(F_i|'facade') \cdot \Pr('facade') \\
\Pr('not facade'|F) &= \prod_i \Pr(F_i|'not facade') \cdot \Pr('not facade')
\end{align*}
\]

After normalization, these equations give the prior probabilities of a patch \( p \) with feature vector \( F \) belonging to 'facade' or 'not facade'. These probabilities are used to compute \( R_p(A_p) \), which will be used to construct the graph in section 5.1.4.

### 5.1.3 Neighbor distances

The graph cut model also requires a metric for determining the distance between two neighboring patches. A segmentation that cuts two neighbor patches with a small distance incurs a large penalty on the energy function, while a segmentation that cuts two neighbor patches with a large distance does not incur a penalty the energy function.

Consider \( F_1 \) and \( F_2 \) from patches \( p_1 \) and \( p_2 \). We will define a distance metric for two
patches, similar to what was done by the authors of [4], as follows

\[ d_{(p_1, p_2)} = \exp \left( -\frac{1}{2} (F_1 - F_2) \cdot \Sigma^{-1} \cdot (F_1 - F_2)^T \right) \]

where \( \Sigma \) is the covariance matrix of \( F \). Since there are two separate distributions of \( F \), one for facades and one for non-facades, we compute the covariance matrices for both distributions and add them together, weighted by \( \Pr(\text{facade}) \) and \( \Pr(\text{not facade}) \). The term inside the exponential is hence a sort of distance model that penalizes heavily for patches with different feature vectors, but does not penalize for patches with similar feature vectors.

### 5.1.4 Graph construction

To construct the graph, we first define

\[ R_p(A_p) = -\log \Pr(A_p|F_p) \]

where \( F_p \) is the feature vector computed for patch \( p \), and \( A_p = 0 \) indicates that \( p \) is segmented as a non-facade and \( A_p = 1 \) indicates that \( p \) is segmented as a facade. In the notation, \( \Pr(A_p|F_p) \) simply means \( \Pr(\text{facade}|F) \) or \( \Pr(\text{not facade}|F) \) depending on the value of \( A_p \).

Next, we define

\[ B_{(p,q)} = d_{(p,q)} = \exp \left( -\frac{1}{2} (F_p - F_q) \cdot \Sigma^{-1} \cdot (F_p - F_q)^T \right) \]

The graph is constructed in the same way as is described in [4]. A node is added for every patch, and edges are connected between two patches’ nodes if the two patches are neighbors.

In this project, a 4-way, 2-dimensional neighbor system is used. Finally, a source node \( s \) is connected to every patch node, and a sink node \( t \) is also connected to every patch node. Edge weights are assigned as follows in Table 5.1:
Table 5.1: Edge weights for the min-st graph cut of the branch-and-bound scene parser. Edges between neighbors are assigned weight $B_{(p,q)}$, and edges between a node and a source or a sink are assigned $\lambda \cdot R_p$.

Solving the min-st graph cut for this graph will yield the best $A = [A_1, A_2, \ldots, A_p]$ that minimizes the energy function described in 5.1.1. Any efficient graph cut algorithm, such as one described in [5], may be used. A patch $p$ will be labeled as a facade if $A_p = 1$, and not a facade if $A_p = 0$.

5.1.5 Conflict resolution

Two separate graph cuts are run for each dominant rectification, one to identify left-facing facades, and one to identify right-facing facades. Parts of the image that are identified as both need to be resolved, since each facade can only be facing left or right but not both directions.

We propose the following conflict resolution scheme. For any given pixel that is in conflict, we analyze the line passing through that pixel and the vertical vanishing point of the image. If that line passes through a larger number of pixels labeled as left-facing, the pixels in conflict on that line are resolved by assigning them the left-facing label. If the line passes through a larger number of right-facing pixels, the pixels in conflict on that line are labeled as right-facing. Figure 5.2 illustrates this concept.

