

**TOWARDS AUTOMATIC 3D RECONSTRUCTION OF
PITCHED ROOFS IN MONOCULAR
SATELLITE/AERIAL IMAGES**

by

Zachary Blair

B.A.Sc., Simon Fraser University, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Sciences

© Zachary Blair 2012

SIMON FRASER UNIVERSITY

Fall 2012

All rights reserved.

However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for “Fair Dealing.” Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

APPROVAL

Name: Zachary Blair
Degree: Master of Applied Science
Title of Thesis: Towards Automatic 3D Reconstruction of Pitched Roofs in Monocular Satellite/Aerial Images

Examining Committee: Dr. Ljiljana Trajkovic
Chair

Dr. Parvaneh Saeedi, Senior Supervisor

Dr. Shahram Payandeh, Supervisor

Dr. Ivan Bajic, Internal Examiner

Date Approved: October 22, 2012

Partial Copyright Licence



The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website (www.lib.sfu.ca) at <http://summit/sfu.ca> and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

Abstract

This thesis presents a novel method for estimating the three-dimensional shape of pitched roofs in monocular satellite/aerial images by utilizing the acquisition geometry (both sun and camera). This method consists of four steps: rooftop extraction, texture and vent removal, identification of planar sections, and 3D roof model generation. During the fourth step, 3D reconstruction candidates are partially rendered and iteratively compared against the original image to evaluate a fitness function. The experimental results illustrate that the proposed framework can derive 3D building rooftop models, including estimates for roof pitch, from single satellite/aerial images in situations where previous techniques fail.

Acknowledgments

I would like to thank Dr. Parvaneh Saeedi, my senior supervisor, for providing me the opportunity to work with her and the other members of SFU's Laboratory for Robotic Vision (LRV). Your encouragement and guidance has been very much appreciated, and without it, I would not have been able to accomplish what I have. I would also like to thank my supervisory committee for their time in reviewing this thesis.

Contents

| | |
|---|-----------|
| Approval | ii |
| Abstract | iii |
| Acknowledgments | iv |
| Contents | v |
| List of Tables | viii |
| List of Figures | ix |
| Preface | xii |
| List of Acronyms and Abbreviations | xiii |
| 1 Introduction | 1 |
| 1.1 Thesis Objective | 1 |
| 1.2 Contribution | 2 |
| 1.3 Thesis Organization | 2 |
| 2 State of the Art in Single-Image 3D Reconstruction | 3 |
| 2.1 The Problem of 3D Reconstruction | 3 |
| 2.1.1 Interactive Approaches | 3 |
| 2.1.2 Automatic Approaches | 5 |
| 3 A Novel 3D Rooftop Reconstruction System | 15 |
| 3.1 Overview | 15 |

| | | |
|----------|--|-----------|
| 3.2 | Extracting Roofs From Nadir Aerial Images | 15 |
| 3.2.1 | Rooftop Extraction Using Corners and Variational Level Set Evolution | 18 |
| 3.3 | Determining the Position of the Sun | 22 |
| 3.4 | Removing Insignificant Rooftop Details | 22 |
| 3.4.1 | Identification and Removal of Vents, Skylights, and Other Fixtures . . | 23 |
| 3.4.2 | Smoothing Texture While Extracting Differences Due to Shading | 24 |
| 3.5 | Identifying Planar Rooftop Surfaces | 25 |
| 3.5.1 | Roof Ridge Identification and Linking | 27 |
| 3.5.2 | Segmentation Into Polygonal Shapes | 30 |
| 3.5.3 | Representing Rooftop Structure | 32 |
| 3.6 | Deducing Depth From Shading | 32 |
| 3.6.1 | Shape Models and the BRDF | 33 |
| 3.6.2 | Model Candidate Reconstructions | 36 |
| 3.6.3 | Evaluating Candidate Reconstructions for Goodness-of-Fit | 37 |
| 3.6.4 | Evolution of a Solution Using Gradient Descent | 37 |
| 3.6.5 | Incorporating Constraints Into the 3D Shapes | 39 |
| 3.6.6 | Why Not Use a System of Linear Equations? | 39 |
| 3.7 | Software Implementation | 39 |
| 3.7.1 | Software Architecture | 39 |
| 3.7.2 | Leveraging OpenCV for Rooftop Segmentation | 40 |
| 3.7.3 | Rendering Roofs in OpenGL | 40 |
| 3.7.4 | Alternatives to OpenGL for Rendering Rooftops | 45 |
| 3.7.5 | Combining OpenCV and OpenGL for Computer Vision | 45 |
| 3.7.6 | Optimizing Tight Loops in C++ Image Processing Programs | 46 |
| 3.7.7 | Utilizing Qt for Cross-Platform Applications | 47 |
| 3.8 | Utilizing GPU Hardware | 49 |
| 4 | Experimental Results | 50 |
| 4.1 | Examples of Rooftop Segmentation | 50 |
| 4.2 | Examples of 3D Reconstructions | 50 |
| 4.3 | Rooftop Pitch Estimation | 52 |
| 4.4 | Run-Time Considerations | 55 |
| 4.5 | Limitations and Assumptions | 56 |

| | | |
|----------|---|-----------|
| 4.6 | Sensitivity Analysis and Portability Issues | 57 |
| 5 | Conclusions | 60 |
| 5.1 | Summary of Contributions | 60 |
| 5.2 | Potential Applications | 60 |
| 5.2.1 | Map Generation and City Planning | 61 |
| 5.2.2 | Architectural Design | 61 |
| 5.2.3 | Real Estate | 63 |
| 5.2.4 | Insurance Assessment | 64 |
| 5.2.5 | Population Estimation | 64 |
| 5.3 | Future Research | 65 |
| | Appendix A Telea Inpainting | 66 |
| | Appendix B Bilateral Filtering | 68 |
| | Appendix C Marker-based Watershed Segmentation | 70 |
| | Bibliography | 72 |

List of Tables

| | | |
|-----|---|----|
| 3.1 | Peak signal-to-noise-ratio (PSNR) of an image blurred with a 5×5 Gaussian filter vs. an image blurred with a bilateral filter with a 5×5 spatial kernel and a 3×3 colour kernel. | 26 |
| 3.2 | Percent of the incorrectly segmented pixels, relative to ground truth segmentations generated by manually segmenting the rooftop images. | 30 |
| 4.1 | Rooftop pitch estimation. | 53 |
| 4.2 | Processor and memory utilization. | 56 |
| 4.3 | Free parameters of the algorithm. | 58 |
| 4.4 | Change trends of the free parameters. | 59 |

