3D Rooftop Detection And Modeling Using Orthographic Aerial Images

by

Kunal Khanna

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2013 by the
Graduate Supervisory Committee:

Peter Wonka, Co-Chair
John Femiani, Co-Chair
Ross Maciejewski
Anshuman Razdan

ARIZONA STATE UNIVERSITY

May 2013

ABSTRACT

Detection of extruded features such as rooftops and trees, in aerial images automatically is a very active area of research. Elevated features identified from aerial imagery have potential applications in urban planning, identifying cover in military training or flight training. Detection of such features using commonly available geospatial data for example orthographic aerial imagery is very challenging because rooftop and tree textures are often camouflaged by similar looking features such as roads, ground and grass. So, additonal data such as LIDAR, multispectral imagery and multiple viewpoints are exploited for more accurate detection. However, such data is often not available, or may be improperly registered or inacurate. In this thesis, we discuss a novel framework that only uses orthographic images for detection and modeling of rooftops. A segmentation scheme that initializes by assigning either foreground (rooftop) or background labels to certain pixels in the image based on shadows is proposed. Then it employs grabcut to assign one of those two labels to the rest of the pixels based on initial labeling. Parametric model fitting is performed on the segmented results in order to create a 3D scene and to facilitate roof-shape and height estimation. The framework can also benefit from additional geospatial data such as streetmaps and LIDAR, if available.

ACKNOWLEDGEMENTS

I wish to thank, first and foremost, Professor John Femiani who has been a great mentor and a friend to me. Without his guidance and persisitence this thesis would have not been possible. He always motivated me to come up with fresh ideas, gave me freedom to explore them and pointed them in the right direction. He invested a lot of time and often got his hands dirty doing the work with me as a colleague. I'm truly grateful to him.

I would like to thank Professor Peter Wonka for having faith in my abilities. He introduced me to the field of Computer Vision, Graphics and Gaming. He constantly reviewed my progress and has been a great teacher. I specially thank him for his infinite patience.

I'm grateful to my committee members Professor Anshuman Razdan and Professor Ross Maciejewski for their support. In addition, a thank you to Chia-Yuan Chuang with whom I had several discussions on this project. I'm grateful to all my friends, most importantly Ankur and Shruti who have been like a family away from home these past 2 years.

Last but not the least, I want to thank my mom, dad and brother whose love and support is always with me.

TABLE OF CONTENTS

LIST OF FIGURES

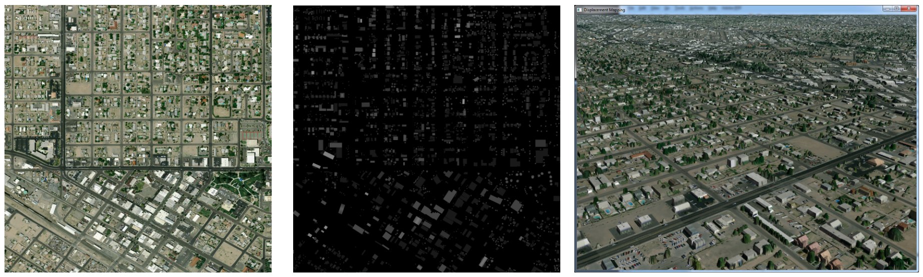
Figure Page

Figure Page

Figure Page

Chapter 1

A FRAMEWORK FOR EXTRACTING 3D FEATURES FROM ORTHOPHOTOS FOR REAL-TIME VISUAL SIMULATIONS



(a) (b) (c)

Figure 1.1: Urban city modeling; (a) aerial image; (b) elevation map; (c) 3D rendering of scene

Three dimensional models of urban environments (Figure 1.1c) find numerous applications. They are used in flight simulators, they are required for urban town planning and architecture, and they are also used in tourism and navigation systems. Urban city modeling is a two step process. The first step is the extraction of height features (such as buildings and trees) in which various geospatial sources such as aerial images (Figure 1.1a), vector layers of road networks, vegetation mask, LIDAR data (Figure 1.2a) and multispectral imagery (Figure 1.2b) related to the region of interest (ROI) are used to detect areas that extrude from the ground. Figure 1.1b shows height features identified in Figure 1.1a. It is a displacement map, which is a grayscale image that encodes height information at each pixel. In a displacement map, black color corresponds to ground (zero height), and white corresponds to the tallest height identified above ground. All the grey values in between represent the corresponding height as a fraction of the tallest height. The second step is rendering in which the height features detected in step one (Figure 1.1b) are used to render a scene in a virtual environment using 3D rendering techniques such as GPU accelerated displacement mapping (Figure 1.1c). In this particular case, first the height information encoded in Figure 1.1b is used to displace the corresponding points in Figure 1.1c by that distance and then Figure 1.1a is used as a texture map to assign colors to the same points. Some of the sources for example LIDAR provide approximate height information at pixel level, but they often have errors such as detection of inconsistent height of pixels within the same roof area. So they cannot be used directly as displacement maps. Others such as vegetation mask provide information about areas covered by grass and trees. Street networks provide clues about where buildings could be located and which direction they might be facing. For example, most buildings face towards a road next to it. All these data sources provide some clues to locating extruded objects in the image and in the past few decades many authors have proposed various methods to use them to identify height features.

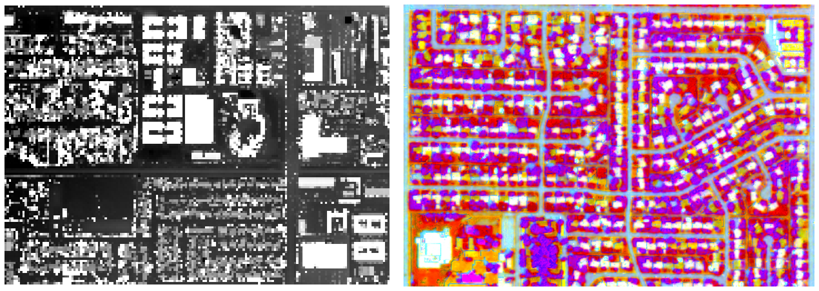

(a) (b)

Figure 1.2: Examples of geospatial data; (a) LIDAR (grey level encodes height above ground); (b) multispectral imagery

Many of the efficient methods proposed in the past require specific data for example LIDAR and multi-view imagery for detection of height features [8], [13], [17], [22], [23], [26]. However, such data is not always available for the target site. On the other hand, methods that do not use additional data and rely on just orthographic aerial images face a lot of challenges in order to detect extruded objects. This is because roofs are often camouflaged by roads, sidewalks and trees are camouflaged by grass. Even if additional data is available, these methods are not able to benefit from them. There is a need for a framework that can adapt to different inputs and take their full advantage.

In this thesis, we propose such a framework and following are its key features:

1. We propose a novel framework that can use commonly available geospatial data for rooftop detection. Only orthorectified aerial imagery of the region of interest is required, but the framework can benefit from additional data, if available. Satellite images are often orthorectified because the earth's surface is curved, due to which the areas captured away from the view direction of camera seem smaller and distorted. Satellite images are flattened so that they can represent the true sizes of these areas. Orthorectification does not affect our algorithm but it is recommended because it represents the areas covered by the images more accurately.

2. We present a novel fully-automated rooftop segmentation algorithm that operates on orthographic aerial images, which is discussed in Chapter 2.

3. We present a novel fully-automated algorithm to model rooftops from a rooftop segmented image along with its original aerial image. This is discussed in Chapter 3.

Chapter 2

SHADOW INITIALIZED GRAB-CUT FOR ROOFTOP SEGMENTATION IN AERIAL IMAGES

2.1 Introduction

In the last chapter, we proposed a framework that uses commonly available geospatial data for rooftop detection. In this chapter, we discuss the first stage of that framework in detail, i.e. rooftop segmentation. This work proposes a novel solution to exploit high resolution color imagery in order to identify rooftops in aerial photos or satellite images. We approach building detection as an image segmentation problem that can be solved with only high resolution color orthorectified imagery. The proposed method is novel and has the following benefits:

1. The method requires only reasonably high resolution aerial image (e.g. 1 meter per pixel). We do not require multiple views, multispectral imagery, LIDAR, or any elevation data.

2. The method can also optionally exploit additional layers of GIS information such as street maps, parcel boundaries, or thematic data if it is present.

3. The method is not sensitive to noise in the image that would negatively affect many edge-based methods, and it does not make assumptions about the shape (rectangular, circular, polygonal, etc) , or colors of buildings.

4. The method does not require training, and has only a few parameters that are simple to determine.

It is important to note that the method does depend on visible shadows and often imagery is collected to minimize the sizes of shadows because they obscure data. In addition, areas of particular interest in applications like flight simulators may have the shadows artificially removed in order to estimate the albedo of the underlying imagery. However, these issues do not pose a significant challenge to the approach because our approach works even when only a small thickness of the shadow is visible (Figure 2.1), and it is very difficult to acquire imagery at a time of day that completely eliminates shadows.

2.2 Prior Art

Many techniques detect buildings by exploiting high resolution height information obtained by LIDAR or stereophotogrammetry ([8], [13], [17], [22], [23], [26]). However, the additional infor-

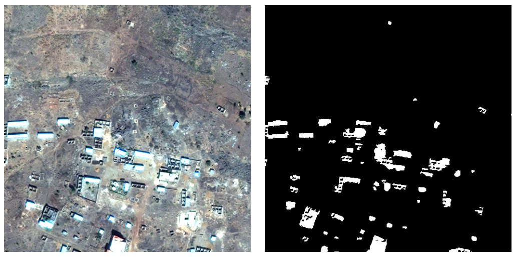

(a) (b)

Figure 2.1: Example showing the result of running our algorithm on an image with very thin shadows; (a) aerial image; (b) rooftop segmentation done using our algorithm

mation changes the problem significantly. The ability of the proposed technique to identify rooftops without height map data is important because archival data often does not include this information, and even new datasets discard this information as imagery is processed to produce orthophotos. For example, one significant application of the proposed work is to generate high-resolution displacement maps for use in flight simulators. Many existing databases have terrain and texture map data without the elevation needed for displacement map, and the proposed approach can identify rooftops using only the texture map.

Several authors have used shadows for rooftop segmentation from aerial images, however shadows are most-often used after an initial building detection step, for its verification and height estimation ([5], [6], [7], [9], [10]). The proposed approach differs from these in that it actually uses shadows to generate building hypothesis rather than as a verification step.

Shadows are used to detect buildings by Sirmace et al. [20], however this work is limited to rectangular buildings. In addition this paper primarily uses colors to identify rooftops, and it uses shadows only to detect cases where it failed to identify buildings based on rooftop colors. For those cases a rectangle fitting method is used to align a rectangle with the canny edges of the image. However, aerial images can often exhibit poor edge quality. Also, rooftops can have strong internal lines like ridges that may be confused with edges (Figure 2.2b). But the grabcut based approach proposed in this thesis does not depend on edge detection. It can handle buildings that are non-rectangular (Figure 2.2a) or, in many cases, rooftops that have internal edges such as gable ridges.

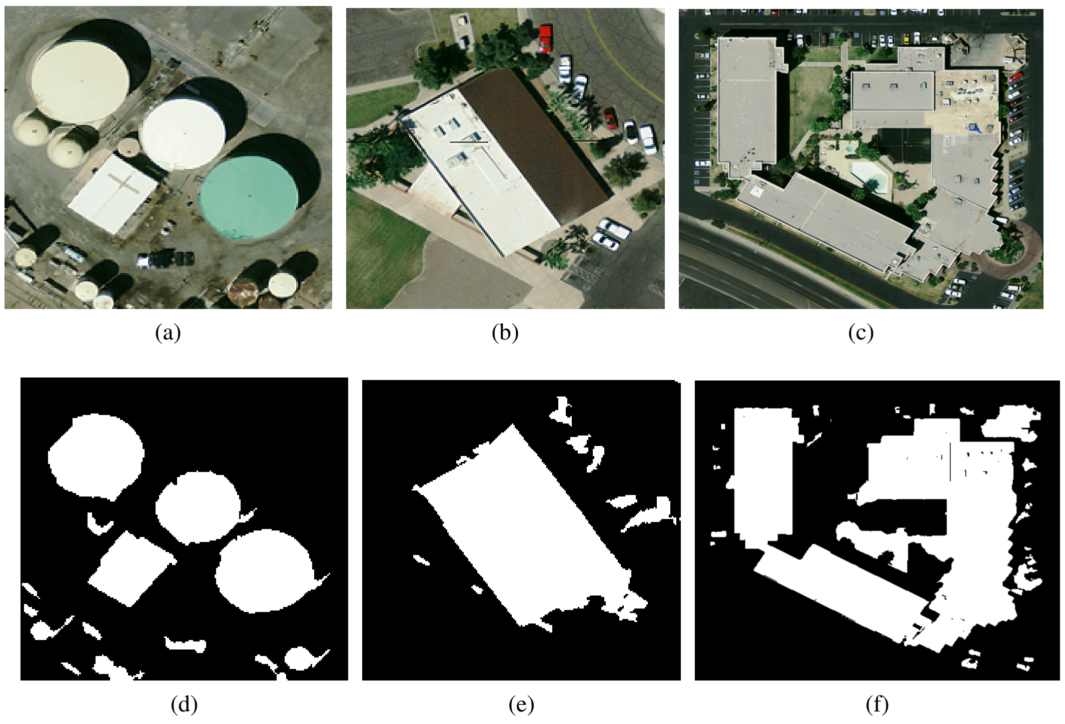


Figure 2.2: Examples of roof shapes that the proposed method can handle; (a) objects with circular or curved roofs; (b) rooftops with internal edges and occlusion; (c) very large rooftops with obtuse angles; (d), (e), (f) show the segmentation results of (a), (b), (c) respectively using the proposed algorithm

Shadows are also used for building hypothesis by Liow et al. [11]. Line segments are first extracted and those segments that lie next to shadows are considered for building hypothesis. A region growing algorithm is used to find other edges of the building. However, gable and hip roofs can have strong ridges (Figure 2.2b) which can be confused with edges of the building, so a region growth algorithm for fitting edges is vulnerable.

Recently, Ok et al. [14] used a method that employed grabcut and shadows to segment rooftops from high resolution imagery. Similar to the proposed approach, shadows were first detected and foreground (rooftop) and background pixels were labeled adjacent to them based on light direction. This was followed by grabcut on a region of interest for each shadow. Unlike the approach we propose, Ok et al. employed a region of interest (ROI) determined by dilation of the shadow component with flat kernels opposite to the direction of light. A certain bounding box around the ROI is determined. Pixels within the bounding box that are identified as shadow or vegetation are constrained to background. Foreground pixels are determined by double thresholding a fuzzy region extended from shadows opposite to light direction. Figure 2.3a shows the shadow region (in blue) and fuzzy region extended from it (in Cyan). Figure 2.3b shows the foreground pixels found after double thresholding the fuzzy region. Our method differs from Ok et al.'s method in several important ways. We run grabcut globally on the whole image and not on a region of interest for each shadow. Thus we are also able to find buildings that don't cast shadows or are larger than the region of interest, such as industrial buildings or warehouses in commercial areas (see Figure 2.2c). This is important for applications involving flight simulators because those types of buildings are common near airports where pilots can see the depth in those building during low altitude flight. We further implement a self-correcting scheme that identifies falsely labeled pixels by analysing the contours of buildings identified by the first pass of grabcut. We re-run grabcut until the segmented results are consistent with shadows.