5.1.6 Optimizing over $\lambda$

One of the input parameters to the graph construction is $\lambda$, which weighs the relative importance of assigning each patch to its most likely category with minimizing the total length of all boundaries. Note that the $\sum_{(p,q)\in N} B_{(p,q)} \cdot \delta(A_p, A_q)$ term has as many non-zero elements
Figure 5.2: A line is drawn through each point in a conflicted region. Two such lines are shown in (a). The side (left or right) with which the line has a larger intersection is assigned as the label for the conflicted point. The conflict in line \( l_1 \) would hence be resolved by labeling the conflicted region as left-facing, and the conflict in line \( l_2 \) would be resolved by labeling the conflicted region as right-facing.

As there are total boundary edges. Because of that, if \( \lambda \) is small, then the graph cut will produce a segmentation that has fewer total boundaries, is smoother, but may have more false positives or false negatives. If \( \lambda \) is large, then the graph cut will produce a segmentation that has more boundaries and is more rough.

To find the optimal value for \( \lambda \), a line search is performed and the precision and recall of the algorithm is evaluated for different values of \( \lambda \). Evaluation is carried out on a dataset of 112 images from Microsoft Bing Maps. Each image is approximately 600 \( \times \) 600 pixels and contains several Manhattan World buildings. Each image also has a manually-labeled ground truth segmentation, as well as two ground truth rectifications. The ground truth rectifications for each image can be computed using the automatic rectification method described in Section 4 so they are assumed to be known and are given to the facade parsing algorithm as input.

Below, Figure 5.3 is shown with the precision and recall of the algorithm as a function of \( \lambda \). Precision is maximum when \( \lambda = 0.2 \), and recall is maximum when \( \lambda = 0.8 \). A good compromise is achieved at \( \lambda = 0.5 \), where both precision and recall are high. The average processing time per image was 4.8 minutes on an Intel 3.4GHz Core i7 machine.
A more in-depth discussion of results from the branch-and-bound rank minimization algorithm is found at the end of Section 5.

5.2 Brute force TILT

Markov random fields have also been used extensively in image segmentation, such as in [8] and [36]. Image segmentation can be formulated as a Markov random field problem by treating the pixels as nodes in an undirected graph. Nodes are connected their 4 neighboring nodes. In the classic Ising model of a Markov random field, each node has two states, $s_i = \{-1, 1\}$, indicating whether the node is segmented as background or foreground, respectively. Each state is dependent on its neighbors, imposing a type of spatial continuity in the segmentation. The background and foreground may have different distributions of pixel
intensities. We can use those two distributions of pixel intensities to compute the probability $\Pr(I|s)$, where $I = [I_1, I_2, ..., I_{|p|}]$ are the intensities, $s = [s_1, s_2, ..., s_{|p|}]$ are the states or segmentations, and $|p|$ is the number of pixels in the image.

Suppose that we wish to find the set of states $s$ that maximizes the probability $\Pr(I|s)$. The most likely state $s$ can be found by minimizing a corresponding energy function using a graph cut [36], [18].

5.2.1 Graph model

The brute force TILT method works similarly to the standard Ising Markov random field segmentation model, but uses image patches instead of pixels and homographies instead of intensities. Hence, each node in the graph will correspond to 1 patch and will be connected to its 4 neighbors. $|p|$ will be the total number of patches. Each patch has a transformation $\tau_i$, which is the transformation that reduces the rank of the patch to the smallest possible value. Note that because this facade parser uses a rectified image as input, patches that are part of facades will have transformations close to the identity, while patches that are not part of facades may have arbitrary transformations.

Our image segmentation problem can now be phrased as follows: Given transformations for all patches, $\tau = [\tau_1, \tau_2, ..., \tau_{|p|}]$, what segmentation $s$ yields the highest a posteriori probability $\Pr(\tau|s)$?

The problem can be solved by creating an energy function from $\Pr(\tau|s)$ and minimizing it using a graph cut.
5.2.2 Probability mass function of $\tau$

From the Ising Markov random field model, the probability mass function of a segmentation $s$ by itself is given by

$$\Pr(s) \propto \exp \left( \sum_{(i,j) \in N} \lambda_{(i,j)} s_i s_j + \sum_{i \in P} \lambda_i s_i \right)$$

where $N$ is the set of neighbor edges and $P$ is the set of patches. $\lambda_i$ gives us a priori information about each $s_i$, and $\lambda_{(i,j)}$ gives us a priori information about the interaction between patches $i$ and $j$ (i.e., should they be segmented into the same group or into different groups). Since we don't have any a priori information, we set $\lambda_i = 0$ and $\lambda_{(i,j)}$ to a constant, $\lambda$.