List of Figures

| | | |
|-----|--|----|
| 2.1 | Example screen-shot from Google Building Maker, showing the user fitting a rectangular prism outline to an image. | 4 |
| 2.2 | Screen capture of the watershed-assisted segmentation program described in the paper by Cates and Whitaker [24]. | 6 |
| 2.3 | An example of an aerial edge image and a match to a known template (found in 6.5 seconds). | 8 |
| 2.4 | An example of identifying all of the occurrences of a particular model of a home (shown at the top) in an aerial photo of a residential area. | 9 |
| 2.5 | An example of a reflectance map for a Lambertian surface. Each curve identifies a range of surface gradients (p, q) that would result in identical brightness. | 10 |
| 2.6 | Two examples of images and their corresponding estimated 3D shapes using the Tsai-Shah method [39]. | 12 |
| 2.7 | An example of preprocessed and final 3D shapes generated from Bichsel and Pentland's [2] SFS algorithm. | 14 |
| 3.1 | System overview of the pitched rooftop 3D reconstruction algorithm. | 16 |
| 3.2 | Example of the XML output by the rooftop identification program. | 16 |
| 3.3 | Example of the original aerial image, and the extracted rooftop outlines (shown in blue). | 17 |
| 3.4 | Building extraction GUI. | 17 |
| 3.5 | Overview of the roof boundary extraction algorithm. | 18 |
| 3.6 | Position of the sun can be described using polar coordinates. | 22 |
| 3.7 | Illustration of how thresholding and inpainting are used together to remove vents and skylights. | 23 |

| | | |
|------|--|----|
| 3.8 | Two examples of an original rooftop image (left) and the processed image with the vents removed (right). | 24 |
| 3.9 | (a) A hip roof, (b) a square hip roof, and (c) a gabled roof. | 27 |
| 3.10 | Edges and lines detected at several angles using a Prewitt edge detector (top row) and Hough line detector (bottom row). | 28 |
| 3.11 | Edges detected by the Canny edge detector. | 28 |
| 3.12 | Some examples of original, preprocessed, and segmented rooftop images. | 29 |
| 3.13 | The iterative procedure of the Ramer-Douglas-Peucker algorithm as it adds progressively more vertices to approximate a contour. | 31 |
| 3.14 | The effect of different thresholds chosen for the Ramer-Douglas-Peucker algorithm. | 31 |
| 3.15 | (a) The preprocessed image, (b) the image after edge detection, (c) the image after watershed segmentation, and (d) the image after simplifying the regions into simpler polygons. | 32 |
| 3.16 | (a) An example segmented rooftop, and (b) the resulting graph data structure used to represent the segmentation. | 33 |
| 3.17 | Gonioreflectometer setup for measuring the BRDF of a rooftop shingle. Figure from [27]. | 34 |
| 3.18 | Comparison of measured BRDFs for a roughly Lambertian surface (plaster) and rooftop shingles. Figure from [27]. | 35 |
| 3.19 | Reflected light from a rooftop showing L_i , θ_i , L_r , θ_r , and their relationship to the light source, surface, and viewer. | 36 |
| 3.20 | Example of a simple rooftop model (seen from the side), with varying z coordinates for its internal vertices. | 37 |
| 3.21 | Mean-squared error (MSE) relative to the number of iterations. | 38 |
| 3.22 | Example of the rendered 3D model generated for an image. | 39 |
| 3.23 | C++ code snippet illustrating how one can specify the viewing volume. | 41 |
| 3.24 | (a) the original rooftop image, and (b) the resulting image rendered in OpenGL. | 41 |
| 3.25 | C++ code snippet illustrating drawing a simple roof top. | 42 |
| 3.26 | A typical graphics rendering pipeline for a modern graphics processing unit. | 43 |
| 3.27 | GLSL fragment shader for implementing Oren-Nayer shading (adapted from glslang-library [7]). | 44 |
| 3.28 | Comparison of the same building rendered using OpenGL and PBRT. | 45 |

| | | |
|------|---|----|
| 3.29 | C++ code snippet illustrating how one can retrieve pixel data from an image rendered with OpenGL. | 46 |
| 3.30 | Example of a C++ program that applies a threshold to an image inefficiently. | 47 |
| 3.31 | Example of a C++ program that applies a threshold to an image efficiently. . | 48 |
| 4.1 | Examples of original, preprocessed, segmented, and simplified polygon images. | 51 |
| 4.2 | Examples of original images and reconstructed 3D models. | 52 |
| 4.3 | Comparison of 3D reconstruction for Bichsel and Penland’s algorithm [2] and the proposed algorithm. | 52 |
| 4.4 | The Crosstown Food Market from (a) above and (b) street level. These images are from Google Maps [16]. | 54 |
| 4.5 | Measuring roof pitch using the Gimp. | 54 |
| 4.6 | Roofs are designed to have only certain discrete pitches, identified by a whole number of inches in vertical rise over 12 inches of vertical run. | 55 |
| 5.1 | A screen-shot of the Android Google Maps 5.0 application, captured by user Wikipedia user ’Zouzzou’. | 62 |
| 5.2 | A screen-shot of the “Truss 3D” software from FINE civil engineering software [12]. | 63 |
| A.1 | Telea inpainting. | 66 |
| B.1 | Comparison of the preservation of strong edges by the Gaussian filter (top) and Bilateral filter (bottom). | 69 |

Preface

This thesis is submitted in partial fulfilment of the requirements for a degree of Masters of Applied Science. It contains work done during 2011 and 2012. My supervisor on the project has been Dr. Parvaneh Saeedi, assistant professor in the School of Engineering Science at Simon Fraser University (SFU) and faculty member of SFU's Laboratory for Robotic Vision.

Dr. Saeedi introduced me to the problem of estimating the three-dimensional shape of rooftops from aerial or satellite imagery. I found it to be a very interesting and challenging problem. Consequently, I decided to pursue a solution to that problem for my thesis. Working on this project and writing this thesis has been difficult but has taught me a lot about image processing and the shape from shading problem.

List of Acronyms and Abbreviations

| | |
|---|--|
| 2D Two Dimensional | HVS Human Visual System |
| 3D Three Dimensional | LIDAR Light Detection And Ranging |
| AMD Advanced Micro Devices | MATLAB Matrix Laboratory |
| BRDF Bidirectional Reflectance Distribution Function | MSE Mean Squared Error |
| CAD Computer Aided Design | OpenGL Open Graphics Library |
| CT Computerized Tomography | OS Operating System |
| CPU Central Processing Unit | PBRT Physics-based Ray Tracing |
| CUDA Compute Unified Device Architecture | PC Personal Computer |
| DEM Digital Elevation Model | PGM Portable Grey Map |
| FMM Fast Marching Method | PSNR Power Signal to Noise Ratio |
| GLSL Graphics Library Shader Language | QT Qt Toolkit (a recursive acronym) |
| GPU Graphics Processing Unit | RGB Red, Green, Blue |
| GUI Graphical User Interface | SFS Shape From Shading |
| | XML eXtensible Markup Language |

Chapter 1

Introduction

Three-dimensional (3D) rooftop reconstruction from a single monocular electro-optic image is a compelling prospect for a variety of purposes, including architectural design, 3D map generation, urban planning, real estate marketing, insurance assessment, and population density estimation.