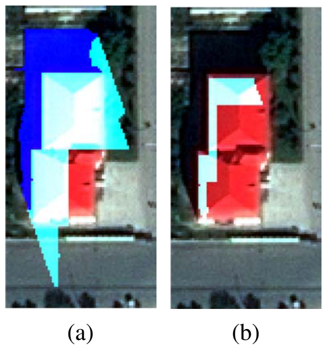


Figure 2.3: Foreground and background assignment reproduced from Ok et al. [14]; (a) shadow (blue) and fuzzy area extended from shadow (cyan) overlaid on aerial image; (b) foreground (cyan) after double-thresholding fuzzy region

2.3 Segmentation Algorithm

The proposed approach for segmenting rooftops is automatic, but it is based on an interactive approach called Grabcut [18]. The grabcut algorithm is initialized by a set of constraints called a trimap created interactively by a user. Users are provided with a sketch based interface that allows them to mark certain pixel labels to foreground or background but leaves most of the pixels labeled as unknown. The grabcut algorithm then iterates between an expectation maximization step in order to fit a Gaussian mixture model to foreground and background pixels and a markov random field optimization step in order to assign labels to the unconstrained pixels using a globally optimal graph cut method [1]. Grabcut is an effective approach for segmenting images, but it can mislabel large portions of the image when the colors are underconstrained by the initial trimap. In an interactive setting a user would look at the result of grabcut and place marks on only a few pixels where the image is over segmented, or under segmented. The user would then repeat grabcut with new constraints, leading to highly accurate results with very little user interaction (Figure 2.4).

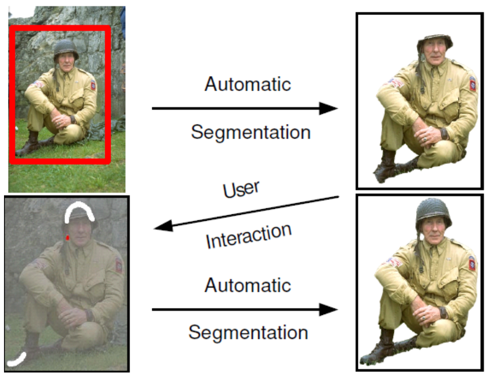


Figure 2.4: Interactive image segmentation with the grabcut algorithm, reproduced from Rother et al. [18]. In the top row the method is initialized only by constraining pixels within the red rectangle to background, in the bottom row the user adds constraints to correct for over and under segmented parts

The main idea of our approach is to provide an automatic process to replace the user interaction in grabcut. We initialize the algorithm based on shadows, and we also add corrections to the results wherever they are inconsistent with shadows in the image until grabcut converges to a segmentation that is consistent with the shadows detected in the image. The key steps of the proposed process are:

1. We generate an initial trimap (Figure 2.5b) using shadow information, along with constraints from other GIS data sources such as street-maps if they are available.

2. We use grabcut to determine the initial segmentation of the image (Figure 2.5c).

3. We analyze the contours of the segmented image and generate new constraints where the contour edges do not cast shadows as expected (Figure 2.5d).

4. Steps 2 and 3 are repeated until convergence. Figure 2.5e shows the result after repeating step 2 and 3 once.

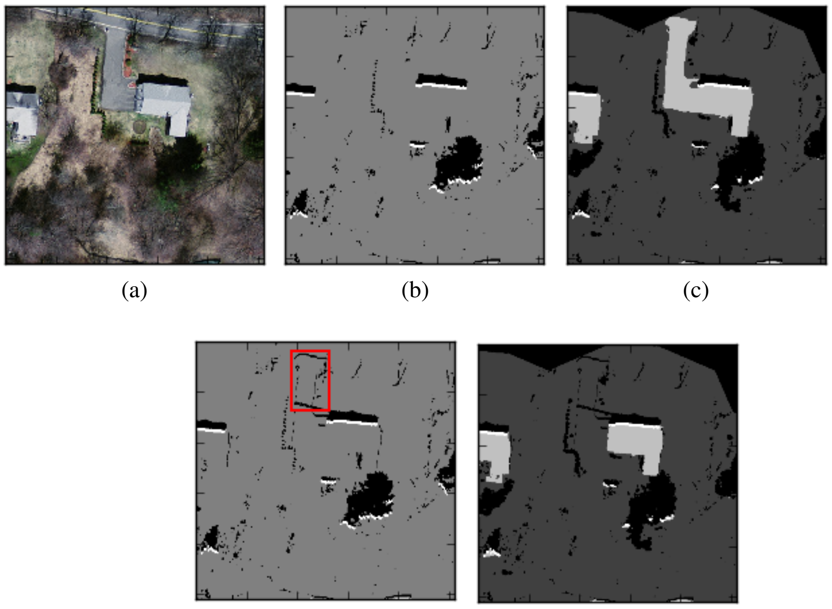

(a) (b) (c) (d) (e)

Figure 2.5: Key steps of rooftop segmentation; (a) aerial image; (b) initial labeling of foreground (white) and background (black) based on shadows; (c) result of running grabcut on (b); (d) additional background constraints added for roof pixels that don't cast shadows (inside the red window); (e) corrected result after running grabcut on (d)

Now we explain each of the steps in detail

1. Foreground and Background Labeling using shadows

A key to the success of our algorithm is to avoid adding constraints to the image except where our confidence in the image labels is very high, and to let grabcut assign labels to the remaining pixels. In our approach shadows are the key. We first detect shadow regions, and use them to place both foreground (rooftop) and background constraints in the image. Shadows are an important feature to indicate depth in aerial images, and a key assumption in our approach is that shadows are simpler to identify that rooftops themselves. There are a variety of applications for finding shadows in aerial photos, for example when the angle of the sun is low enough they can be used to determine the heights of buildings [5], [6], [9], [10], [19]. Shadows are also important because they occlude the underlying imagery. Some authors have developed methods to detect shadows so that they can recover or synthesize the albedo of the occluded part of the scene [21].

We observe that in many images, pixels in shadow are nearly separable from lit pixels based in their intensity or luminance. Shadows show very low luminance compared to their surroundings. We speculate that this may be because outdoor surfaces have a tendency to lighten with exposure to sunlight and weathering. Tsai et al. [21] noted that shadows have a blue-shift in color images. Tsai also noted that shadows have lower luminance because the Electromagnetic Radiation from the sun is obstructed and have increased hue values because the change in their intensity when in shadow and not shadow is positive proportional to the wavelength. Since shadows exhibit low luminance and high hue, Tsai computed $\frac{Hue+1}{Luminance+1}$ ratio image in which the shadow regions showed higher values than others and the automatic threshold selection of Otsu [15] was used to segment the image into shaded and lit regions.

Shadows are not always cast onto flat ground. In fact, quite often there are bushes, trees, cars, or other elements within the shaded region. If any of those extend upwards then they may be lit, and introduce gaps or other artifacts within the shaded region (Figure 2.6a).

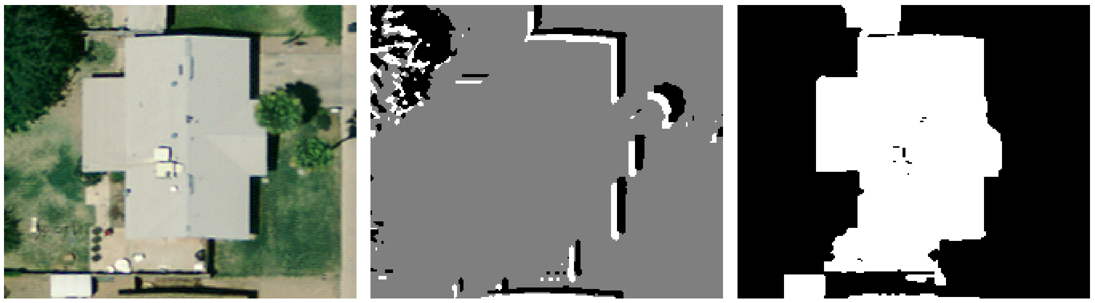


Figure 2.6: Example of successful roof detection when shadow has gaps (notice tree obstructing shadow of building); (a) roof image; (b) initial foreground / background assignment; (c) result of grabcut on (b)

Our method does not require that the shadow contours be complete because any gaps in the shadow may be filled in by the graph cut minimization steps used in the grabcut algorithm. In Figure 2.6, even with broken shadow the rooftop segmentation result is accurate. However, the algorithm doesn't identify the rooftop area occluded by the tree as foreground. We have found that simply thresholding the luminance channel of the aerial image provides sufficient shadows.

Once shadows have been identified we create a trimap for grabcut. We assume that the direction of light is provided for the image. The shadow pixels found are shifted opposite to the direction of light and the resulting locations are labeled as foreground (rooftop). We denote this shift distance by D_F. The optimal D_F depends on the resolution of the image and the angle of light relative to the building contour. It needs to be far enough that some pixels in the image will be constrained as foreground. In [14], a similar approach is employed and significant attention is paid to making sure that the foreground pixels do not extend past the edge of the building. We choose D_F as a small constant (e.g. 4) number of pixels in all cases because it is small enough such that the shifted pixels will not cross the roof on the other side. At the same time it is large enough to assign enough foreground to work for grabcut. The shadow pixels themselves are marked as background. The initial labeling depends a lot on the accuracy of shadow extraction and D_F. Their effects are discussed in the section "Evaluation" later in this chapter. Figure 2.5b shows the trimap generated from shadows on an example image using the method discussed here. Regions constrained to foreground (rooftop) are indicated by white pixels, and regions constrained to the background are indicated by black pixels. The gray pixels are unconstrained and will be determined by grab cut.

Our approach to labeling pixels by shadows makes the simplifying assumption that a scene has two levels of depth - elevated part (rooftops) and non elevated parts (ground). This is not accurate to real scenes. Rooftops vary in height and one rooftop may cast shadows onto another. Many buildings have multiple levels and one portions of the roof may cast a shadow onto another portion of the roof. Rooftops with multiple gables will contain wedge-shaped shadows as indicated in Figure 2.7. In these cases the shadow constraints discussed in this section will introduce errors, but usually these errors are rare and confined to small regions within a rooftop.

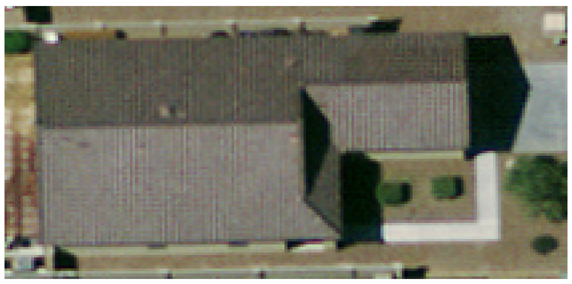


Figure 2.7: Building casting wedge shaped shadow

Another, more significant limitation of our method is the assumption that an edge of a shadow closer to the light source is an elevated structure. This is nearly always the case, but there are two important situations where it is not true. The first occurs for walls or fences, which cast shadows in a scene but are often so thin that the wall itself does not cover a single pixel, as shown in Figure 2.8. As an extreme example, consider the case of an L-shaped wall that casts a shadow. This is completely indistinguishable from a building, even by a human observer. However, walls usually occur as complete loops that enclose an area, and cast an inner shadow that forces grabcut to label the enclosed area as ground. The second problem occurs when an object has overhanging parts that extend outward and are not vertically connected to the ground. Figure 2.9c is a contrived example of this case, which casts a shadow nearly indistinguishable from a building. For most datasets this type of feature is rare, buildings often have rooftops that extend outward in a slight overhang but it is not significant enough to affect our algorithm.

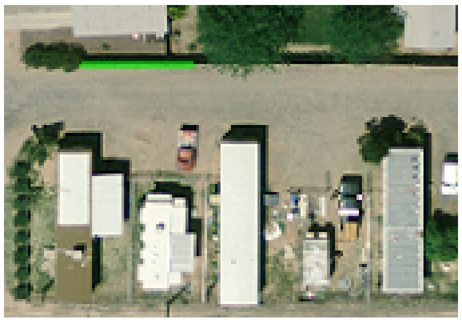


Figure 2.8: Green highlight showing a thin fence casting shadow

2. Detecting foliage to add constraints

Optionally, portions of the image that are unlikely to be buildings are detected, and constrained to the background. This step allows us to exploit geospatial data such as street networks, parcel data and thematic maps to prevent these regions from being labeled as foreground. A vegetation mask covers the region occupied by trees, grass, shrubs. If a vegetation mask is not present, we suppress pixels that have a certain dominance of green in their color. This is because most green colored regions belong to trees, grass, shrubs and green colored roofs are very less likely to be present. We constrain pixels whose green component is more than the larger of red and blue by some percentage which we call Vegetation Coefficient (C_V). The vegetation mask (V) is given by given by 2.1. C_V typically varies from 5% to 30% depending on the image. Here G, R and B stand for green, red and

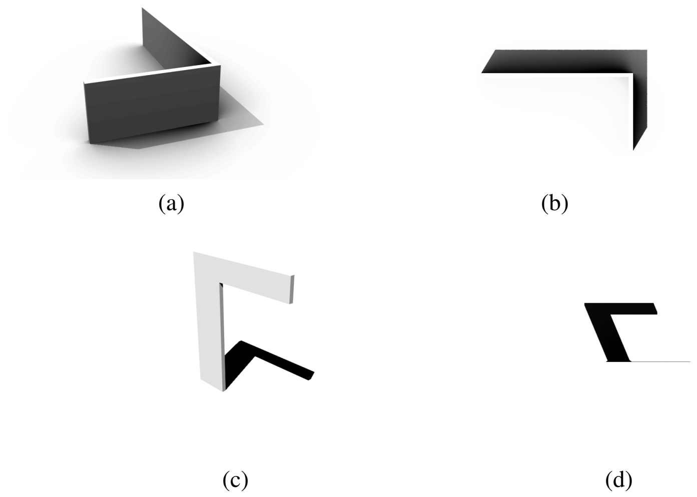


Figure 2.9: Ambiguous shadow cases; (a) a partial wall that casts a shadow; (b) identical to a building; (c) a structure with an overhanging (non vertical) piece that that casts a shadow; (d) nearly identical to a building

blue components of the pixel color.

$$V = G > (1 + C_V) * Max(R, B) \tag{2.1}$$

3. Grab cut segmentation

Grabcut is used to label the remaining pixels in the image. Figure 2.5c shows the result of running grab cut on Figure 2.5b. The unconstrained gray pixels are replaced with either light gray (probably foreground) or dark gray (probably background) pixels.

4. Self-Correction

The grabcut approach to image segmentation may under segment or over segment building contours. When rooftop pixels are mislabeled as background we introduce some errors, but a single rooftop generally does not cover a significant portion of the image and the errors are difficult to notice. Furthermore even if an entire building is not captured, a portion of it will be and that is sufficient for many applications. We are much more concerned about cases where background pixels are mislabeled as rooftop because the background regions are large compared to the buildings and errors tend to bleed out and mislabel large parts of the image. Figure 2.5c shows such a case in which foreground bled into the road next to the rooftop. This happened because the texture of the road is too similar to the roof next to it for grabcut to distinguish. In order to address this issue, Ok et. al [14] introduced a maximum building size and limited grabcut to run within that region. Their approach may fail to identify large rooftops, and in addition, they do not detect cases when pixels have been mislabeled as rooftop and yet remain within their maximum building size.