Next, using the Markov property, we know that given a segmentation $s$, the homographies of each patch are independent of each other. Therefore,

$$\Pr(\tau, s) = \Pr(\tau|s) \Pr(s) = \left( \prod_{i} \Pr(\tau_i|s_i) \right) \Pr(s)$$

$$\propto \exp \left( \sum_{(i,j) \in N} \lambda s_i s_j + \sum_{i \in P} \log \Pr(\tau_i|s_i) \right)$$

To deal with $\log \Pr(\tau_i|s_i)$, we exploit the fact that $\tau_i$ should be similar to identity transformation if $s_i = 1$ indicating that patch $i$ belongs to a facade. $\tau_i$ should be different from the identity transformation if $s_i = -1$ indicating that patch $i$ is not part of a facade. As a substitution for $\log \Pr(\tau_i|s_i)$, an ad-hoc distance function between $\tau_i$ and the identity transformation is detailed in the next section.

5.2.3 Ad-hoc distance using geometry and texture

Consider a patch $p_i$ and its transformation $\tau_i$. Over the small range of a single image patch, we can approximate $\tau_i$ as an affine transformation. Let $u = [\tau_{i(1,1)}, \tau_{i(2,1)}]^T$ and $v =$
\[ [\tau_{i(1,2)}, \tau_{i(2,2)}]^T \] be the two basis vectors of the affine transformation \( \tau_i \). Let \( i = [1,0]^T \) and \( j = [0,1]^T \) be the two basis vectors of the reference coordinate system.

Define \( \text{dist}(\tau_i, p_i) \) as follows:

\[
\text{dist}(\tau_i, p_i) \equiv c_1 \cdot \angle (u, i) + c_2 \cdot \angle (v, j)
\]

where \( \angle (u, i) \) and \( \angle (v, j) \) are the angles between the two pairs of basis vectors. \( c_1 \) and \( c_2 \) are constants that skew the angular distance in favor of the dimension with the stronger texture. The sum of the magnitudes of the gradients of patch \( p_i \) are used to determine whether vertical or horizontal textures are stronger:

\[
\begin{align*}
    h_x &= \sum \left| \frac{\partial}{\partial y} p_i \right| \\
    h_y &= \sum \left| \frac{\partial}{\partial x} p_i \right|
\end{align*}
\]

\( c_1 \) and \( c_2 \) are defined as follows:

\[
\begin{align*}
    c_1 &= \frac{\frac{h_x^2}{h_x^2 + h_y^2}} \\
    c_2 &= \frac{\frac{h_y^2}{h_x^2 + h_y^2}}
\end{align*}
\]

Finally, the distance is passed through a thresholding function:

\[
f(\tau_i, p_i) = -1 + \frac{2}{1 + \exp\left(k_1 \left(\text{dist}(\tau_i, p_i) - k_2\right)\right)}
\]

The thresholding function is such that if \( \text{dist}(\tau_i) \) is larger than \( k_2 \), then \( f(\tau_i) \) will be close to \( +1 \). If \( \text{dist}(\tau_i) \) is smaller than \( k_2 \), then \( f(\tau_i) \) will be close to \( -1 \). The constant \( k_1 \) is the sharpness of the threshold and can be set to any number that is adequately large.
5.2.4 Graph cut

The distance function, with the thresholding function applied on top, is used here in lieu of the log-probability which was used in section 5.1.4.

\[
\Pr(\tau, s) \propto \exp \left( \sum_{(i,j) \in N} \lambda_{s_i s_j} + \sum_{i \in P} f(\tau_i, p_i) \right)
\]

Our goal is to find \( s \) that maximizes this probability. We know that the same result will be produced if we instead minimize the negation of the log. Let us define our energy function \( E(s) \) that we want to minimize:

\[
E(s) = \sum_{(i,j) \in N} -\lambda_{s_i s_j} - \sum_{i \in P} f(\tau_i, p_i)
\]