Previous approaches to three-dimensional rooftop reconstruction have typically relied on imaging technologies such as stereo or photometric imaging, multi-ray photogrammetry, or LiDAR (Light Detection And Ranging)-based sampled data [44]. This requirement generally makes these approaches more expensive as they require specialized equipment with lower update rates. For instance, LiDAR frequently costs one to four dollars per hectare whereas aerial photography frequently costs pennies per hectare [33]. Additionally, downward-facing LiDAR data used to produce digital elevation models (DEMs) are often more limited in spatial resolution than in aerial photography.

Considering the expense and limitations of other approaches to three-dimensional rooftop reconstruction, there is great interest in alternative methods that might utilize less expensive, higher resolution or more frequently updated imaging technologies.

1.1 Thesis Objective

The objective of this thesis is the development of a framework for estimating the three-dimensional shape of pitched rooftops from a single monocular electro-optic satellite or nadir aerial image. The technical details, applicability, performance, computational requirements, and limitations of this method are presented along with a survey of other related methods.

1.2 Contribution

This thesis proposes a novel framework for three-dimensional rooftop reconstruction from a single photometric colour image, combined with image acquisition geometry, and builds upon previous work for determining building boundaries automatically [23]. This approach requires only a single monocular electro-optic image for its reconstruction. It has been implemented using a cross-platform framework to leverage Graphics Processing Unit (GPU) hardware, thereby reducing the algorithm’s run time [3].

Additionally, the proposed framework utilizes a minimization technique that imposes constraints on the recovered three-dimensional shapes. We are able to impose these constraints because we know that the objects that we are recovering are rooftops that share common characteristics such as straight lines, flat, downwardly-pitched surfaces, and typically are made of asphalt tiled roofing material.

Some novel aspects of the proposed framework include:

- The application of histogram equalization, thresholding, and Telea inpainting [37] to automatically remove rooftop features such as vents, chimneys, and skylights (described in Section 3.4.1) to aide in further processing.
- The utilization of Canny edge detection [5] and morphological dilation for automatically producing seed regions for watershed segmentation [31] of rooftops (described in Section 3.5.1).
- The iterative refinement of a 3D rooftop model by successively rendering hypothesized models and comparing them with the input image (described in Section 3.6).

1.3 Thesis Organization

The remainder of this thesis is divided into five chapters. Chapter 2 presents the state of the art in three-dimensional rooftop reconstruction, with a description of the wider shape-from-shading category of problems. Chapter 3 summarizes the main contributions described in the thesis and explains in detail the proposed algorithm for the three-dimensional reconstruction of pitched roofs from single nadir aerial/satellite images. Chapter 4 outlines the experimental results derived from the proposed framework. Finally, Chapter 5 presents conclusions and opportunities for future research.

Chapter 2

State of the Art in Single-Image 3D Reconstruction

Although it is a long-standing problem in computer vision, there has yet to be a generic method that is capable of reliably reconstructing a three-dimensional shape given only a single monocular electro-optic image. In this chapter, we summarize some of the previous approaches for 3D reconstruction that are particularly relevant.

2.1 The Problem of 3D Reconstruction

2.1.1 Interactive Approaches

Reconstructing the three-dimensional shape of an object from its single monocular electro-optic image is a difficult problem to achieve automatically. Yet, the human visual system (HVS) is usually able to easily estimate three-dimensional shape from such images. For instance, artists are able to produce sculptures of scenes and faces based only on a photograph or sometimes even a painting of a scene or person. To leverage this capability of the human visual system, many approaches to 3D reconstruction are interactive, relying on input from a skilled human operator to shape the reconstructed three-dimensional model in conjunction with an algorithm running on a computer.

Google Earth and Google Building Maker

Google Building Maker is a web application that utilizes the Google Earth browser plugin to enable users to design 3D models of buildings – in any of 127 different cities as of July 2012 – for inclusion in Google Earth [28]. This web application does not provide any means to automatically reconstruct building models; rather, it provides a set of common three-dimensional shapes to be used as building blocks for 3D buildings. The user overlays these three-dimensional building blocks over-top of aerial images of buildings. For instance, Figure 2.1 shows a portion of a building being matched to a rectangular prism shape (shown using red lines) manually.

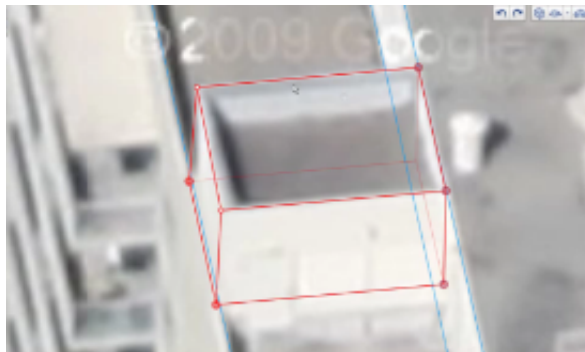


Figure 2.1: Example screen-shot from Google Building Maker, showing the user fitting a rectangular prism outline to an image.

The user repeats this shape-matching process on images of the building from several different perspectives (six are initially provided) and Google Building Maker combines these matched shapes to provide a final textured 3D model. The three-dimensional model is then reviewed for inclusion in Google Earth.

This approach of constructing potentially complex three-dimensional building models out of relatively simple components such as rectangular and triangular prisms works well because many typical building shapes can be closely approximated by intersecting rectangular and triangular prisms. By providing these shapes as building blocks, Google Building Maker leverages this common characteristic of buildings to facilitate easier reconstruction of three-dimensional building models.

User-Assisted Segmentation and 3D Reconstruction in Medicine

Although separate from three-dimensional reconstruction of buildings or rooftops, reconstructing the 3D shape of objects within the body using a series of CT (Computerized Tomography) scanned images requires many of the same computer vision tasks as does reconstructing the three-dimensional shape of buildings or rooftops. For example, in medical tomography, image segmentation and information about the image acquisition geometry is frequently utilized by the three-dimensional reconstruction algorithm. Both of these are also utilized by the framework proposed in this thesis.

Medical technicians may segment medical images to highlight particular organs or other objects of interest. If each image is a slice through a three-dimensional object, multiple segmentations can be combined to create a three-dimensional model of the object of interest. One technique described by Cates and Whitaker [24] uses a hierarchical watershed segmentation algorithm whereby the user is able to select watershed basins for various levels in a segmentation hierarchy. If the image was initially over-segmented using watershed segmentation, the user could then select segments to merge as needed. Similarly, the user could add more catchment basins to achieve a satisfactory segmentation. The paper studied actual users performing segmentation using their user-assisted watershed segmentation program and found that the results compared favourably compared to manual segmentation.