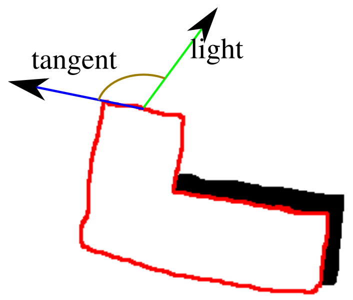


Figure 2.10: Self correction algorithm; red - building contour; black - shadow found; direction of crossproduct of light and tangent points towards the reader, so point must cast shadow. But it doesn't, so it is constrained to background

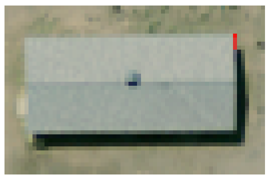


Figure 2.11: Image showing overhang of shadow (notice the red line marked between the corner of roof and point from where shadow starts)

We however address this differently. We don't limit grabcut within a region by assuming a maximum size. Instead, we detect pixels that have been mislabeled as foreground. We analyse the contours of the foreground and find pixels on the contour that are marked as rooftop but do not cast shadows as expected. These pixels are then constrained to the background, and grabcut is repeated until a plausible set of labels is identified. Assuming that the list of points of a foreground contour are arranged in an anti-clockwise pattern, the cross-product of light vector and tangent at a point on contour should be positive if the point casts shadow (Figure 2.10). For all such points, if the pixel next to them in the direction of light is not shadow, then we infer that the point is over-segmented. We label such pixels as background and re-run grabcut with new constraints. Figure 2.5d (compare with Figure 2.5b) shows additional background pixels (inside the red box) added to the image on the foreground contour in Figure 2.5c because shadow next to them was not found. In many cases, shadows usually don't start at the corners of roofs. There is an overhang of some amount and if we neglect that, then certain roof pixels lying close to the corners can be inaccurately labeled as background as apparently they don't cast shadows (Figure 2.11). So we assume an overhang of 4 pixels and mark a pixel as background only if these number of pixels on either side on the contour don't cast shadows either. Figure 2.5e shows the result of grabcut on Figure 2.5d. Comparison between Figure 2.5e and Figure 2.5c shows the correction of result after one self-correcting iteration. The road area labeled as rooftop is removed. This result is consistent with shadows, so no more self correction steps are performed.

5. Pruning misclassified contours by size constraints

The final segmentation may contain very small regions like cars and fences misclassified as rooftops. In order to prune them, we find the contour of each segmented blob, and if the size of its boundary is less than 20 pixels, we remove them. Figure 2.12c shows the segmentation after pruning small contours from Figure 2.12b. A more thorough pruning based on size is done during parametric modeling discussed in Chapter 3.

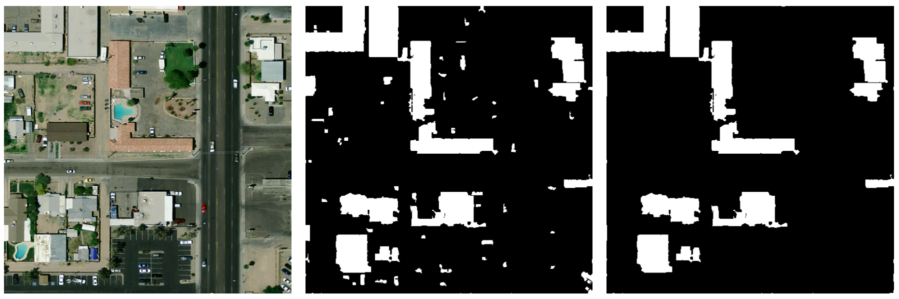


Figure 2.12: Pruning small contours; (a) reference aerial image; (b) rooftop segmentation; (c) rooftop segmentation after pruning small contours

The proposed process requires the following parameters to be specified by a user:

1. The direction of the light within the image plane.

2. The amount of overhang allowed in a rooftop. The eaves of a rooftop affect the shape of the cast shadow, so that the shadow does not always start at the corner of a rooftop but occasionally some

number of pixels away from the corner. The proposed method needs to know the maximum size of the eaves so that it does not mislabel those pixels due to their lack of shadow.

3. A threshold on the image luminance channel, used to identify shadows. Fortunately, shadows in most aerial images seem to be separable based on intensity and this threshold is not difficult to identify if the user can preview the thresholded image.

4. Graphcut iterations to be performed by Grabcut - Grabcut internally performs a number of iterations of graph cut internally. The effect of this is discussed in the section “Evaluation”, later in this Chapter.

2.4 Evaluation

Experimental Results

In order to test the accuracy of the proposed approach, we implemented it in python and used opencv’s implementation of grabcut. Our implementation was not optimized and took 40 seconds on average for each 512x512 pixels square tile using a Dell Precision M6500 laptop. The actual run times varied from 25 seconds to 90 seconds based on the complexity of the tiles. The tiles that have more number of buildings, shadows and trees are considered to be more complex than others. A few images from the dataset are shown in Figure 2.23 continued in Figure 2.24. The images #1-7 have 1 meter / px resolution and consist of a variety of complex rectilinear buildings like “L-shaped” and “T-shaped” buildings. Image #3 also has a building with curved roof. Image #8 has a resolution of 10 meters per pixel. In these figures, first column is the set of input RGB aerial images, second column is the set of results of rooftop segmentation and third column is the ground truth which is manually generated.

We have employed very commonly used evaluation schemes, precision and recall given by 2.2 and 2.3 respectively. The ground truth against which the results are compared are generated manually. TP stands for true positives and refers to the number of pixels assigned as rooftop in both ground truth and segmentation result images in Figure 2.23. FP stands for false positives and refers to the numbers of pixels assigned as rooftop in result but not in ground truth. FN stands for false negatives and refers to the number of pixels assigned as rooftop in ground truth but not in result. Precision gives a sense of what percentage of buildings detected were actually buildings and recall

Test image	Precision (%)	Recall (%)	F-score (%)
#1	76	75	75
#2	85	95	90
#3	85	87	86
#4	84	86	85
#5	90	90	90
#6	87	94	90
#7	97	93	95
#8	83	74	78
Avg.	86	87	86

Table 2.1: Evaluation of Rooftop Segmentation

gives a sense of what percentage of actual buildings were detected. Precision and recall individually hold no relevance because if out of 100 rooftops only 1 is recognized then we will record a precision of 100%, but the recall will be 1%. Similarly, if there are 10 buildings and our algorithm locates 100 buildings such that all the 10 buildings are covered, we will record a recall of 100%. However, precision will be 10%. So, together precision and recall reflect the accuracy of algorithm and better accuracy is observed when both of them are high. Fscore (2.4) captures both precision and recall into a single metric and can therefore reflect the accuracy of the algorithm independently. Table 2.1 shows precision, recall and fscore for all the test images shown. Compared to Ok et al. [14] who show an average of 80.3% precision and 83% recall, our precision and recall has an average of 86% and 87% respectively.

$$Precision = \frac{TP}{TP+FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.3}$$

$$Fscore = \frac{2*Precision*Recall}{Precision+Recall} \tag{2.4}$$

Various parameters drive the segmentation result of grabcut. These are discussed below along with their impact.

1. Graphcut iterations

One of the key parameters is the number of graphcut iterations performed by grabcut each time before returning the results. Figure 2.13 shows the results after using 1, 2, 5 and 10 iterations. With larger number of iterations, larger foreground areas are returned. But after a certain number the results don't show any noticeable increase in the sizes of foreground like after 5 (Figure 2.13d). The number beyond which there is no perceivable change in segmentation should be chosen because that is the number at which grabcut converges. If a lower number is chosen, then its possible that a portion of foreground will not be recovered, as shown in Figure 2.13b and 2.13c. If a higher number is chosen for example 10 in Figure 2.13e, then it slows down the algorithm by running useless iterations of graphcut that don't cause any perceivable change. For our test images 5 turns out to be the optimum number of iterations.

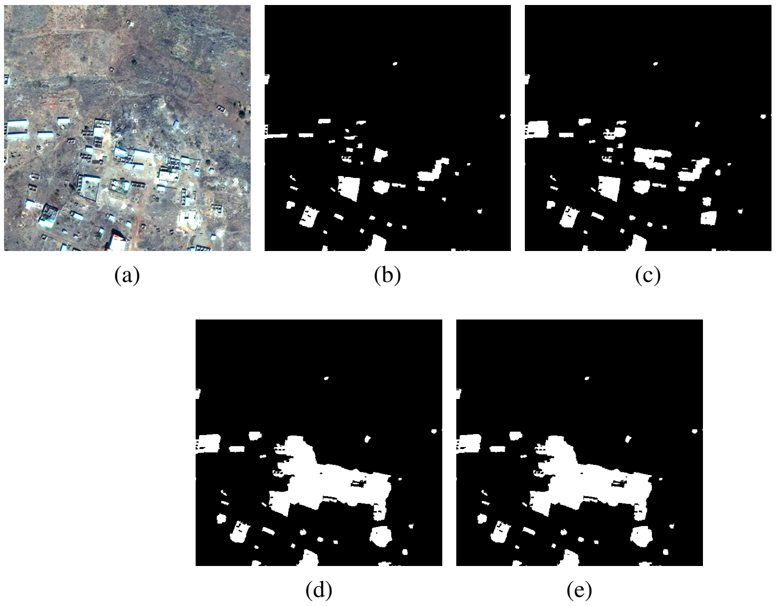


Figure 2.13: Effect of different graphcut iterations; (a) reference aerial image; (b), (c), (d), (e) show the result of 1,2,5 and 10 iterations of graphcut respectively

2. Accuracy of shadow detection

The accuracy of shadow detection plays a very important role because foreground and background are initially assigned on their basis. Figure 2.14 shows the impact of using different thresholding values on luminance channel of image in order to determine shadows. Figure 2.14a shows initial foreground and background segmentation based on detected shadows and Figure 2.14b shows the result of running grabcut on it. The threshold on the luminance channel is chosen as 0.25 on a scale of 1 which means all the pixels with intensity less than 0.25 are labeled as shadows. Figure 2.14c and Figure 2.14d show the same for a threshold of 0.15. As you can see, less shadows are detected in Figure 2.14c because of lower threshold. So less foreground area was assigned on the basis of shadows and thus the final segmentation result has less foreground too. Figure 2.15 shows the variation of precision, recall and fscore with different shadow threshold chosen for the extraction of shadows. As expected, precision drops and recall improves as we increase the shadow threshold because other dark objects are identified as shadows leading to larger areas identified as rooftops.

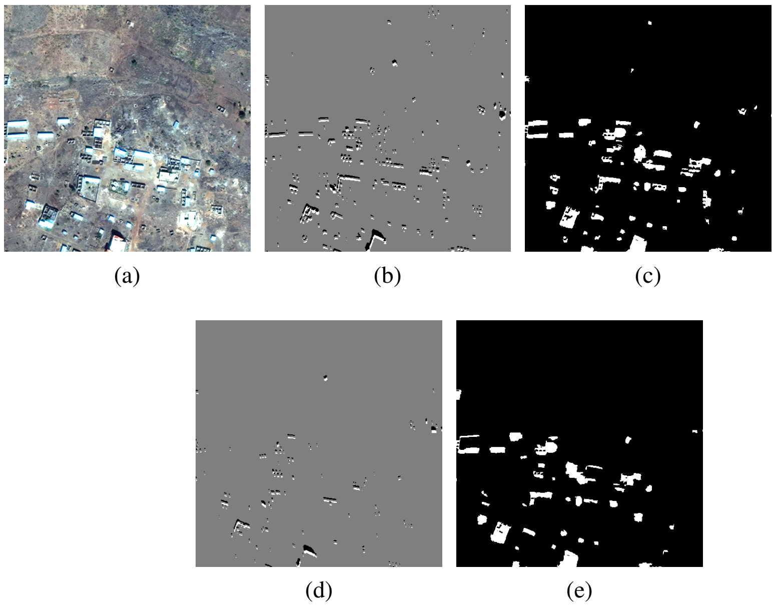


Figure 2.14: Effect of accuracy of shadow detection; (a) reference aerial image; (b), (d) show initial foreground and background labeling using luminance threshold of 0.25 and 0.15 respectively; (c) and (e) show the result of grabcut on (b) and (d) respectively

3. Accuracy of direction of light

The accuracy of the direction of light decides in what direction the shadow pixels will be moved by D_F in order to assign initial foreground and background pixels. Foreground pixels are located by moving shadow pixels against the direction of light, whereas shadows themselves are labeled

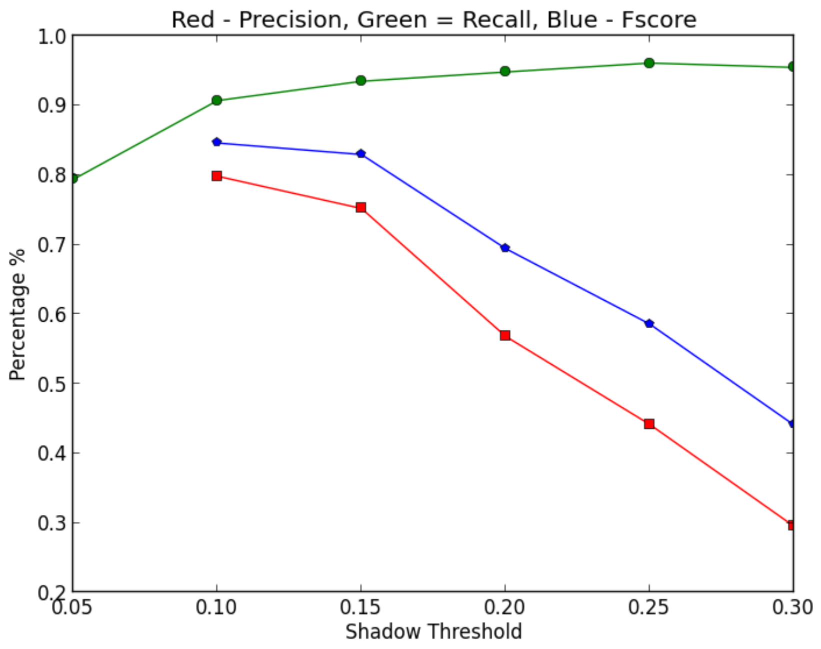


Figure 2.15: Effects of shadow threshold on the accuracy of results

as background pixels. If the light direction is not accurate, then its possible that foreground and background are incorrectly assigned thus causing error in segmentation. Figure 2.16b show initial foreground and background labeling before grabcut for the correct light vector. Figure 2.16c shows the result of grabcut on Figure 2.16b. Figure 2.16d and Figure 2.16e show the same for light rotate by 45 degrees clockwise. So now, the light vector used is significantly different from the actual light. Figure 2.16f and Figure 2.16g show the same for light rotate by 45 degrees anti-clockwise. According to the observation, there is not a very high impact of changing lighting by a small angle. The difference can only be seen in a few buildings after rotating light direction by a high variation of 45 degrees in both directions. We speculate that this happened because one of the two shadow facing edges are enough to reproduce the complete building from grabcut. The resistance to error in light direction may vary for individual images but we conclude that grabcut is quite resistant to small light direction errors. Figure 2.17 shows the variation of accuracy with light rotated from the correct light direction. As we rotate the light direction more, the accuracy decreases. Recall is much more affected than precision because lesser foreground is found as we rotate the light direction more causing grabcut to miss on more bulding areas.