Kolmogorov and Zabih [18] proved that this type of energy function can always be constructed as a graph and minimized using a min-st graph cut. The graph is constructed by adding one node for each patch and one edge between two patches if the two patches are neighbors. Every node is connected to a source node \( s \) as well as a sink node \( t \). Edge weights are assigned as follows in Table 5.2:

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>((i, j))</td>
<td>(4\lambda)</td>
</tr>
<tr>
<td>((s, i))</td>
<td>(2\lambda + {f(\tau_i, p_i) \text{ if } f(\tau_i, p_i) &gt; 0, \text{ otherwise } 0})</td>
</tr>
<tr>
<td>((t, i))</td>
<td>(2\lambda - {f(\tau_i, p_i) \text{ if } f(\tau_i, p_i) &lt; 0, \text{ otherwise } 0})</td>
</tr>
</tbody>
</table>

Table 5.2: Edge weights for the min-st graph cut of the brute force TILT scene parser. \((i, j)\) denotes an edge that is assigned between neighboring nodes, \((s, i)\) denotes an edge between a node and the source node, and \((t, i)\) denotes an edge between a node and the sink node.

Performing a min-st cut divides the graph into two so that each vertex is connected to either \( s \), or \( t \), but not both. Vertices connected to \( s \) indicate that its corresponding patch is a facade, while vertices connected to \( t \) indicate that its corresponding patch is not.
After the graph cut is complete, some areas of the image may have been labeled as a facade from both graph cuts (recall that one graph cut is performed for each of the two dominant rectifications). Areas in these conflicted regions are resolved using the method prescribed in section 5.1.5.

5.2.5 Optimizing over $\lambda$ and $k_2$

This graph cut model has two intrinsic parameters, $\lambda$, which weighs producing smooth segmentations with assigning patches to their best category, and $k_2$, which decides the threshold between facade and non-facade areas. (In the actual implementation, the $f(\tau_i, p_i)$ term was multiplied by $\lambda$ rather than the $s_i s_j$ term, but the process for optimizing over $\lambda$ is identical.)

We performed a double line search to find the best $\lambda$ and $k_2$. In the first pass, we empirically estimate a good value of $k_2$ to be $k_2 = 0.2$ and perform a line search to find the best value of $\lambda$. Precision was highest at $\lambda = 3$, and recall was highest at $\lambda = 10.5$. Setting $\lambda$ to an intermediate value of 5 yielded the best precision + recall percentage.

Next, we held $\lambda$ constant at the three optimal values of $\lambda = 3$, $\lambda = 5$, and $\lambda = 10.5$ and performed three line searches to find the best values of $k_2$. In each of those three cases, we found that precision was a strictly decreasing function as $k_2$ increased, and recall was a strictly increasing function as $k_2$ increased. The values of $k_2$ that gave the best precision + recall percentages were $k_2 = 0.2$, $k_2 = 0.16$, and $k_2 = 0.12$ for $\lambda = 3$, $\lambda = 5$, and $\lambda = 10.5$, respectively.

Finally, we held $k_2$ constant at the three values $k_2 = 0.2$, $k_2 = 0.16$, and $k_2 = 0.12$ and performed three line searches to find the best values of $\lambda$. We found that for all three values of $k_2$, the best precision occurred at approximately $\lambda = 3$, the best recall at approximately $\lambda = 10.5$, and the best combination at approximately $\lambda = 5$. Such an outcome is desirable.
and indicates that the 2-dimensional functions precision(λ, k₂) and recall(λ, k₂) are well-behaved. Figures 5.4 and 5.5 summarize our results.

It is observed that the precision and recall as a function of k₂ are either strictly increasing or decreasing. If k₂ is decreased, the precision will become strictly better, at the cost of recall. If k₂ is increased, recall will strictly become better, at the cost of precision. On the other hand, there are distinct optimal values for λ if we wish to maximize either precision or recall. This type of variation in k₂ allows for the possibility of identifying uncertain or borderline regions on the graph cut, which can be modified using other image cues such as color, autocorrelation, or Fast Fourier Transform analysis. More details will be discussed in Section 8, regarding future research directions.
The average processing time per image was 25.4 minutes on an Intel 3.4GHz Core i7 machine.