2.1.2 Automatic Approaches

In addition to the interactive approaches, there are also several fully automatic approaches to estimating three-dimensional shape from a single monocular electro-optic image.

Template-Matching Approaches

When a complete list of possible building models is known, an approach to reconstructing a three-dimensional model of a neighbourhood is to exhaustively compare each building in an aerial image against each of the known models to associate a known model with each building in the image. This general approach works very well but variations in lighting resulting from weather, time of day or year, or geographical location may result in dramatically varying image intensities in different images of the same rooftop. Consequently, a simple test for correlation between the pixels in a template and the aerial image is not sufficient unless the lighting direction in the template image is the same as the lighting direction in the aerial

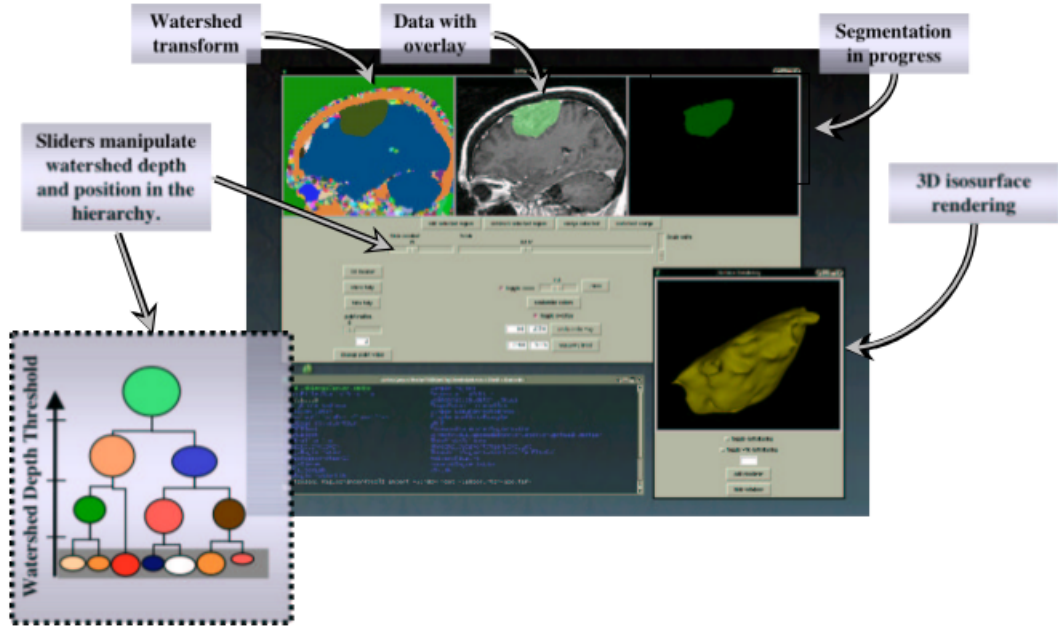


Figure 2.2: Screen capture of the watershed-assisted segmentation program described in the paper by Cates and Whitaker [24].

image.

An alternative to simply testing a template against an aerial image for pixel-by-pixel correlation that does not depend heavily on lighting involves computing the directed Hausdorff distance $h(M, I)$ [1] between the edge image of the template M and the edge image of the aerial image I .

Using this approach, the directed Hausdorff distance $h(M, I)$ between the edge image of the template and the edge image of the aerial image, represented by point sets $M = \{m_1, \dots, m_n\}$ and $I = \{i_1, \dots, i_m\}$, can be calculated as

$$h(M, I) = \max_{m \in M} \min_{i \in I} \|m - i\|. \quad (2.1)$$

The directed Hausdorff distance has several desirable characteristics:

- The distance between two identical shapes is always zero (the identity property).
- Small perturbations, noise, or occlusions such as those resulting from vegetation near buildings only affect the distance by a proportionally small amount.

Some weaknesses of the directed Hausdorff distance are:

- It is susceptible to noise in the form of outlier points because outliers often contribute to the minimum and maximum terms in the Hausdorff distance.
- It may be slow to compute for large images (typically > 5 s for 800×600 natural images) because (at least for the naïve algorithm implementation) it requires that every point in the model be compared against every point in the image.

To overcome these weaknesses, we have experimented with a modified Hausdorff distance known as the partial directed Hausdorff distance:

$$h(M, I) = f^{th}_{m \in M} \min_{i \in I} \|m - i\| \quad (2.2)$$

where $f^{th}(x)$ is the f^{th} percentile of x instead of the maximum. The value of f can be adjusted to make the distance more or less tolerant of outlier noise.

To reduce the time required to process large images, we have taken advantage of the fact that the right-most term in (2.1) ($\min_{i \in I} \|m - i\|$) is simply the distance transformation of I , defined as:

$$I_d(m) = \min_{i \in I} \|m - i\| \quad (2.3)$$

Therefore, for searching applications where the Hausdorff distance would be calculated many times, there are typically time savings of at least one order of magnitude in calculating first the distance transformation of I .

For instance, in Figure 2.3, we have taken an aerial image of Simon Fraser University and extracted strong edges within that image using a Canny edge detector [5]. Then, using the edge image, we found the position that minimized the directed Hausdorff distance between a template of an administrative building and the edge image. This approach also works well in residential areas where there are many nearly identical buildings present in a neighbourhood. For example, in Figure 2.4, we were able to identify all occurrences of a particular model of home in a residential area in about five seconds when running the algorithm on an 800 MHz AMD (Advanced Micro Devices) Athlon(tm)-based machine. If a three-dimensional model for each identified model of home is available in a database, a three-dimensional reconstruction of a neighbourhood could be reconstructed in this way.