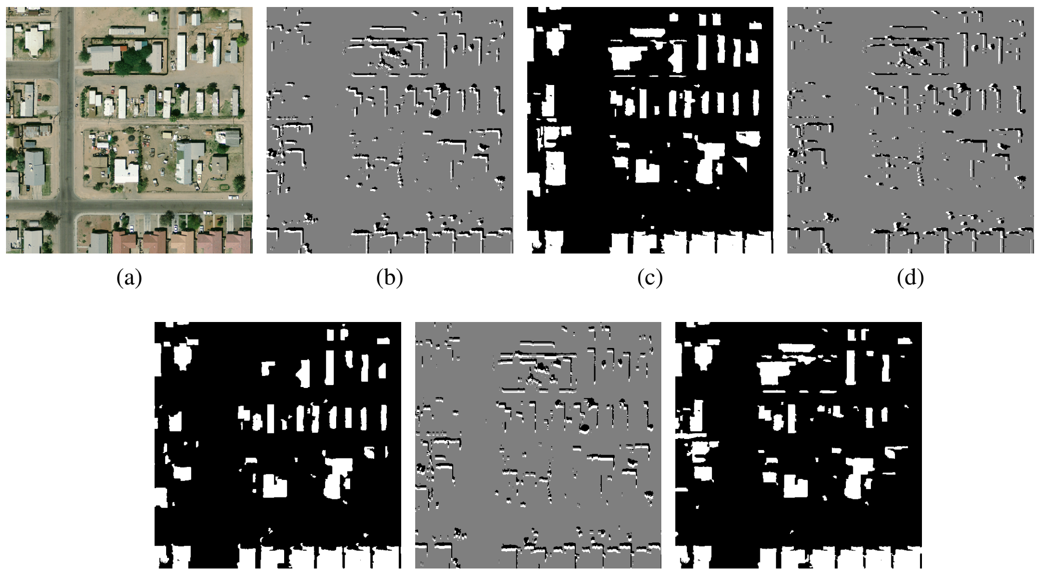


Figure 2.16: Effect of accuracy of light direction; (a) reference aerial image; (b) using correct light direction for initial foreground and background labeling; (c) result of grabcut on (b); (d) initial labeling using light rotated by 45 degrees clockwise; (e) result of grabcut on (d); (f) initial labeling using light rotated by 45 degrees anti-clockwise; (g) result of grabcut on (f)

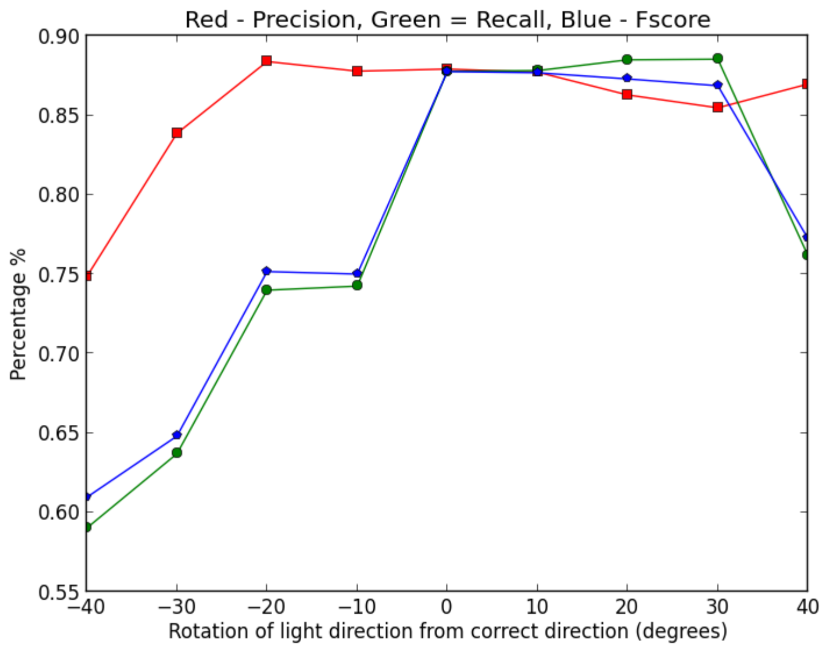


Figure 2.17: Effects of light error on the accuracy of results (x-axis shows light rotate from correct light direction)

4. Accuracy of vegetation suppression

If a vegetation mask is not available, then we create our own vegetation mask by constraining green colored pixels to background. Any pixel that has its green component more than the larger of red and blue by C_V is labeled as background before running grabcut (2.1). This has a huge impact on the segmentation results because, if a tree casting shadow is not suppressed then it will be labeled as foreground and because of it a portion of trees and other vegetation can be labeled as building by grabcut. The optimal percentage of green may vary from image to image between 5% to 50%. It is advised to keep it lower so that no trees are left out. Usually buildings have more dominance of red and blue, so even with a low value it does not impact the building detection in a negative way too much. Figure 2.18 (b)-(g) show three cases in which C_V is chosen as 5%, 10% and 25%. For 5%, Figure 2.18b and Figure 2.18c show initial foreground / background labeling and the grab cut result. The black pixels in Figure 2.18b show background that includes both suppressed vegetation and shadows themselves. Figure 2.18d and Figure 2.18e show the same for 10%. Figure 2.18f and Figure 2.18g show the same for 30%. Notice that the background pixels reduce from Figure 2.18b to Figure 2.18d to Figure 2.18f as we increase C_V. The impact of this is that, more and more trees are labeled as buildings by grabcut from Figure 2.18c to Figure 2.18e to Figure 2.18g. In fact, the actual buildings and trees merge with each other and its hard to differentiate between closely located buildings and trees. Figure 2.20 shows the variation in accuracy with C_V. As we increase C_V, we are forcing the algorithm to detect pixels that are more green. As a result, less trees are suppressed and thus identified as rooftops causing precision to decrease and recall to increase.

5. Effect of shadow shifting distance (D_F) for initial foreground labeling

Grabcut results improve if there are more number of accurate foreground pixels. Figure 2.21b shows initial labeling for $D_F = 4$ pixels and Figure 2.21c shows the result of grabcut on Figure 2.21b. Figure 2.21d and 2.21e show the same for $D_F = 2$. As you can see, some roofs were not detected in Figure 2.21e because enough foreground was not assigned initially to detect them. Based on our tests, we conclude that 4 pixels is a good value to generate enough foreground for grabcut to detect all buildings. Figure 2.22 shows the variation in accuracy with varying D_F (shadow shifting distance for marking foreground). As expected, for low values like 1-2 pixels there isn't enough foreground for grabcut to recover all the complete rooftops leading to a lower recall. As we go

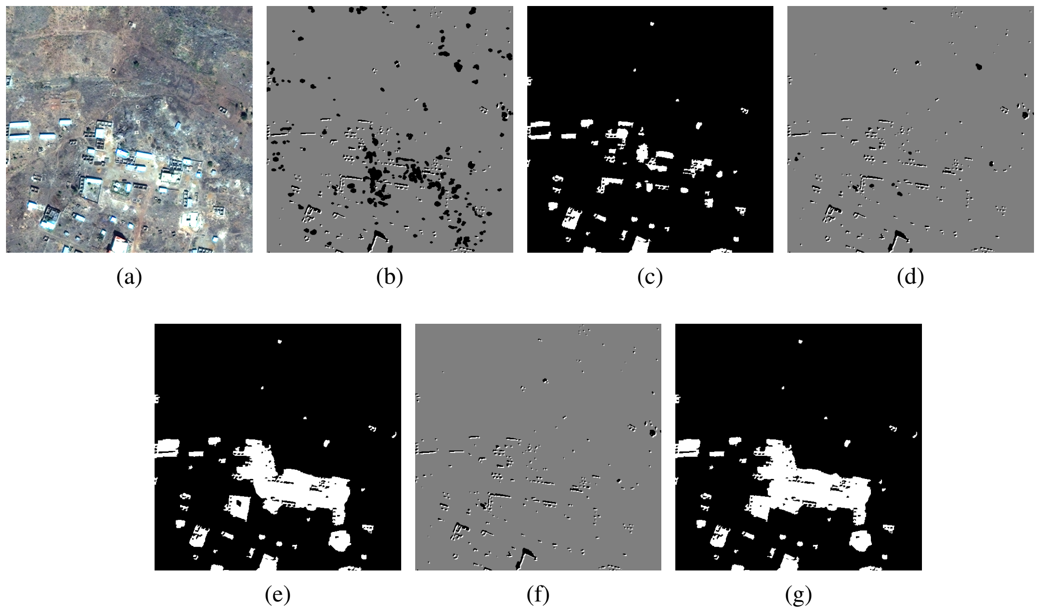


Figure 2.18: Effect of accuracy of vegetation suppression; (a) reference aerial image; (b), (d) and (f) show initial foreground and background labeling with C_V set to 5%, 10% and 30% respectively; (c), (e) and (g) show grabcut result on (b), (d) and (f) respectively

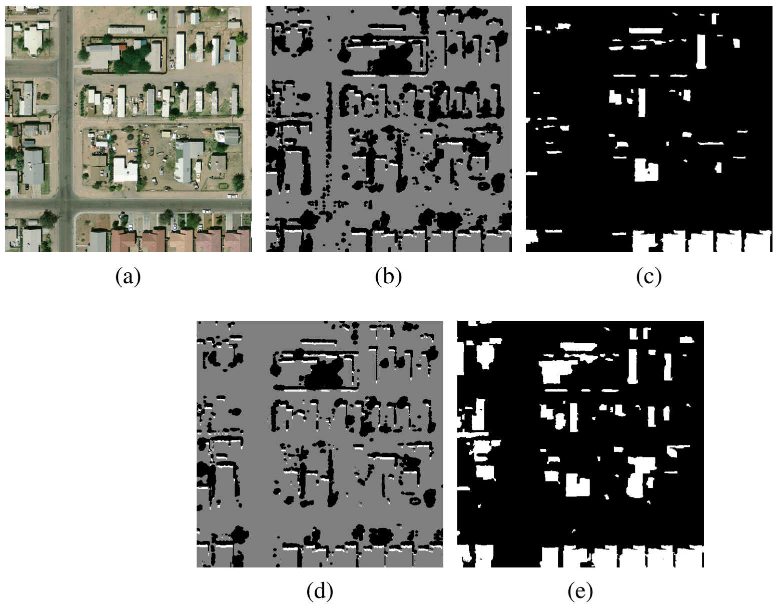


Figure 2.19: Effect of accuracy of vegetation suppression; (a) reference aerial image; (b) and (d) show initial foreground and background labeling with C_V set to 5% and 25% respectively; (c) and (e) show grabcut result on (b) and (d) respectively

beyond 4 pixels precision drops by a little amount as larger foreground is recovered than needed.

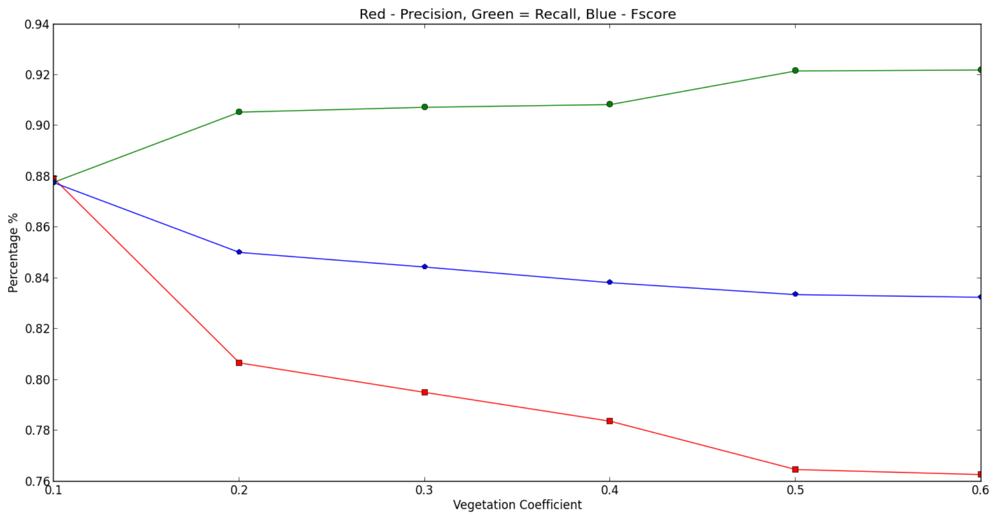


Figure 2.20: Effects of Vegetation Coefficient (C_V) on the accuracy of results

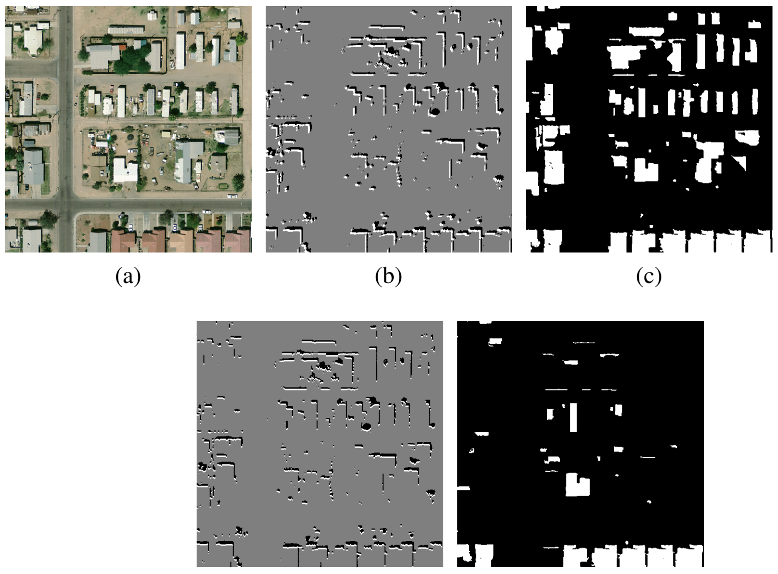


Figure 2.21: Effect of choosing different D_F; (a) reference aerial image; (b) and (d) show initial labeling with $D_F = 4$ and $D_F = 2$ respectively; (c) and (e) show the result of grabcut on (b) and (d) respectively

2.5 Conclusion and Future Work

We've presented a novel approach for segmenting rooftops only using aerial images. The approach doesn't require any elevation or other additional data. It is fully automatic considering that only

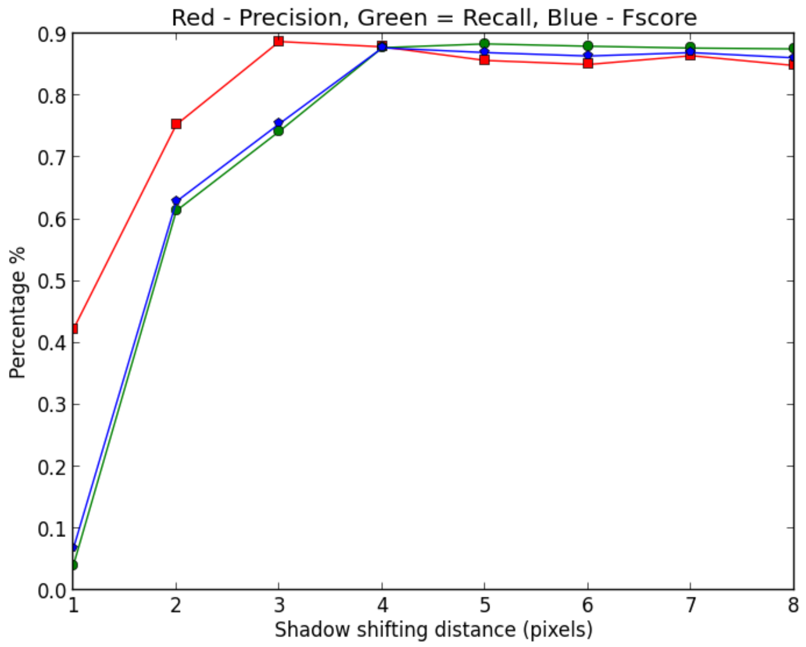
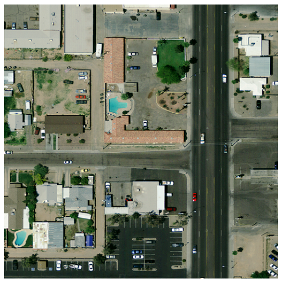
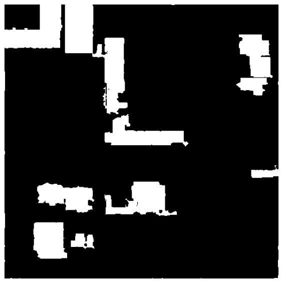
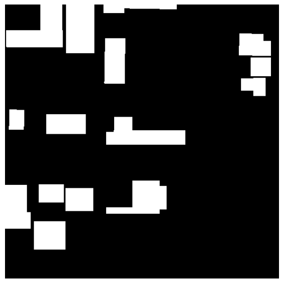