5.3 Evaluation and Discussion

In this section, we will examine in detail the strengths and weaknesses of the two facade parsing algorithms. The success rates, advantages over existing methods, and failure cases will be analyzed.

5.3.1 Branch-and-bound parser

The branch-and-bound facade parser had very few complete misses and consistently identified even very small facades. It performed well on facades that had textures with strong lines. Figure 5.6 below contains the outputs of several successful runs of the branch-and-bound scene parser. Even tiny slivers of a facade, such as the one in Figure 5.6a near the bottom or in Figure 5.6d, are correctly identified. More successful examples are displayed in the Appendix.

On the other hand, the branch-and-bound facade parser is fairly sensitive to noisy textures, textures that do not have strong lines, and general textureless regions. These three causes contribute to a lot of misses from the algorithm. Figure 5.7a shows a test case where the bottom right building has heavy noise and gross reflections on both facades. As a result, only small portions of both facades were correctly identified. Figure 5.7b shows a case where the right-facing facades in the middle of the image are slightly oversaturated, and as a result have weaker line features causing the facade parser to miss those facades. Figure 5.7c shows a generic case where a textureless region is missed.

Conversely, non-facade areas that have strong lines, such as rooftops and the top and bottom edges of facades, are often mistakenly identified as facades. Occasional errors from conflict resolution also contribute to false positives. For example, Figure 5.9d illustrates how the
Figure 5.6: Examples of successes of the branch-and-bound facade parser.
Figure 5.7: Difficult test cases encountered with the branch-and-bound facade parser.
rooftop edges of the center building contaminate the facades of the top building because the
gradient across the boundary is very strong. Figure 5.9e contains some false positives arising
from roads that contain strong lines and resemble rectified textures. Figure 5.9f shows an
error that arose from the conflict resolution step.

In general, the output of this facade parsing algorithm is somewhat noisy, but is able to
pick out fine details and identify the locations of very small facades.

5.3.2 Brute force TILT parser

The Brute force TILT parser has excellent performance on well-textured facades. It also
detects most 1-dimensional textured regions that do not have a stable low-rank rectification,
due to the ad-hoc distance function that evaluates the strength of the texture between two
patches. One-dimensional textures do not incur a penalty for converging toward a vanishing
point in the direction where there is no texture. Figure 5.8 below shows some success cases
of the Brute force TILT parser. Notice that the building on the right side of 5.8a and the
building on the right side of 5.8b have their facades correctly identified, even though the
1-dimensional textures make it difficult to compute the correct rectifications based on local
observations only.

The Brute force TILT parser is also fairly resilient to images that have been scaled. Geotags
from the test images indicated that the aerial images were photographed from a variety of
above-ground altitudes ranging from 600 meters to 1200 meters. Yet, performance remained
consistent independent of how zoomed in the images were. Additionally, this facade parser
also performs well on facades that do not show strong lines, such as the top two buildings in
5.8d. This is a potential advantage over scene parsers that use the Hough transform or use
lines as features. More success examples for this facade parser are displayed in the Appendix.
Figure 5.8: Examples of successes of the Brute force TILT facade parser.
Figure 5.9: Difficult test cases encountered with the Brute force TILT facade parser.
There were several conditions under which the brute force TILT parser would miss a facade. Causes included textureless regions, reflections on facades, and facades that were either too small or irregularly shaped. For example, the result in Figure 5.9a is split in two because the facade parser could not identify the textureless region in the middle. The building on the right in Figure 5.9b is dark in color, causing even slight reflections to significantly change the intensity of its texture. Figure 5.9c shows an example of facades that were missed because they were too small in size, and Figure 15d demonstrates how facades that have complex shapes are difficult to simplify and fit to the Manhattan-world model.

There were also a few conditions under which the facade parser would produce a false positive. The primary cause was from noise that looked similar to 1-dimensional facades, such as roads and overpasses. These textures were often picked up as 1-dimensional facades. Additionally, a few conflict resolution failures also contributed to misses and false positives. Figure 5.9e shows an example where roads aligned with right-facing facades were rectified and labeled because large groups of those patches resembled 1-dimensional textures. Figure 5.9f illustrates another example where the conflict resolution failed on the left-facing facades because there was no "seed" region from which the left-facing facade could have been grown.