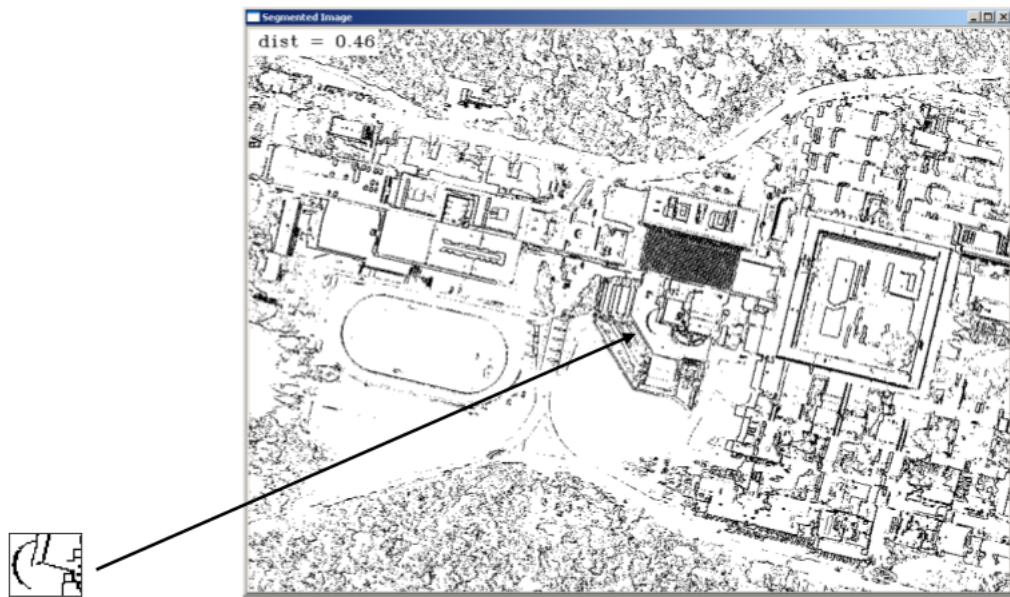


Figure 2.3: An example of an aerial edge image and a match to a known template (found in 6.5 seconds).

Shape from Shading Approaches

Algorithms that recover a three-dimensional shape from a single 2D image based on shading are said to be “shape-from-shading” (SFS) algorithms. Although SFS is one of the classic problems in computer vision, there is yet to be a single algorithm that performs well in all situations [46]. SFS algorithms have been applied to a variety of problems, from estimating the shape of microscopic objects to estimating the shape of geographical features [29].

Many previous SFS approaches assume a surface with Lambertian reflectance, meaning that light reflected from the surface is reflected in all directions equally and the irradiance emitted from the surface is proportional to the cosine of the angle of the incident light. This is a mathematically simple model named after its discoverer Johann Heinrich Lambert for an ideal diffusely reflecting surface. Lambertian reflectance closely approximates the reflectance of very matte materials such as plaster, but is not a good approximation of the reflectance of typical roofing shingles [27].

In an image of a surface exhibiting Lambertian reflectance, the pixel intensity $E(x, y)$



Figure 2.4: An example of identifying all of the occurrences of a particular model of a home (shown at the top) in an aerial photo of a residential area.

at any point is given by the equation:

$$E(x, y) = \frac{\cos(\sigma) + p \cos(\tau) \sin(\sigma) + q \sin(\tau) \sin(\sigma)}{\sqrt{1 + p^2 + q^2}}. \quad (2.4)$$

where τ is the tilt of the illuminant, σ is the slant of the illuminant, and p and q define the surface orientation in terms of the partial derivatives $p = \frac{\partial Z}{\partial x}$ and $q = \frac{\partial Z}{\partial y}$. An important consequence of Lambertian surfaces reflecting light equally in all directions is that the perceived brightness of such a surface does not depend on the viewing angle. By knowing the orientation of the surface and light source, we can compute a reflectance map, which projects surface gradients onto expected image pixel intensities. For instance, Figure 2.5

shows a reflectance map for a Lambertian surface.

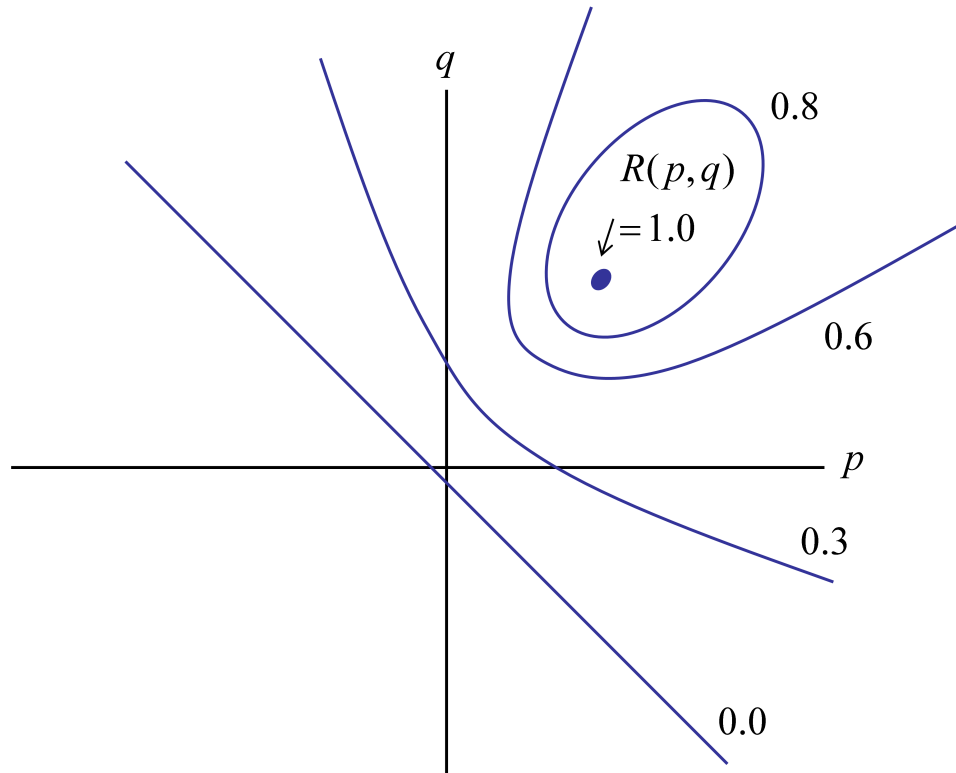


Figure 2.5: An example of a reflectance map for a Lambertian surface. Each curve identifies a range of surface gradients (p, q) that would result in identical brightness.

Given the brightness of an individual point on a surface, there are generally an infinite number of possible surface orientations that would result in that brightness. Each curve shown on Figure 2.5 corresponds to a particular reflectance — or brightness — and each point along any given curve identifies a surface gradient (p, q) that would result in that reflectance. Most SFS techniques utilize such a reflectance map although a reflectance map only enables an algorithm to directly calculate a range of possible surface orientations for each point on the surface. Consequently, various SFS approaches differ in large part by how they choose to combine local orientation information to reconstruct a three-dimensional shape [46]. In general, SFS approaches may be grouped into four categories: propagation approaches, local approaches, linear approaches, and minimization-based approaches [46].

Propagation approaches Propagation approaches, such as those described by Horn [21], were some of the first to be developed. They rely on propagating shape information outward from a set of initially chosen singular points at which the depth is known. In the algorithm proposed by Horn, the term “characteristic strips” is used to describe the paths emanating from the singular points along which the depth is estimated [21].

Local approaches Local approaches compute the depth at each point based on the point’s intensity value and the first and the second derivatives of the intensity with respect to space. They operate under the assumption that the surface at each point is approximately spherical and the radius of curvature of the surface at each point varies. One of the most influential papers on a local SFS approach was published by Pentland [34].