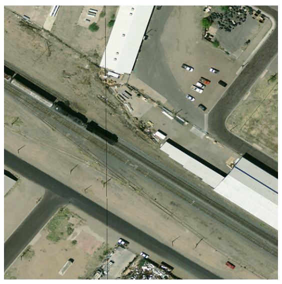
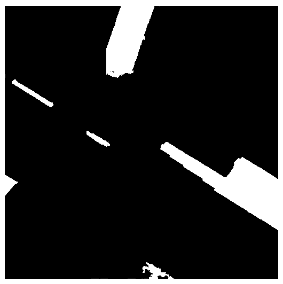
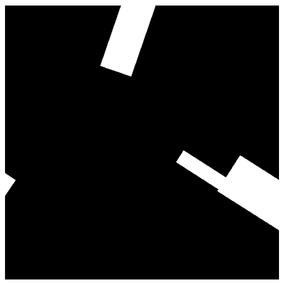
Figure 2.22: Effects of shadow shifting distance (D_F) on the accuracy of results

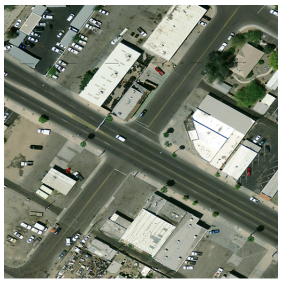
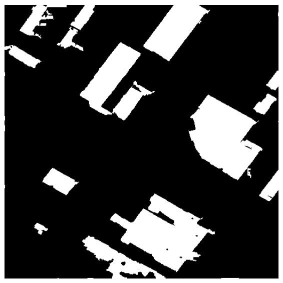
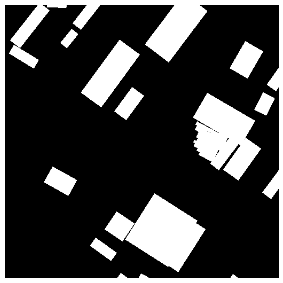
a few configuration parameters are required to initialize the algorithm. It has shown an average precision of 87% and average recall of 85%. The algorithm is sensitive to certain parameters such as shadow intensity threshold and vegetation coefficient (C_V) which need to be set accurately for best detection. But these can be easily tuned. However, there are certain areas where we see a scope for further improvement.

1. For a very large dataset, it is possible that light directions are not consistent throughout the region. We process such datasets in the form of tiles and it would be a good idea to automatically estimate the light direction in each tile using shadows or some other means.

2. As of now, we are manually thresholding luminance channel of aerial images to extract shadows. In future, we would like to implement the algorithm mentioned by Tsai et al. [21] to automatically detect shadows in aerial images.


(a)

(b)

(c)


(d)

(e)

(f)


(g)

(h)

(i)

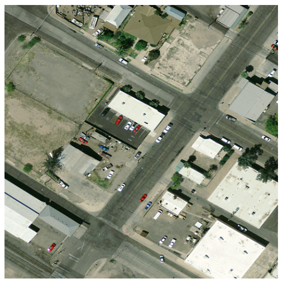
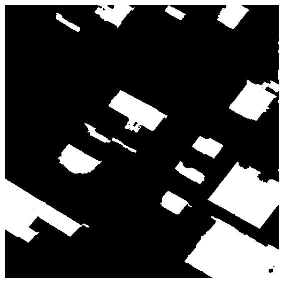
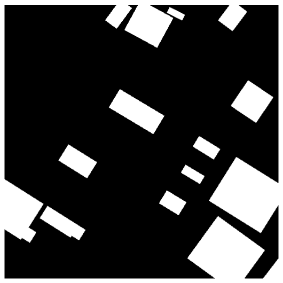
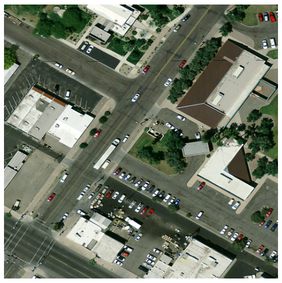
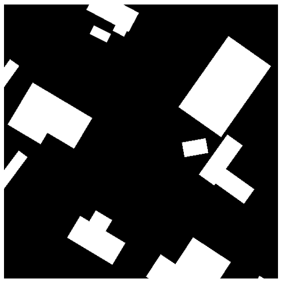
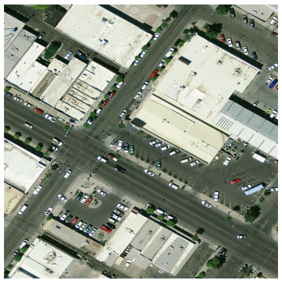
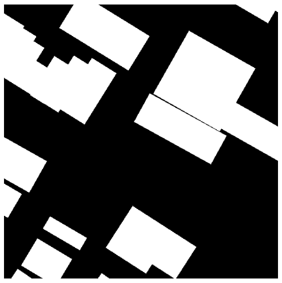

(j)

(k)

(l)

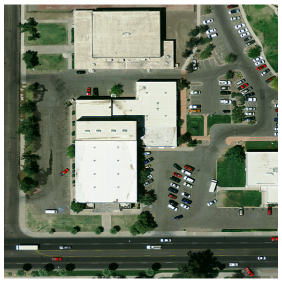
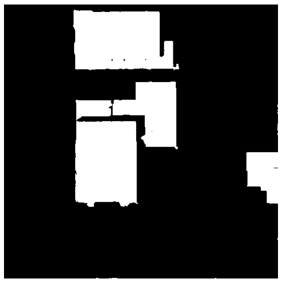
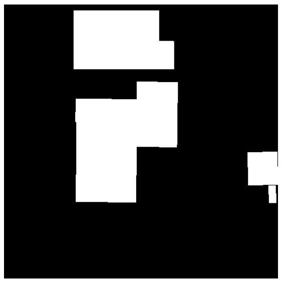
Figure 2.23: (First Column) Test images #1-8; (Second Column) segmentation results for test images #1-8; (Third Column) ground truth for test images #1-8


(a)

(b)

(c)


(d)

(e)

(f)


(g)

(h)

(i)

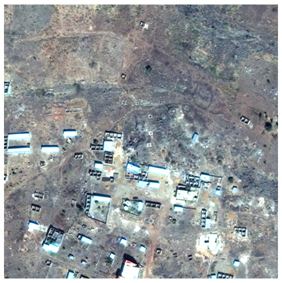
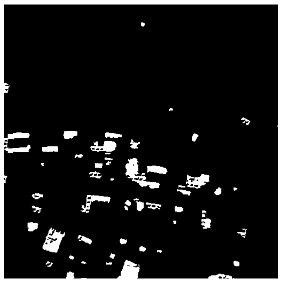
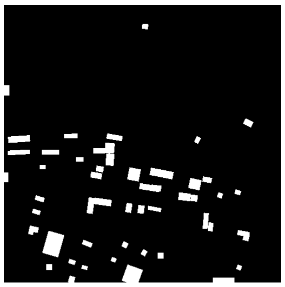

(j)

(k)

(l)

Figure 2.24: Figure 2.23 continued; (First Column) Test images #1-8; (Second Column) segmentation results for test images #1-8; (Third Column) ground truth for test images #1-8

Chapter 3

PARAMETRIC MODELING OF ROOFTOP SEGMENTED DATA IN AERIAL IMAGES

3.1 Introduction

A novel approach to segmenting rooftops in aerial images was proposed in Chapter 2. Our contribution in this chapter is to find 3D models consistent with the aerial imagery. 3D models of urban areas are important for many applications including flight simulators and urban planning. This is because they can model various objects such as trees, buildings, and can be rendered at any resolution. Our contribution is to define a parametric model for buildings in urban areas. We can use it to generate geotypical solutions. Such solutions generate synthetic urban scenes similar to an urban city. But we aim at generating geo-specific solutions that match an existing urban aerial imagery and our building models match the appearance of buildings in the imagery when viewed from nadir. The segmentation results achieved from grabcut in Chapter 2 only provide approximate footprints of buildings. This is because segmentation algorithms are inherently noisy. They don't always make straight or smooth curved edges usually found in buildings. Also, blobs extracted from segmented data may also contain tiny holes which need to be filled. Using them as elevation maps for 3D rendering produces poor quality scenes which are easily noticeable. So, in such cases parametric model fitting is performed that gets rid of noise and can be rendered at any resolution. Also, parametric models of rooftops let us describe roof shapes and heights consistently, which is not possible with segmented blobs. Parametric models are vector data so they also need much less storage space compared to segmented data which is raster. In the next section we discuss prior research done in this area followed by our parametric model description and method of fitting.

3.2 Prior Art

Most of the research on fitting parametric models is done on segmentation achieved by additional sources such as LIDAR. Some of these sources also provides the height information at each pixel, so various types of parametric models have been proposed that can also exploit this information to detect roof-shape. Given a segmented blob, the first step is to perform polygonal approximation depending upon the model. For example, if a rectilinear model is employed, then segmented blobs are approximated to rectilinear polygons (Figure 3.1). After polygonal approximation, the roof of the polygon is modeled.

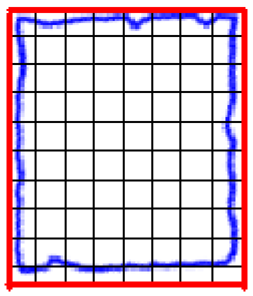


Figure 3.1: Rectilinear polygon (in red) approximated over a segmented blob (in blue) from Matei et al. [13]

Various techniques have been proposed for polygonal approximation. Vestri et al. [23] employed split and merge algorithm [16]. Poullis and You [17] used Douglas-Peucker approximation [2] and then Iterative End-Point Fit [3]. Matei et al. [13] fitted rectilinear polygons at sampled orientations between 0 to 90 degrees on the the boundary of model and computed dominant orientation by using the orientation for which rectilinear polygon fitted has least number of vertices.

In order to model the roof shape, many authors used triangulated polygonal meshes [13], [17], [22], [26]. These models are extremely powerful because they can represent all sorts of complex roof shapes (Figure 3.2). This kind of roof modeling is performed on elevation data like LIDAR because it has detailed height information. However, due to imperfect elevation data, roof shape modeling may show errors (Figure 3.3). To handle such errors Verma et al. [22] used primitives (Figure 3.4b) to approximate the mesh. But, we rely on aerial photography for roof shape detection from which little elevation information can be inferred. Hence we employed a rectangular parametric model that can represent rectilinear flat, gable and hip roofs. Figure 3.4a shows the parametric model for a gable roof. Similar parametric models are employed by other authors [12], [22], [24]. However, they work on elevation data too. A combination of more than one such models can be used to represent buildings with complex footprints such as L, T and U (Figure 3.5) with complex roof shapes exhibiting multiple gables or hips.

3.3 Parametric Model Description

Most of the urban buildings are either rectangle or made up of a bunch of rectangles to form other complex shapes such as L, T and U (Figure 3.5). So we limit our scope only to such buildings. Our parametric model of a building block is rectangular and supports 3 different kinds of roof shapes -

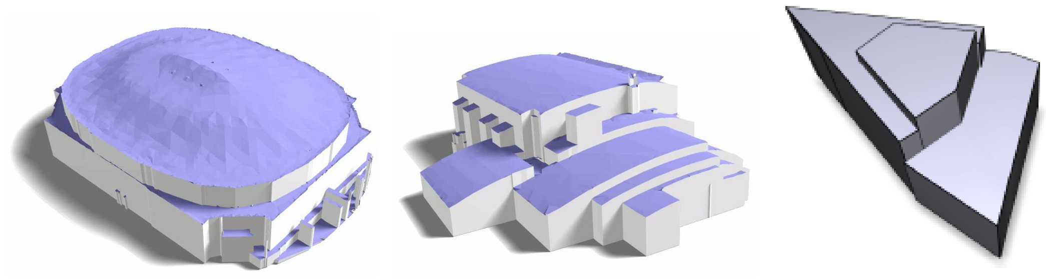

(a) (b) (c)

Figure 3.2: Examples of polygonal mesh modeling to represent arbitrary shaped roofs; (a) and (b) are taken from Zhou et al. [26], (c) from Zhou et al. [25]

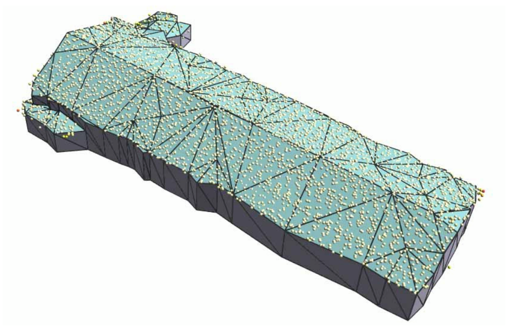
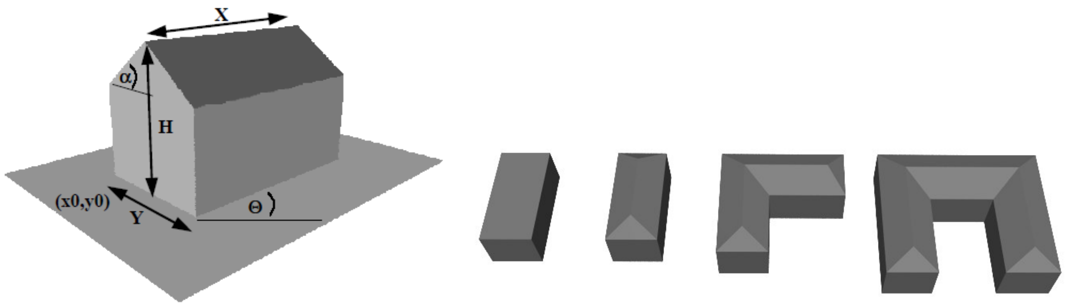


Figure 3.3: Example of triangulated polygonal mesh modeling from Zhou et al. [26]



(a) (b)

Figure 3.4: Parametric models; (a) gable roof model taken from Maas et al. [12]; (b) corner based models taken from Verma et al. [22]

flat, gable and hip. Figure 3.6 shows the image of our model and following are its parameters.