We notice that in most cases, even in scenarios where noise, occlusions or non-regularities caused the facade parser to produce incorrect segmentations, the borders of the segmented regions are still very close to the actual borders of each facade. We will use a heuristic to estimate the locations of each building using a box-model system that pairs up left and right facades.
6 Facade Parameterization into Structures

The parameterization of individual facades into higher-level structures such as buildings enables us to perform higher-level vision tasks such as viewpoint transitioning and augmented virtual reality. In this section, we propose a method to estimate the location and dimensions of buildings in the segmented image produced by the facade parser.

<table>
<thead>
<tr>
<th>Pixel Segmentation</th>
<th>Kernel Part</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>Left</td>
<td>+1</td>
</tr>
<tr>
<td>Blank</td>
<td>Left</td>
<td>0</td>
</tr>
<tr>
<td>Right</td>
<td>Left</td>
<td>-1</td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
<td>-1</td>
</tr>
<tr>
<td>Blank</td>
<td>Right</td>
<td>0</td>
</tr>
<tr>
<td>Right</td>
<td>Right</td>
<td>+1</td>
</tr>
<tr>
<td>Left</td>
<td>Top</td>
<td>-1</td>
</tr>
<tr>
<td>Blank</td>
<td>Top</td>
<td>+1</td>
</tr>
<tr>
<td>Right</td>
<td>Top</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table 6.1: Every pixel inside the kernel contributes to the total score of the kernel at a given location. Pixels that are in their correct kernel part add to the score. For instance, a pixel labeled as left facade adds +1 if it’s in the left part of the kernel.

A building generally consists of a left facade, a right facade, and a roof area that contains either noise or is textureless. We use a building-finding kernel that we convolve over the result of the segmentation. The convolution produces higher values as we pass the kernel over locations that more closely resemble buildings. The peaks of the scores will indicate locations of the top corner of each building.

The kernel is split into three parts. For each of those three parts, differently segmented pixels are penalized or rewarded so that a kernel that contains a favorable segmentation pattern will have a favorable score. Table 6.1 displays how this score is assigned. For example, if the left part of the kernel contains 1000 left-segmented pixels, 200 blank pixels, and 100 right-segmented pixels, its contribution to the total score from the left part will be
1000 \cdot (1) + 200 \cdot (0) + 100 \cdot (-1) = 900. The kernel is initially set so that the left, right, and top regions have an edge length of 100 pixels.

Figure 6.1: Convolution kernel

Figure 6.1 shows the kernel that is used with the image in Figure 6.2a. Cyan labels the left area, yellow labels the right area, and magenta labels the top area. When the kernel is convolved over the image in Figure 6.2a, a score is produced for every location in the image. Figure 6.2c illustrates how the locations of the corners of the four box-model buildings coincide with peaks in the convolution score.

Finally, once the number of facades and their corner locations have been determined, we perform a line search over different sizes of the left, right, and top kernel regions to find the best combination of sizes that result in the highest score. For example, if the 100-pixel kernel region was initially too small, extending the kernel in one or more directions should allow it to include more correctly segmented pixels and increase the score. Figure 6.3 displays the result of the size estimation.

7 Applications

The facade parsing pipeline outputs the location, size, and facade orientations of major buildings in an image. There are several applications that can benefit from this pipeline.
Figure 6.2: Figures illustrating the methodology behind the building-finding kernel. The original image is in (a) and the segmentation results are shown in (b). Subfigure (c) shows a color map of the kernel scores over the entire image. Peaks in scores indicate building corners.

Figure 6.3: Results of kernel-based building detection
7.1 3-D modeling and viewpoint transitions

Most automatic 3-D reconstruction techniques in existence today require some form of segmentation to identify salient areas in the image. For instance, in [22], the user must trace out the edges of objects in the scene before the algorithm can construct a 3-D model. In [37], the areas of the image containing low-rank textures are assumed to be known beforehand before the 3-D reconstruction algorithm is commenced. Our facade parsing pipeline presented in this thesis can greatly increase the number of types of data that existing 3-D reconstruction methods can use, as well as reduce the amount of human interaction needed.