Linear approaches Linear approaches rely on a linear approximation of the reflectance function. An example is the Tsai-Shah method for extracting shape from shading using linear approximation [39]. Some results from that algorithm are shown in Figure 2.6.

The original technical report by Tsai and Shah provided a link to a program written in the C programming language that implemented the algorithm. The program accepted a portable grey map (PGM) image and a description of the light source direction as an input, and output a depth map. For a 128×128 image, the program completed 5 iterations in a fraction of a second while running on an 800 MHz machine. The gnuplot utility was used to generate three-dimensional surface plots of the resulting depth maps. The results were good for the sample images but were not as good for other images. The resulting depth maps seem to closely resemble the original images themselves.

Minimization Approaches Minimization approaches, explored by many different researchers, recover a three-dimensional shape by minimizing an energy function over the entire image [46]. For each pixel value, only the intensity is initially known. From that single piece of data, the shape-from-shading algorithm attempts to determine both the horizontal and vertical gradient of the surface at that point. Clearly, solving for two unknowns given only a single known quantity (the intensity) is an under-constrained problem, and so we must employ additional constraints to find a unique solution.

Therefore, the energy function applied to the entire image necessarily must place a constraint on the shape-from-shading problem. Some common types of constraints include:

- brightness constraints

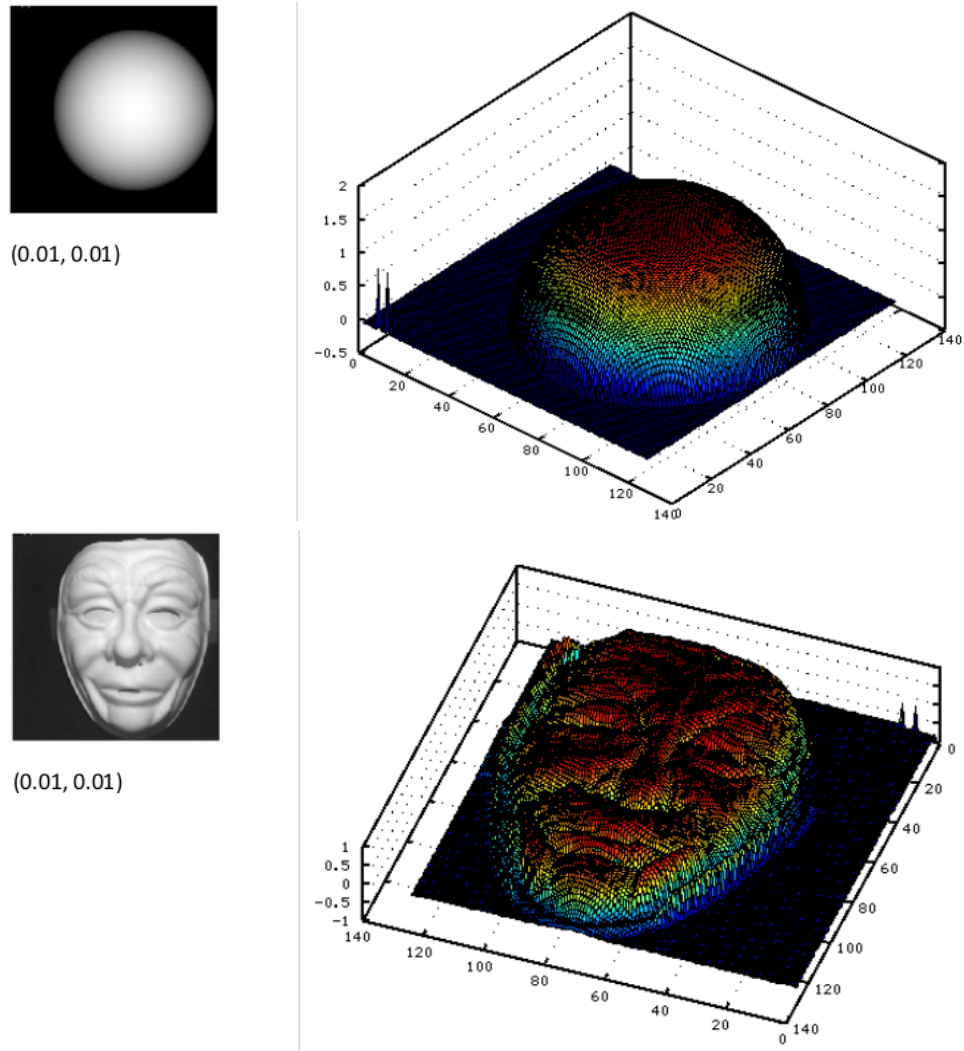


Figure 2.6: Two examples of images and their corresponding estimated 3D shapes using the Tsai-Shah method [39].

- smoothness constraints
- intensity gradient constraints

The brightness constraint attempts to minimize the total brightness error of the reconstructed image compared to the input image. This is what the algorithm in this thesis proposes, so it could be classified as a minimization approach that utilizes a brightness constraint. The function that is minimized for methods that utilize a brightness constraint is given by,

$$\iint (I(x, y) - R(x, y))^2 dx dy, \quad (2.5)$$

where I is the original image, and R is the reconstructed image.

The smoothness constraint attempts to minimize the rate at which the surface gradient changes over the x and y directions by utilizing an energy function that penalizes rapid changes in surface orientation. This favours surfaces that are smooth over those that have sharp discontinuities. Consequently, methods that impose a smoothness constraint use an energy function of the form,

$$\iint (p_x^2 + p_y^2 + q_x^2 + q_y^2) dx dy, \quad (2.6)$$

where p_x , p_y , q_x , and q_y are the partial derivatives of the components of the surface gradient vector (p, q) with respect to x and y .

The intensity gradient constraint attempts to minimize the difference between the intensity gradient of the original image and that of the reconstructed image. The energy function used to impose an intensity gradient constraint takes the form,

$$\iint (R_x(x, y) - I_x(x, y))^2 + (R_y(x, y) - I_y(x, y))^2 dx dy. \quad (2.7)$$

where R_x and R_y are the reconstructed image's intensity gradients along the x and y axis, and I_x and I_y are the original image's intensity gradients along the x and y axis. The gradient images R_x , R_y , I_x , and I_y could be computed by convolving R and I with a Sobel filter.

Limitations and Challenges

Unfortunately, the approaches listed above are generally intolerant to noise or rely on assumptions that are hard to be realized for satellite/aerial images of cities. For instance,

in propagation-based shape-from-shading methods, noise can accumulate as it propagates from one part of the image to the other, and most of these approaches are unable to handle non-Lambertian reflectance, inter-reflections, and other complex lighting. Furthermore, reconstructing a shape from shading information, without constraints or further information is an under-constrained problem, so for any given image, multiple recovered shapes are possible [46].