X - x co-ordinate of the center of rectangle in the image

Y - y co-ordinate of the center of rectangle in the image (this increases as we move down in the image)

W - first side of rectangle (width)

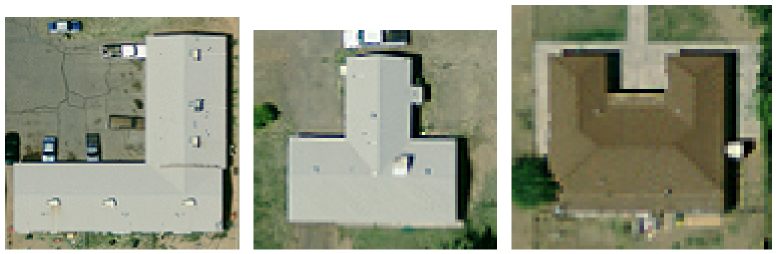

(a) (b) (c)

Figure 3.5: Complex rectilinear roofs; (a) ’L’ shaped; (b) ’T’ shaped; (c) ’U’ shaped

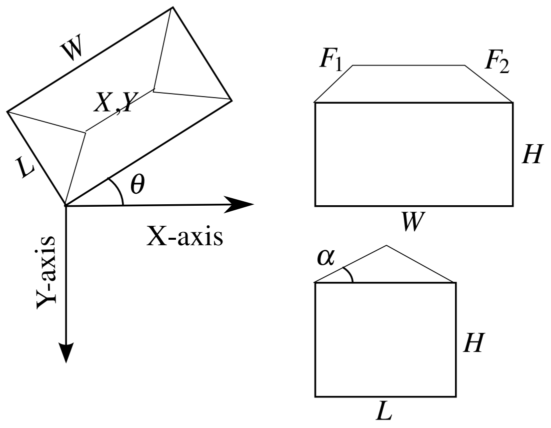


Figure 3.6: Our parametric model for Building Block

L - second side of rectangle (length)

θ - angle (in degrees) between side W and x-axis

H - height of the building above ground

α - angle of gable with ground plane (in degrees). Gable is always parallel to side W.

F_1, F_2 - Offsets of gable from either sides of the building

If α is 0, then it is a flat roof. If α is not 0 and at least one among F_1 or F_1 is not 0, it is a hip roof. Otherwise, it is a gable roof. So using this model, we can describe all the three different types of roofs.

3.4 Model Fitting Algorithm

The key steps for parametric model fitting are:

1. Extract contours of each building footprint from segmented image - Figure 3.10b shows the rooftop segmented image computed for aerial image (Figure 3.10a) after running the algorithm mentioned in Chapter 2. Contour is the set of points running along the boundary of the segmented area. We use OpenCv's implementation of extracting contours from an image.

2. Compute the orientation (θ) of the building with respect to image plane - We do this by applying hough line transformation [4] over the bounding box of contour in the sobel edge image. Hough transformation is a very popular technique that can be used to find lines, ellipses and other figures in an image. We use it to find lines in the image. Hough transformation works by first creating unique bins for all the possible lines that can exist in an image. Lines are represented in their parameteric form (r, θ) where r is the perpendicular distance between the center of the image and the line, and θ is the angle of line segment joining the center of the image and the point on the line closest to it. The parametric form of a line is show in Figure 3.7. θ is sampled across 0-180 at 1 degree and r ranges from $-D_H$ to D_H where D_H stands for half of diagonal of image size. This range of θ and r covers every possible line that can exist in the image. Now, for each edge pixel in the image every possible line passing through it is found and its bin is incremented. In order to find each line, r is computed for every θ from 0 to 180 such that (r, θ) passes through that point. After scanning through all the edge points in the image, an (r, θ) map is generated which has high values for strong lines and less values for weaker lines. This map can be saved as a gray scale image scaled within 0 to 1 with white corresponding to 1 and black corresponding to 0. The bounds of the gray scale image are 2 * D_H x 180 where rows correspond to θ and columns correspond to r. Figure 3.10d shows this hough line transformation image generated for the sobel edge image (Figure 3.10c). The brighter pixels in Figure 3.10d show stronger lines. The angle exhibiting maximum variance in the hough line image is considered as the angle of the building because it shows that most lines in the image are oriented along that angle. The angle with maximum variance is shown by the red line in Figure 3.10d.

3. Compute edge-aligned bounding box - Since we now know the orientation of the building (θ), we

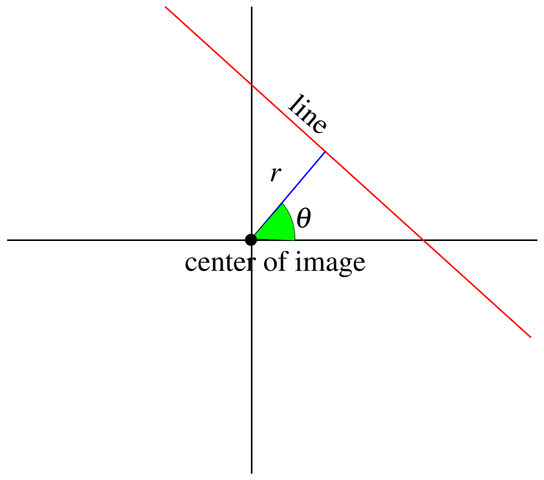
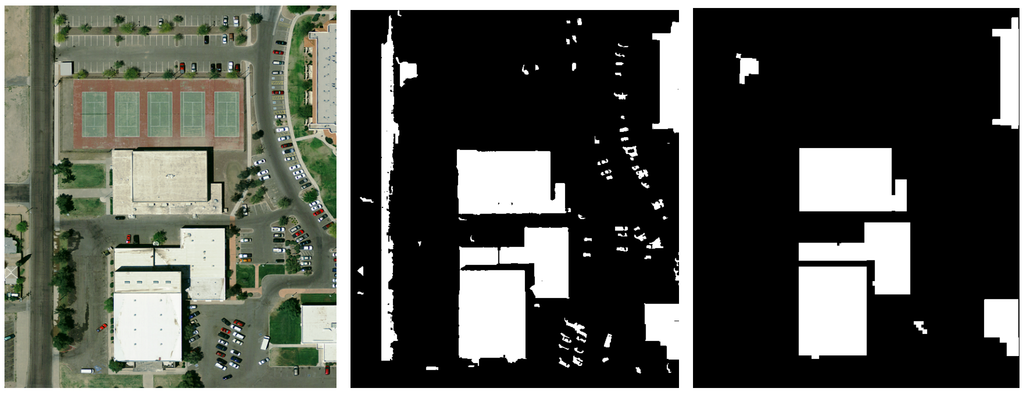


Figure 3.7: Parametric form of a line

compute the edge-aligned bounding box of the contour oriented at θ. This is shown in Figure 3.10e and it is found by first rotating the contour by θ, then computing its bounding box. Then we rotate the bounding box back by $-\theta$.

4. Remove incorrectly identified buildings based on size constraints - If the edge-aligned bounding box is smaller that the minimum size of building, then it is assumed to be a false building and removed. This step gets rid of cars, small bushes and trees that were misclassified as buildings (Figure 3.8). Notice that the cars misclassified as rooftops in segmentation (Figure 3.8b) are pruned after parametric modeling (Figure 3.8c). The building blocks that have very high width-to-height ratios (typically 10:1) are also removed. This is because buildings usually dont have such high aspect ratio. Roads that are mislabeled as roofs usually show this property as they are long strips and hence they are also removed at this stage. Notice that the road misclassified during segmentation (Figure 3.8b) is also removed after parametric modeling (Figure 3.8c).

5. We assume that the buildings are rectangular, but they can be made up of more than one rectangle. They may form complex shapes like “L”, “T” and “U” (Figure 3.5). In order to find these rectangles, we try to find horizontal and vertical lines in the edge-aligned bounding box that align with the edges of segmented building. Two projection histograms are calculated for the horizontal and the vertical direction by finding out the number of edge pixels lying on every horizontal and vertical line within the bounding box. The edge pixels here refer to the outline of the contour. We do not use the edges from the aerial image because edges in it could come from gables or other objects that also



(a) (b) (c)

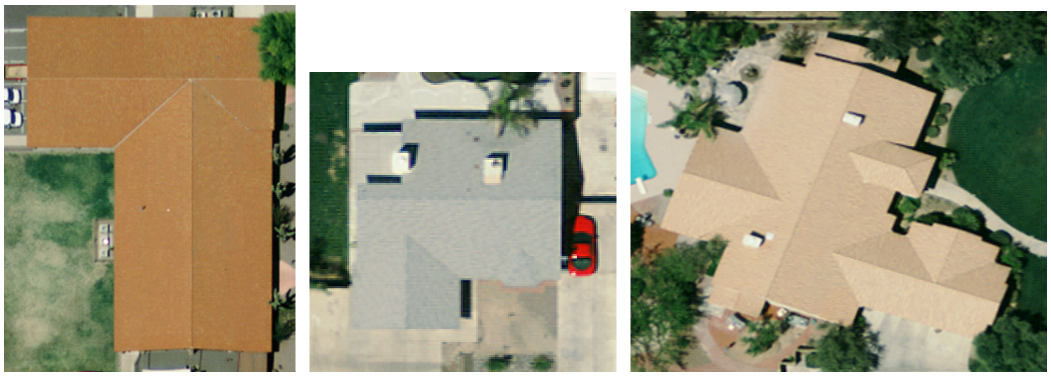
Figure 3.8: Pruning misclassified rooftops based on size constraints; (a) reference aerial image; (b) rooftop segmentation using Grabcut; (c) parametric model fitting (notice that the road on left and cars are removed because of small size and high aspect (width to length) ratio respectively)

lie within the bounding box. The lines for which number of edge pixels lying on it are greater than a threshold (minimum length of building is a good value) are noted. It is possible that two or more closely located lines belong to a strong edge. So we may end up locating more lines than there already are. To fix this, we regroup closely located lines into one line by picking the middle one. In a sense, we want closest lines to be separated by at least a certain distance (like, minimum length of a building employed). Figure 3.10f shows the line segments found using projection histogram.

6. The edge-aligned bounding box is divided into cells based on horizontal and vertical lines found. For each cell, if the number of pixels classified as building lying within it are less than a certain fraction of total pixels (area of cell), it is removed. The fraction that we have used is 50% because we consider that a cell has equal probability of being a part of building or not. If less than half of the pixels in a cell are labeled as building then it is removed. Figure 3.10g shows the remaining cells. In this step we get rid of rectangular areas that lie outside the building within the edge-aligned bounding box.

7. Remaining cells are combined to form larger cells such that the largest rectangular cell is preferred first (Figure 3.10h). This is done because we assume the buildings are made up of different rectangular pieces. So naturally the biggest sized rectangles should be preferred first otherwise we may end up with smaller pieces with same sizes next to each other which should be combined together as one. This helps in representing the building area with the least number of rectangles.

8. The cells are extended into each other till they reach the center of the other cell such that the total area occupied by cells doesn't change. This is done because in most cases, building pieces are not placed next to each other but connected together through gables. Figure 3.9a shows a building with two wings that run into each other till their gables meet. Sometimes they don't extend all the way to the center (Figure 3.9b and Figure 3.9c) and detecting that is something we have left for future work. Thus, we assume that all building blocks meet at the gable of each other. For flat roofs, this doesn't make any difference. Figure 3.10i shows the extended cells. These cells become our final building block.



(a) (b) (c)

Figure 3.9: Different challenging gable interactions; (a) shows a roof with two pieces meeting at the ridge of gable; (b), (c) show roofs where individual pieces dont meet exactly at the gable ridge of each other

9. Finding roof shape - The next step is to find the roof shape of each building block. The roof can be flat, gable or hip. In order to determine if a roof is gable or flat, a hough line in the edge image (Figure 3.10c) at its center perpendicular to one of the two sides (central ridge) is detected. The presence of the line is enough to conclude if its a gable or flat. But it is tricky to estimate the gable angle. Ideally, the difference between the brightness on both sides of gable should be used to figure out the gable angle. The higher the difference, the larger the gable angle unless the light is directly overhead in which case there will be no difference. But we don't know the precise color of roof and ambient and diffuse coefficients. This makes it hard to determine the gable angle precisely. So we choose the intensity of hough line corresponding to the gable to determine the gable angle which is scaled between two permissible extremes. The more intense the gable line, the higher the gable angle. This is not very accurate either, but we do not concern ourselves with gable angle's accuracy

because we aim at rendering plausible roof models. An error in gable angle is not perceivable.

10. If its a gable roof, then hough lines at the four corners of building block are detected that pass from the corner. If a line is present, then it is extended to meet the central ridge. The point at which it meets gives the parameters F_1 and F_2 for the model (Figure 3.6). If either F_1 or F_2 is not 0, the roof is a hip. Figure 3.10j shows the roof shape determined by step 9 and step 10.

11. Determining roof height above ground - We only calculate the height of one building block in a set of building blocks belonging to a building. We choose the one located farthest from light direction. This is because in cases where multiple building blocks are located next to each other, the shadows cast by ones lying closer to the light is obstructed by others. So it is not possible to find their heights using shadows. So we find the height of the shadow facing building block and assume that the height of all building blocks is same. The shadow facing edges of the shadow facing building block are extended until they cover a low intensity region (shadowed region) completely. The edges are extended by 1 pixel at each step, and the pixels on the edge are thresholded to a pre-estimated low luminance to capture shadow pixels. Ideally all the pixels should lie in shadow but many times there are trees or other objects next to buildings that obstruct shadow this causing it to break (Figure 2.6). Also, shadows extend in the direction of light and not in the direction perpendicular to edge. So as we move the edge, more and more pixels at one of the corners of edge will lie outside shadow. We consider an edge to be in shadow if at least 60% of pixels on it are in shadow. We extend the shadow facing edges until they have less than 60% pixels lying on them determined in shadow. If the building footprint determined is correct and the edges of model coincide with the actual building, then this works extremely well because after extending the shadow facing sides, at some point the pixel luminances suddenly rise as the edge goes beyond shadow. This distance by which the edges are displaced gives the thickness of shadow which is used along with the known light direction to compute the height of the building block. The height of all the building blocks is then set to this value. Figure 3.10j shows a green boundary at the top right corresponding to the shadow casted by the model. It matches very well with the underlying low intensity shadow region. Figure 3.10k shows the side view of the larger building block constructed after determining height and roof shape.

12. Using elevation data to determine roof shape - If LIDAR or any other elevation data is available, we can skip step 7-11 and instead use that to determine the roof shape and height.

The proposed process requires the following parameters to be specified by a user:

1. Setting threshold for projection histogram to capture horizontal and vertical lines within the oriented bounding box of contour. The minimum size allowed for a building block in pixels is a good value because it means that at least that many pixels should lie on a line for it to be considered a valid edge. This will also make sure that the size of the smallest cell is not more than the minimum size of building set by us.

2. Minimum fraction of pixels classified as building in a cell that allow the cell to be retained in step 5. Typically 50%.

3. Permissible gable angle range in step 8. Typically between 15 and 60 degrees.

4. Permissible range of building height. Clamped to this range if computed beyond it because of errors. We use 10 pixels to 100 pixels and most buildings in suburban environment are within this range. These may be varied for different locations.