An example of a product that can greatly benefit from fully automatic 3-D reconstruction is Microsoft Bing Map's oblique "bird's-eye view". When zoomed in to a certain level over major cities, the map switches to an oblique aerial view of photographs taken from a low-altitude airplane. The oblique aerial views are photographed from only a few cardinal directions, so when the user wants to rotate the camera, the camera jumps by 90° at a time. This makes it difficult to keep track of one's position on the map, since differently angled photos are shot at different times of the day, under different lighting conditions, and sometimes under different white balance. However, if 3-D models of buildings on the map were present, the 3-D models could be used to slowly rotate the image to allow the user to maintain his/her sense of orientation.

7.2 Image compression

Identifying the locations of low-rank textures in an image also allows us to compress the image while exploiting the presence of these low-rank textures. First, it is known that low-rank matrices are highly compressible - one has to only store its top few singular vectors and singular values as opposed to the entire matrix. For matrices with hundreds of rows or columns, this can amount to huge storage savings. Low-rank regions of an image can be extracted and rectified, and the low-rank component can be stored separately from the
rest of the image. The low-rank region can even be separated into a basic, pure low-rank component and noise or sparse error components, as demonstrated in [35]. Such a technique could potentially be integrated into the standard JPEG image compression algorithm to produce an improved signal-to-noise ratio or smaller file sizes.

8 Conclusion

Fully automatic 3-D reconstruction from images is a difficult problem because most algorithms need a segmentation that labels salient regions in an image. In the past, this segmentation task has been left to humans. Human input is accurate and fast for small datasets, but generally infeasible for large datasets such as satellite imagery or aerial photography. We propose a facade parsing pipeline to automatically identify and parameterize the locations of low-rank textures in the image. The outputs from this pipeline can then be used with existing 3-D reconstruction algorithms, eliminating the need for human work.

The facade parsing pipeline works with oblique aerial imagery of urban areas. In cities, a widely adopted geometric model is the Manhattan-world model, in which building facades are oriented in two dominant directions. The Manhattan-world model allows us to compute the two projective transformations to match these two dominant orientations, which is important because those same two projective transformations rectify large amounts of low-rank textures in the image. Once the regions containing low-rank textures are rectified, an energy-minimizing graph cut is used to segment out the low-rank regions. The segmentations from the two dominant orientations are combined and pattern-matched to create box-model buildings from facades. These box-model buildings contain data on orientation, size, and texture, and are immensely useful to existing 3-D reconstruction algorithms.

The performance of the facade parser was good, averaging around an 80% detection rate
and a 20% false positive rate. Empirical results show that almost all facades in moderately challenging images can be identified. In more difficult cases, performance of the facade parser is negatively affected by reflections, weak or noisy low-rank textures, and objects containing lines such as roads that may be mistaken for low-rank textures.

There are several directions in which this project may be extended. First, the automatic image rectification is not very robust to noise or occlusions in images. It would be interesting to see if including Hough transform results, which have been used in more traditional image rectification algorithms, can be added to improve performance. Second, additional cues could be incorporated into the graph cut method. Features such as edges, color histograms, Fast Fourier Transform results, SIFT points, or singular vectors can be used to provide additional information on patch priors and patch boundaries. Graph cut parameters can also be tweaked to create segmentations with different confidence levels. For example, in the Brute force TILT method, a segmentation produced using a low value of $k_2$ will have high confidence for regions segmented as foreground, and a segmentation produced using a high value of $k_2$ will have a high confidence for regions segmented as background. It would also be interesting to see if curved surfaces, which are explored in [23], can be incorporated into the graph cut. Lastly, topological considerations can be added to improve the parameterization of a scene into box-model buildings. For example, [13] uses scene topology to construct ground-level 3-D models.

Several promising applications lead directly from this facade parsing pipeline. Combined with existing algorithms, fully automatic 3-D modeling can be enabled and user experiences with existing 3-D modeling software can be improved.
References


Appendices

A Additional successful results from facade parsers

Branch-and-bound facade parser
Brute force TILT facade parser