For instance, when applying Bichsel and Pentland’s algorithm [2] to a very simple rooftop image, the best results are often not realistic-looking shapes.

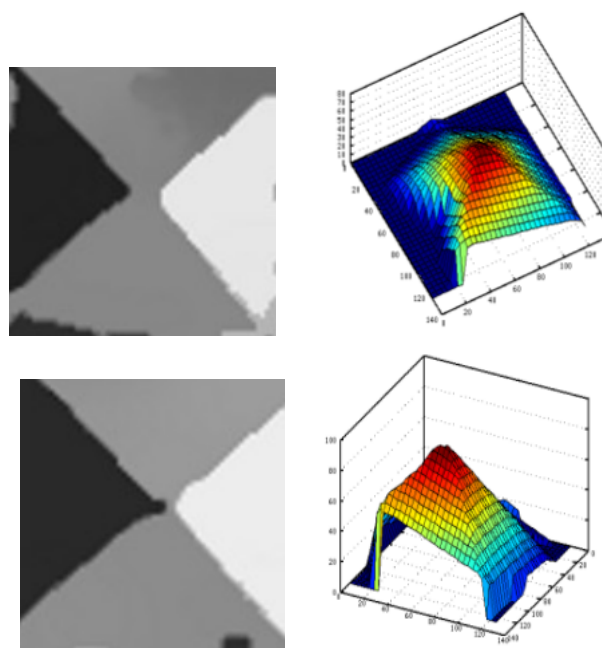


Figure 2.7: An example of preprocessed and final 3D shapes generated from Bichsel and Pentland’s [2] SFS algorithm.

Consequently, some practical approaches rely on user input during segmentation to achieve satisfactory results [11]. Another way to overcome these limitations is by utilizing constraints based on the type of objects being reconstructed [36].

The objective of this thesis is the development of a method for estimating the three-dimensional shape of pitched rooftops from a single monocular electro-optic satellite or aerial image that overcomes some of these limitations of other SFS methods. The applicability, performance, computational requirements, and limitations of this method are presented.

Chapter 3

A Novel 3D Rooftop Reconstruction System

3.1 Overview

The proposed algorithm in this thesis consists of four general steps: extracting each rooftop from a larger image; preprocessing to remove details like vents; segmenting into polygonal planar surfaces; and generating a three-dimensional model from the planar surfaces. Figure 3.1 provides a general overview of these steps, and the following sections describe each step in detail.

3.2 Extracting Roofs From Nadir Aerial Images

The first step, extracting roofs from nadir aerial images, is performed using a hierarchical feature-based image segmentation algorithm [9]. The input of this step is a colour satellite or aerial image of an area containing buildings with pitched roofs, and the output of this step is a set of regions within the image that have been identified as rooftop boundaries. A MATLAB (matrix laboratory) implementation of this algorithm was used. These regions are then cropped automatically out of the image, and for each individual rooftop image, we perform the remaining steps of the algorithm.

The MATLAB program outputs the coordinates of the outlines of each building in the aerial image, in an XML file as shown in Figure 3.2. Using those XML files, we can extract each rooftop from the image, as illustrated in Figure 3.3.

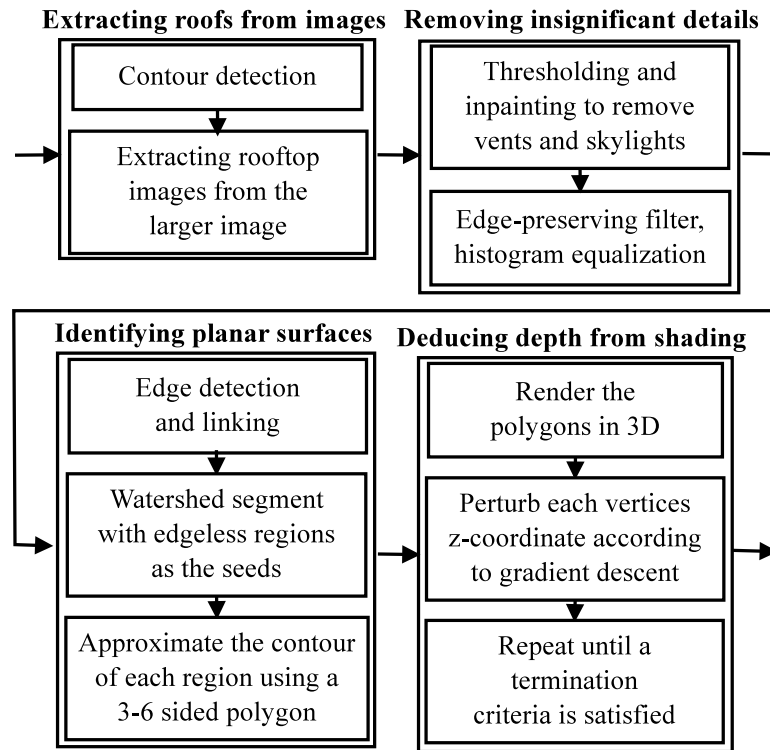


Figure 3.1: System overview of the pitched rooftop 3D reconstruction algorithm.

```

<?xml version="1.0" encoding="utf-8"?>
<Rooftop_Images>
<roof>
<vertice>162.000000,26.000000</vertice>
<vertice>156.000000,28.000000</vertice>
<vertice>356.000000,87.000000</vertice>
<area>161.815542</area>
</roof>
...
<roof>
...
</roof>
</Rooftop_Images>
  
```

Figure 3.2: Example of the XML output by the rooftop identification program.



Figure 3.3: Example of the original aerial image, and the extracted rooftop outlines (shown in blue).

The roof detection implementation includes a GUI (Graphical User Interface) with which a user may interact with the system, shown in Figure 3.4.

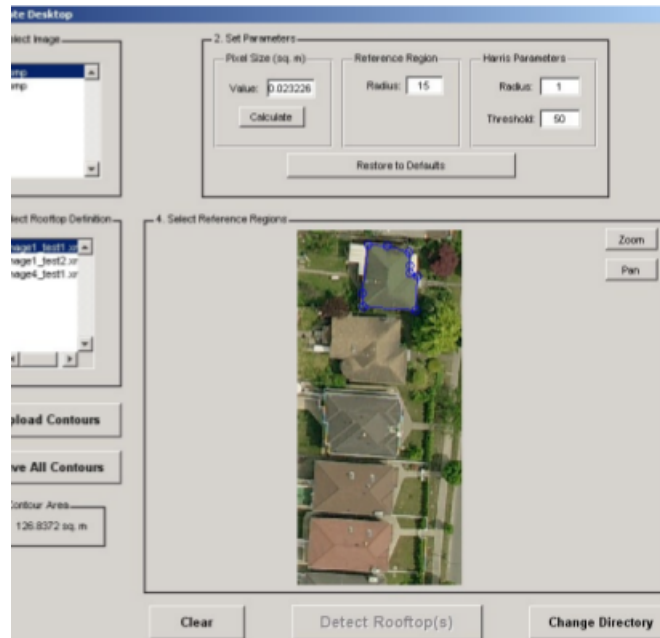


Figure 3.4: Building extraction GUI.