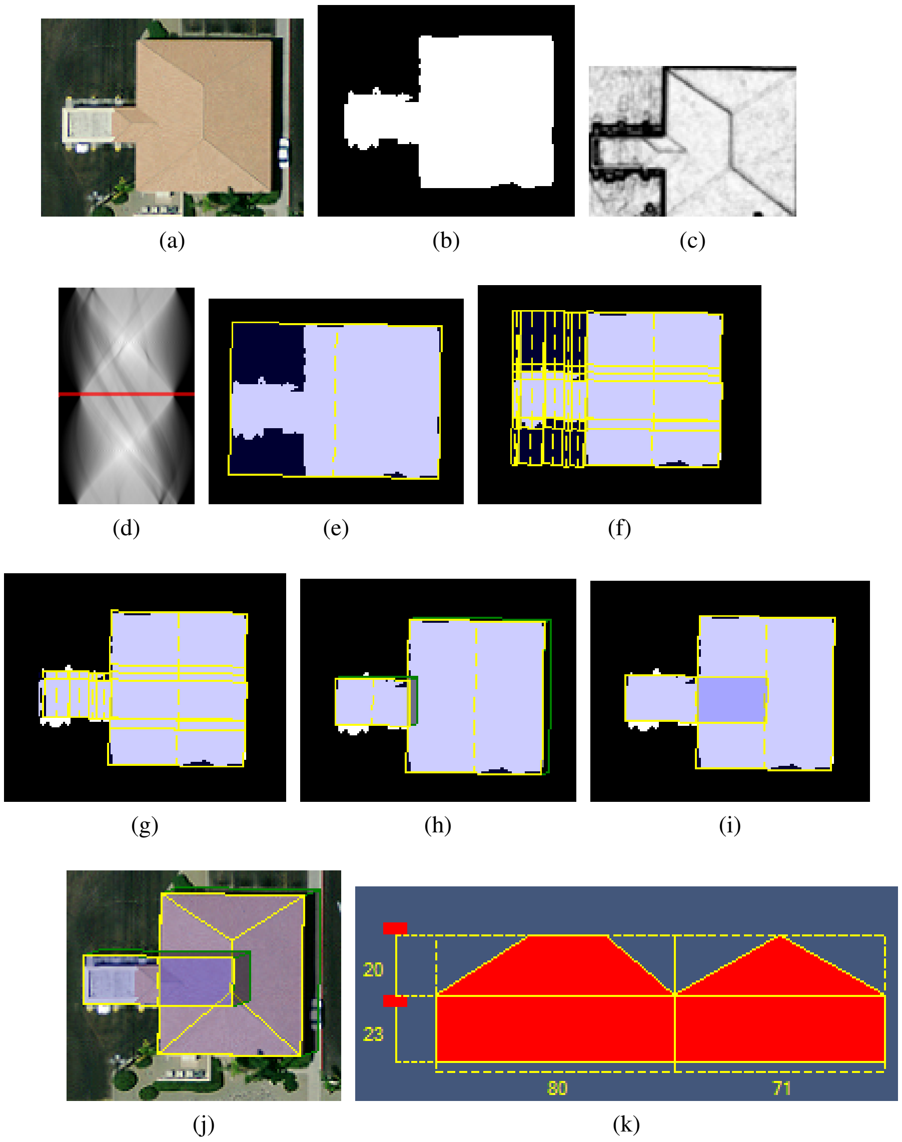


Figure 3.10: Stages of parametric model fitting; (a) aerial image of a hip roof; (b) segmented image from grabCut; (c) sobel edge of aerial image; (d) hough line transform image of (c); (e) edge-aligned bounding box of rooftop segmented area; (f) division of edge-aligned bounding box into horizontal and vertical lines based on projection histogram; (g) cells that remain after filtering those out that don't have enough rooftop pixels; (h) merging small cells into big based on largest rectangle first; (i) extending building blocks into each other preserving the total area; (j) finding roof shape and height using shadow (green outline) in individual building pieces; (k) side views of the bigger building block of the building

3.5 Drawbacks Of Algorithm

In this section we discuss certain drawbacks in our parametric model fitting algorithm.

1. Drawback of merging cells into bigger rectangles based on largest rectangle first - After we

prune out cells from the oriented bounding box of a segmented building region. We merge the remaining cells into non-overlapping larger rectangles such that the rectangle with largest area is preferred first. Then the non-overlapping rectangles are extended into each other such that the total area occupied by the block remains same. This is because we assume that the largest rectangular piece found will actually be a single building piece. However, this is not true in all cases. Figure 3.11a shows such a case. Figure 3.11b and Figure 3.11c show two different ways in which the two building blocks can be arranged. Our algorithm will pick Figure 3.11b because the building block located below together with the shared region has a bigger area compared to the other block together with the shared region. This error can be resolved by first finding out the gable in the shared region and determining which of the two building blocks it belongs to, and then merging the shared region with that building block. Unfortunately, in our current algorithm, we merge the rectangles first and then determine their roof-shape. So we've left this to be resolved in future. The case discussed here is of a simple L-shaped roof. Given an arbitrarily complex rectilinear roof (Figure 3.12), it becomes a big challenge to divide it into correct individual rectangles.

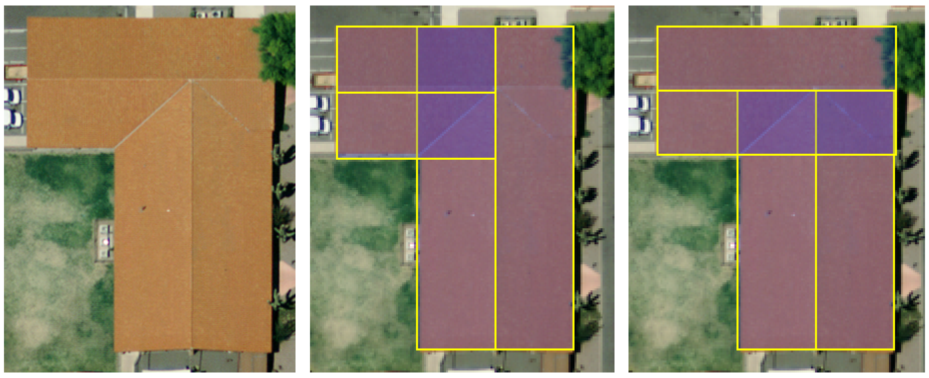

(a) (b) (c)

Figure 3.11: A case where algorithm fails to detect the correct building blocks; (a) reference building; (b) incorrect configuration chosen by our algorithm to represent building based on largest rectangle first; (c) manually chosen correct configuration

2. Drawbacks of assuming that all the building blocks of a building have the same height - We only calculate the height of the shadow-facing building block and assume that to be the height of all the building blocks. However, this is not true in many cases. Figure 3.13 shows such a case in which the bigger building block on the left is taller than its smaller counterpart on the right. However, since the direction of light is such that the smaller building block faces the shadow, the smaller blocks

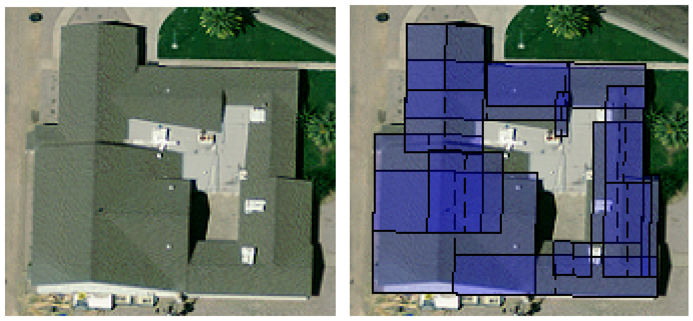


Figure 3.12: Complex rectilinear roof, a difficult case to locate individual rectangular building blocks; (a) roof image; (b) parametric model detected overlayed on top of roof

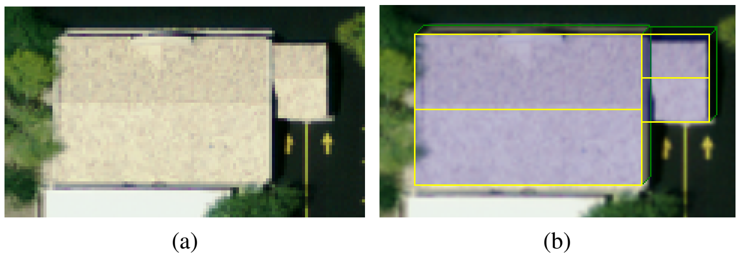


Figure 3.13: Building blocks with different heights; (a) reference building; (b) result of computing height using our algorithm

height is determined and the bigger blocks height is also set to the same by our algorithm (Figure 3.13b).

3. Effect or artifacts on roof in roof shape detection - Artifacts like air-conditioners and chimneys hinder with detection of roof shape. Figure 3.14a shows a roof with an air-conditioner mounted near the center of gable. Figure 3.14b shows the model found, overlayed on the image. Figure 3.14c shows the edge image of the center area of the model. We use hough transform to determine the strongest line in Figure 3.14c. The hough transform does not consider the continuity of lines. It only finds the lines on which most edge pixels lie. As a result, the actual gable is not found because it is too weak compared to the edge of air-conditioner and our algorithm is not able to detect gable in this case. This is a serious problem and in the future we would like to come up with a smarter algorithm that can handle such cases because roofs with such artifacts are very common.

4. Errors due to distortion in rectilinear buildings - In satellite images, rectilinear buildings located further away from the direction in which the camera is facing, look distorted. The adjacent sides

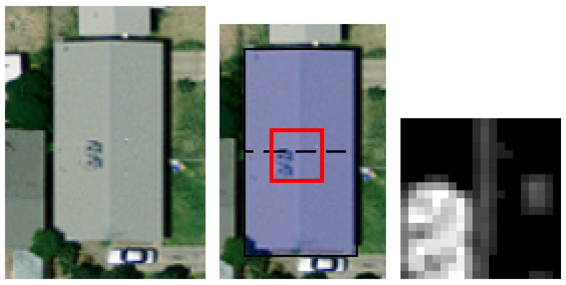

(a) (b) (c)

Figure 3.14: Example showing incorrect roof shape detection because of another object mounted on the roof; (a) reference image of roof; (b) parametric model found for roof (dotted line denotes flat roof was found); (c) edge image near the center of model (notice the gable line is weak and thus undetected by hough transform because of the presence of strong edges of artifact)

do not appear perpendicular to each other even when they actually are because of the obliqueness of the view. Orthorectification of the image only recovers their true locations and sizes, but the distortion still remains (Figure 3.15a). Figure 3.15b shows a model overlaid on the building but only one set of parallel sides match with the building because the adjacent edges of the building are not perpendicular which is enforced by our model.

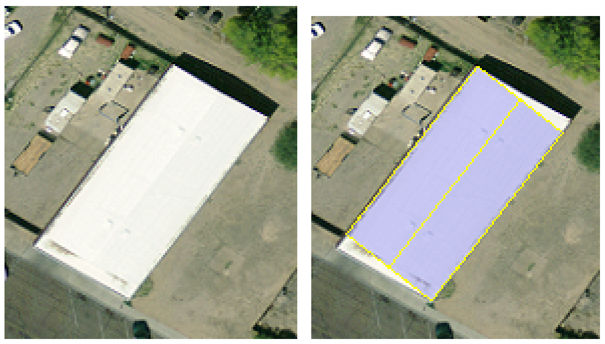

(a) (b)

Figure 3.15: Example showing distortion of a rectilinear building due to obliqueness of view direction; (a) Distorted building; (b) Rectilinear model overlaid on top of building (notice only one set of edges can be aligned since our model enforces perpendicular edges)

3.6 Evaluation

The image dataset used for evaluation here is same as that used for evaluating rooftop segmentation in Chapter 2. The building blocks found after modeling the segmentation results from Chapter 2 are converted into raster images (model image) of same resolution as that of the ground truth. Precision (Equation 2.2) andrRecall (Equation 2.3) is computed between this image and ground truth. Figure

Test image	Precision (%)	Recall (%)
#1	79	75
#2	78	94
#3	79	89
#4	78	86
#5	86	77
#6	84	94
#7	95	96
#8	75	70
Avg.	81.75	85.125

Table 3.1: Evaluation of parametric modeling

3.16 continued to Figure 3.17 show test images (first column), segmentation results using grabcut algorithm mentioned in Chapter 2 (second column), parametric modeling of segmented results (third column) and ground truth (fourth column). Precision and recall is calculated for each fourth and third column pair. Table 3.1 shows the precision and recall for all the test images. We record an average precision of 82% and average recall of 85%. There has been some loss of accuracy over segmented image where we recorded a precision of 87% and recall of 85% as parametric models do not completely overlap segmented blobs, but it is a small price to pay for the advantage of being able to render a high quality, noise-free 3D model of an urban area.

In order to evaluate our algorithm for finding roof shape we tested it on a site of 40 gable roofs. Since we don't have the ground truth about gable angle, we only tested if our algorithm could detect the presence of gable correctly. Out of the 40 gable roofs, 32 were identified as gable whereas the remaining 8 were detected as flat roofs. This shows an accuracy of 80% for finding gable roof shape. Figure 3.18a shows the test site and Figure 3.18b shows the parametric models identified and overlaid on the site.

3.7 Semi-Automatic Building Detection

Using rooftop segmentation and parametric model fitting helps us find most of the buildings accurately. But its possible that some are undetected or not detected accurately. As a part of this framework, we have provided certain tools that can help user improve the results. These are discussed below.

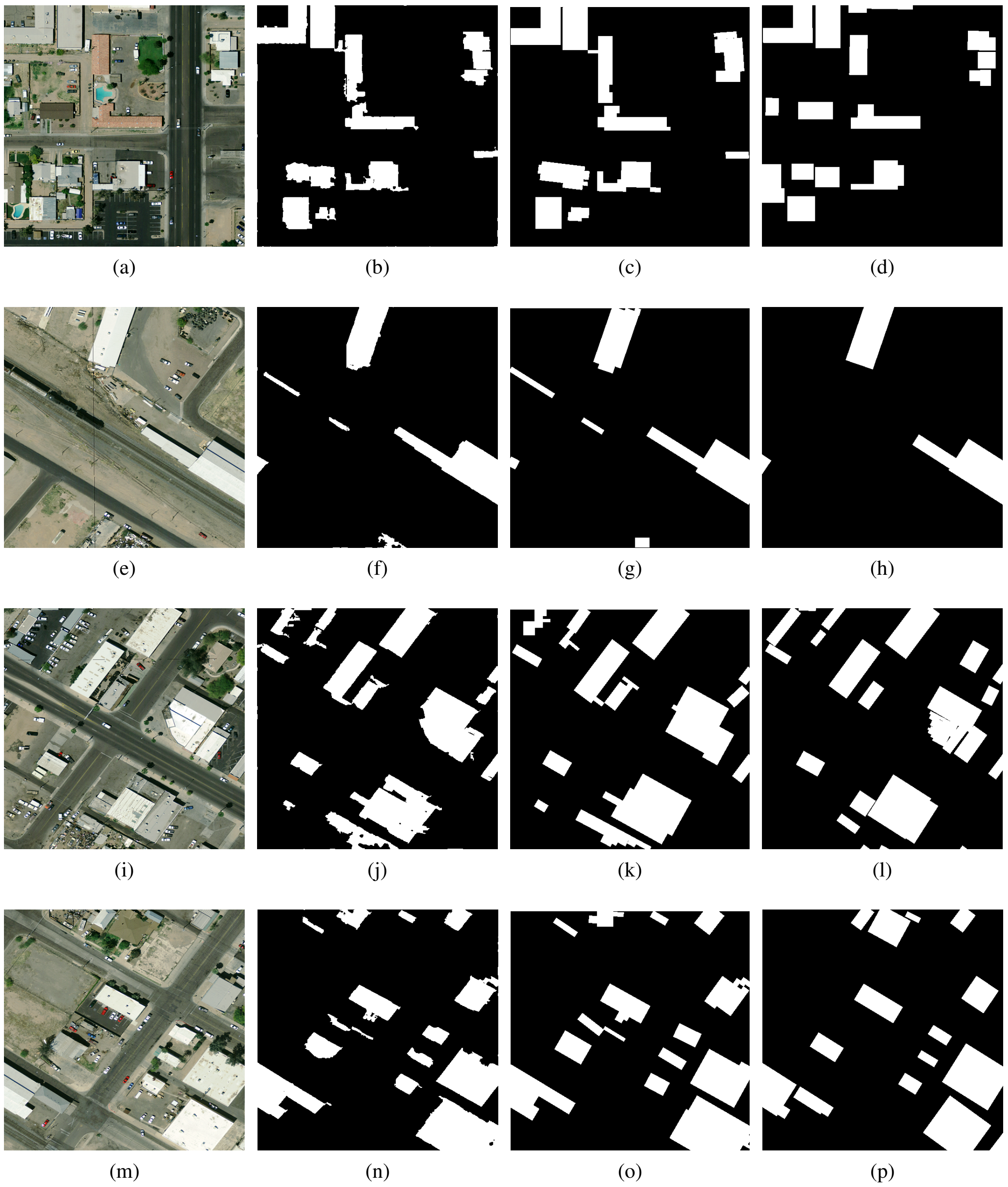
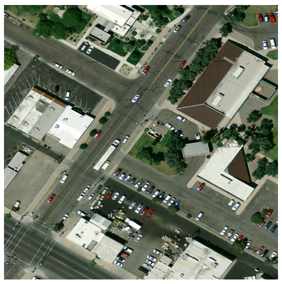
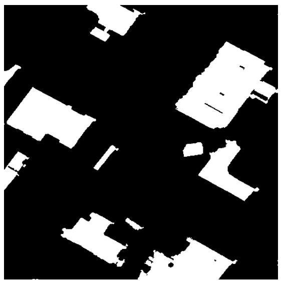
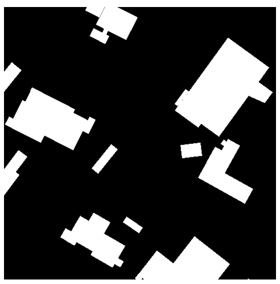


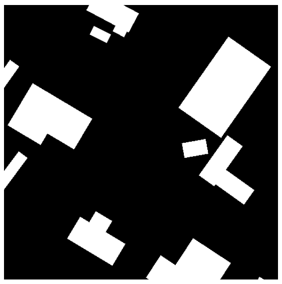
Figure 3.16: (First Column) Test images #1-8; (Second Column) segmentation results for test images #1-8; (Third Column) parametric modeling of second column; (Fourth Column) ground truth for test images #1-8

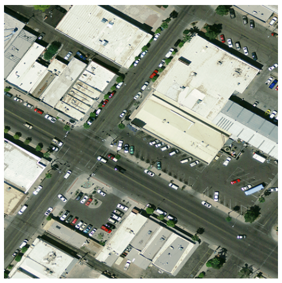
Locating Undetected Buildings - Buildings that remain undetected after automatic rooftop segmentation and parametric model fitting can be manually generated and fitted. But we have employed an algorithm that can work with very little user input. Our algorithm allows a user to specify a point in

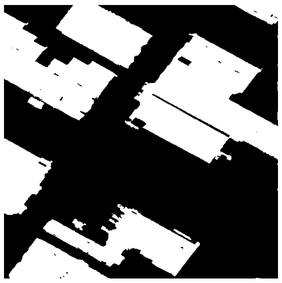

(a)

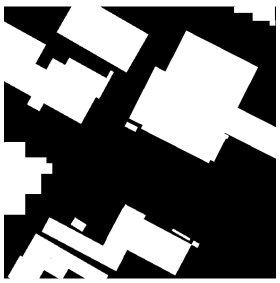

(b)

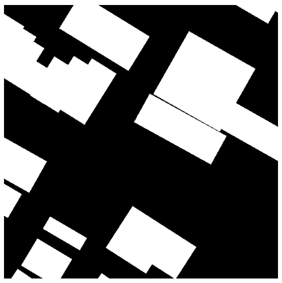

(c)

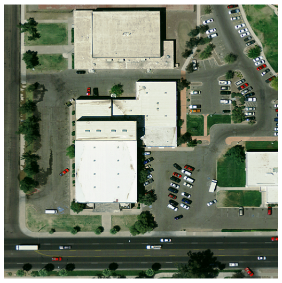

(d)

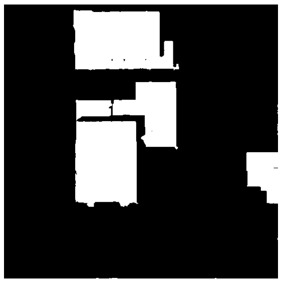

(e)

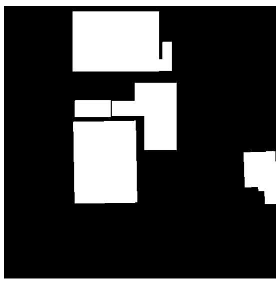

(f)

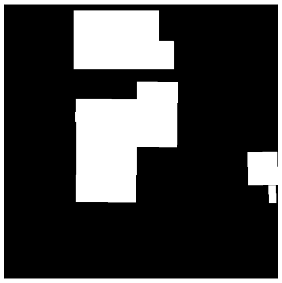

(g)

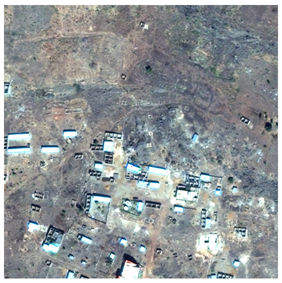

(h)

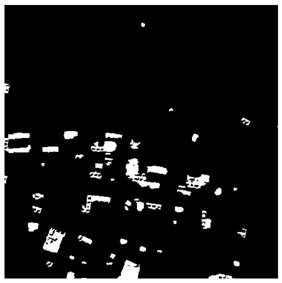

(i)

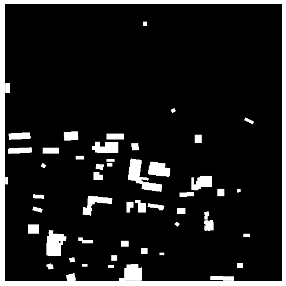

(j)


(k)


(l)


(m)


(n)


(o)

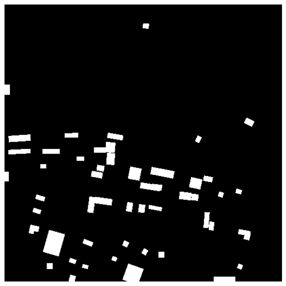

(p)

Figure 3.17: Figure 3.16 continued; (First Column) Test images #1-8; (Second Column) segmentation results for test images #1-8; (Third Column) parametric modeling of second column; (Fourth Column) ground truth for test images #1-8

the image that lie within a building and it detects the building. It can work with complex buildings and can generate parametric models that fit them perfectly along with finding roof-shape and height. Since this algorithm is very fast (1-2 seconds per building), it can give instantaneous results as the

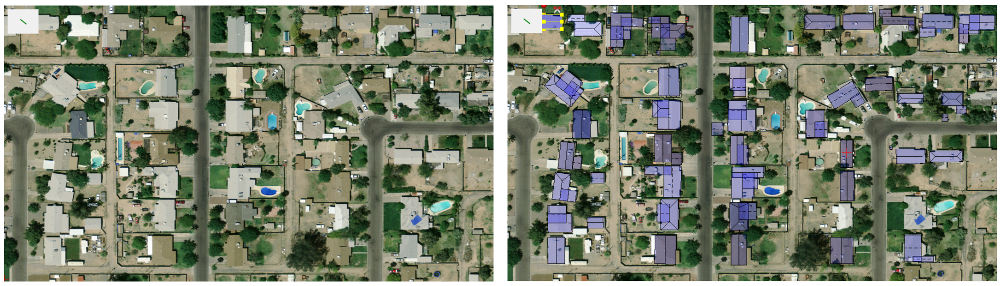

(a) (b)

Figure 3.18: Evaluation of gable roof shape detection by our algorithm; (a) aerial image with 40 gable roofs; (b) parametric models identified, placed over the original image

user moves his mouse over the aerial image thus making it very convenient to use.

Following are the key steps of the algorithm

1. Identify the color of the building in L,U,V format - Let the point specified inside the building be (x0,y0). Figure 3.19a shows a building with a red cross that is marked as a point inside a building. We go through all the pixels within a square of side 10 pixels in the image centered at (x0,y0) and compute their average color. The square is the blue colored patch in Figure 3.19a. Then we again go through all these pixels and identify the color closest to the average color. The metric for closeness is the euclidean distance between the two colors. The color closest to the average color is regarded as the most dominant color of the building, or just color of the building. We do this because, the pixel selected can lie on an air-conditioner or chimney at the top of the roof which might be of a different color. Or its possible that the pixel doesnt represent the true color of building because of poor quality of image. So we average over a small area around the pixel to capture the dominant color of the roof. We then pick the color closest to the average color because if the colors vary a lot, then its possible that the average color turns out to be totally different from the actual roof color. This can happen if the point chosen lies very close to a corner or an edge of a roof. In that case half of the pixels will represent roof color, and other half will represent ground. The average color computed will be somewhere in the middle of roof and ground. So choosing an existing color closest to the average seems like a better choice.

2. Find all the pixels belonging to the building - This step works as a graph search where each pixel acts as a node connected to 8 neighboring node pixels. Given a pixel node that belongs to a building,

we check the 8 neighboring pixels to see if their color is similar to the color of building found in step 1. The similarity is measured by computing similarity in luminance S_L and chroma S_C given by 3.1 and 3.2 respectively. If these two values are less than a threshold typically 0.01 and 0.005 for luminance and chroma respectively, then we assume that the pixels are similar. L, U and V refer to the color in L,U,V format. The pixels found similar are added to the queue and their neighbors are checked for similarity later. Hence, starting with the pixel corresponding to the point specified by the user, we grow this graph and mark all the pixels found similar until no more can be found. The set of pixels found belong to the building. Figure 3.19b shows the pixels highlighted with blue that were found similar when the point specified by user is the center of square in Figure 3.19a. It is possible that this process may cover a very large area specially in a case where a roof's color is similar to a road passing by. So we keep a maximum size of building. We don't process pixels farther than this distance from the point selected thus forcing this step to halt at some point.

$$S_L = (L_{pixel} - L_{building})^2 \tag{3.1}$$

$$S_C = (U_{pixel} - U_{building})^2 + (V_{pixel} - V_{building})^2 \tag{3.2}$$

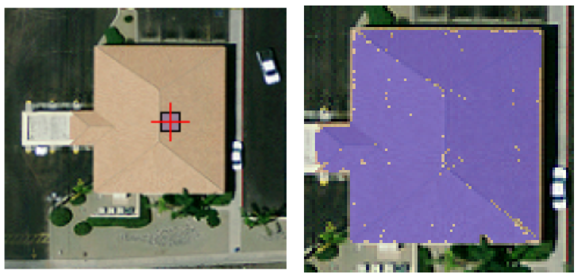


Figure 3.19: Semi-Automatic building detection; (a) compute average color within the patch to find the roof color; (b) pixels found belonging to the building (highlighted in blue)

3. Follow the steps of parametric model fitting - Once we get a set of pixels that belong to building, we find its contour and perform the same steps for parametric model fitting as we perform on the rooftop segmented results from Chapter 2. Thus, all the building blocks pertaining to the building are found along with their roof-shape and height.

3.8 Conclusion and Future Work

We've presented a fully automatic method for modeling rooftops using a segmented image and orthorectified aerial imagery. Our parametric model can successfully define rectiliear buildings with complex roof shapes. Our algorithm can detect rooftops with an overall precison of 83% and recall of 85%.

However there are certain areas where we see a scope for further improvement. These are listed below.

1. Most buildings are rectangular but there can be occasional circular and triangular buildings. So, we can use separate parametric models for triangular and circular buildings. One question is how do we decide which parametric model to use. Simply by fitting a rectangular, circular and triangular model and then picking the one that fits best on the blob is a simple and effective solution.

2. We assume that in complex buildings different building blocks intersect where their gables meet. This is true for most cases but not for all. In many cases, a small building block only intersects with one-fourth of the bigger block next to it. In future, we would like to detect precisely where two building blocks meet. This can be done by performing line segment analysis within the rooftop regions.

3. In many cases, different building pieces of a building don't have equal height. As of now, we detect the height of shadow facing block and assume that to be the height of all the blocks. In future, we would like to detect height of those building blocks separately that cast some shadow.

REFERENCES

[1] Y. Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001.

[2] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[3] R. O. Duda, P. E. Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

[4] P. V. Hough. Method and means for recognizing complex patterns, Dec. 18 1962. US Patent 3,069,654.

[5] R. B. Irvin and D. M. McKeown Jr. Methods for exploiting the relationship between buildings and their shadows in aerial imagery. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6):1564–1575, 1989.

[6] T. Kim, T. Javzandulam, and T.-Y. Lee. Semiautomatic reconstruction of building height and footprints from single satellite images. In *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pages 4737–4740. IEEE, 2007.

[7] T. Kim and J.-P. Muller. Development of a graph-based approach for building detection. *Image and Vision Computing*, 17(1):3–14, 1999.

[8] F. Lafarge, X. Descombes, J. Zerubia, and M. Pierrot-Deseilligny. Automatic building extraction from dems using an object approach and application to the 3d-city modeling. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(3):365–381, 2008.

[9] C. Lin, A. Huertas, and R. Nevatia. Detection of buildings using perceptual grouping and shadows. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 62–69. IEEE, 1994.

[10] C. Lin and R. Nevatia. Building detection and description from a single intensity image. *Computer vision and image understanding*, 72(2):101–121, 1998.

[11] Y.-T. Liow and T. Pavlidis. Use of shadows for extracting buildings in aerial images. *Computer Vision, Graphics, and Image Processing*, 49(2):242–277, 1990.

[12] H.-G. Maas. Closed solutions for the determination of parametric building models from invariant moments of airborne laserscanner data. *transformation*, 2:20, 1999.

[13] B. C. Matei, H. S. Sawhney, S. Samarasekera, J. Kim, and R. Kumar. Building segmentation for densely built urban regions using aerial lidar data. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[14] A. O. Ok, C. Senaras, and B. Yuksel. Automated detection of arbitrarily shaped buildings in complex environments from monocular vhr optical satellite imagery.

[15] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[16] T. Pavlidis and S. L. Horowitz. Segmentation of plane curves. *Computers, IEEE Transactions on*, 100(8):860–870, 1974.

[17] C. Poullis and S. You. Automatic reconstruction of cities from remote sensor data. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2775–2782. IEEE, 2009.

[18] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 309–314. ACM, 2004.

[19] J. A. Shufelt. Exploiting photogrammetric methods for building extraction in aerial images. *International Archives of Photogrammetry and Remote Sensing*, 31:B6, 1996.

[20] B. Sirmacek and C. Unsalan. Building detection from aerial images using invariant color features and shadow information. In *Computer and Information Sciences, 2008. ISCIS'08. 23rd International Symposium on*, pages 1–5. IEEE, 2008.

[21] V. J. Tsai. A comparative study on shadow compensation of color aerial images in invariant color models. *Geoscience and Remote Sensing, IEEE Transactions on*, 44(6):1661–1671, 2006.

[22] V. Verma, R. Kumar, and S. Hsu. 3d building detection and modeling from aerial lidar data. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2213–2220. IEEE, 2006.

[23] C. Vestri and F. Devernay. Using robust methods for automatic extraction of buildings. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–133. IEEE, 2001.

[24] G. Vosselman, S. Dijkman, et al. 3d building model reconstruction from point clouds and ground plans. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/W4):37–44, 2001.

[25] Q.-Y. Zhou and U. Neumann. Fast and extensible building modeling from airborne lidar data. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 7. ACM, 2008.

[26] Q.-Y. Zhou and U. Neumann. 2.5 d dual contouring: A robust approach to creating building models from aerial lidar point clouds. In *Computer Vision–ECCV 2010*, pages 115–128. Springer, 2010.