3.2.1 Rooftop Extraction Using Corners and Variational Level Set Evolution

The algorithm used for automatically extracting roof boundaries is described in a paper by Cote and Saeedi [8].

There are three stages of this algorithm:

1. Reference points identification.
2. Rooftop detection.
3. Rooftop candidates assessment.

These three stages are somewhat of an oversimplification of the algorithm, which is illustrated with greater detail in Figure 3.5, so the following paragraphs will explain each stage in depth.

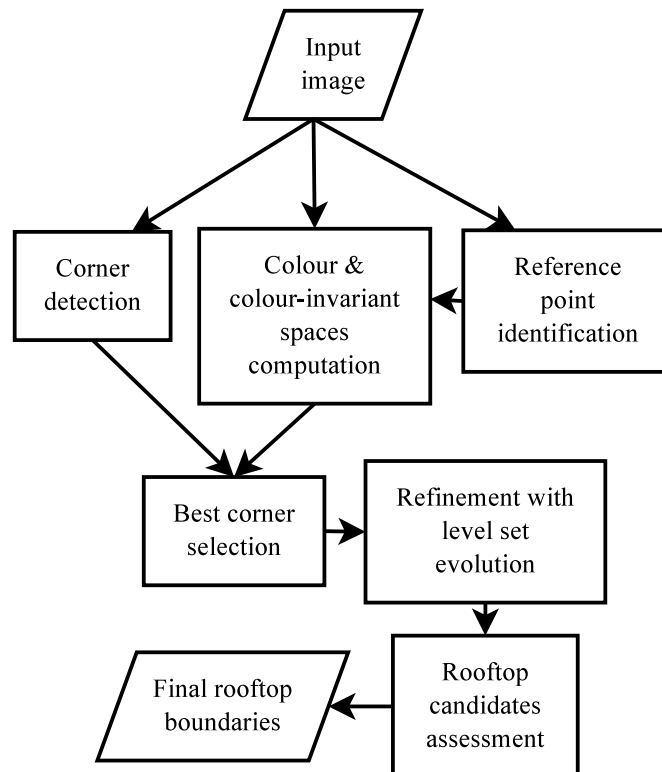


Figure 3.5: Overview of the roof boundary extraction algorithm.

Reference Point Identification

The first stage, reference point identification, involves locating an initial point inside the boundary of each rooftop (referred to as reference point). These reference points are identified using a k-means clustering algorithm to iteratively converge on a set of k clusters within the image, where the pixels within each cluster have similar hue and saturation values. Only the hue and saturation components of the image are considered, so that variations in illumination across rooftops do not interfere with their detection. This is important, because certain classes of rooftops — such as the pitched rooftops described in this thesis — exhibit large variations in illumination on different faces of the rooftop.

The value of k and the initial centroid locations $\{kc_i\}$ are extracted from the 2D histogram computed for the hue-saturation image. Each rooftop is assumed to be made predominantly of one uniform material, so each rooftop will correspond to a peak in the hue-saturation histogram. Consequently, the initial value of k and kc_i is given by

$$k = |\{h, s : H(h, s) = \text{regional maximum}\}| \quad (3.1)$$

$$\{kc_i\} = \{h, s : H(h, s) = \text{regional maximum}\} \quad (3.2)$$

Next, each of the k clusters are further refined, with some clusters being split in two and others being discarded. A morphological opening operation with a square 5×5 structuring element eliminates small, disconnected, or narrow clusters and smooths the remaining cluster boundaries. Clusters that are divided by strong edges are split into smaller clusters. Finally, clusters are evaluated against several criteria, and those that do not satisfy the criteria are discarded:

- Clusters that are connected to the image border are discarded because they correspond to roofs that are not fully within the image.
- Clusters with highly eccentric shapes are also discarded because they are unlikely to correspond to rooftops.
- Clusters smaller than 400 square pixels, or 9 m^2 are discarded because they are too small to correspond to most buildings.
- Clusters with a center of gravity outside the cluster are discarded.
- Clusters with hue and saturation values common to vegetation are discarded.

The centroids of each of the remaining clusters are used as reference points for subsequent steps of the algorithm. A reference radius is also computed for each reference point that is proportional to the size of the cluster but is restricted to a range within 1 m and 2 m.

Rooftop Detection

Corners have been found to be one of the most robustly-detectable features of a rooftop, so they are leveraged to produce a rough estimate of the building boundary. Consequently, the algorithm relies on the assumption that the building has distinctive corners (as identified using a Harris corner detector [18]), and the roofing material is of a roughly uniform colour and reflectance. These limitations are not too restrictive because most buildings have corners and a uniform roof material but they do make this algorithm inappropriate for identifying unusual buildings such as grain silos or water towers, which are typically cylindrical in shape.

Prior to corner detection, the image is transformed using multiple colour and colour-invariant spaces to attenuate differences in pixel values due to illumination and amplify differences due to colour variations. Specifically, the algorithm uses a Gaussian colour invariance image produced using an algorithm proposed by Geusebroek et al [14], the hue-saturation image, and a mean-squared error image. The Gaussian colour invariance image is defined by

$$\Gamma = (E_\lambda/E, E_{\lambda\lambda}/E) \quad (3.3)$$

where

$$\begin{bmatrix} E \\ E_\lambda \\ E_{\lambda\lambda} \end{bmatrix} = \begin{pmatrix} 0.3 & 0.58 & 0.11 \\ 0.25 & 0.25 & -0.05 \\ 0.5 & -0.5 & 0 \end{pmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \quad (3.4)$$

The hue-saturation image is defined more simply as the image generated by converting the HSV image to an RGB image with the following mapping:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} H \\ S \\ 0 \end{bmatrix} \quad (3.5)$$

Using these spaces, the algorithm identifies and localizes the corners of the building using a Harris corner detector [18]. By using these alternative spaces for edge detection, the algorithm is more robustly able to identify the corners of roofs that exhibit varying levels of illumination, such as pitched roofs.

In general, more corners will be detected than correspond to the corners of buildings, so the detected corners are evaluated using two criteria:

- Similarity of the neighborhood around each corner (WC_i) with that of the reference point on the Gaussian colour invariance image, Γ .
- Smoothness of the MSE (mean-squared error) profile along the segment SC_i between C_i and the center of the reference point.

The corners that satisfy the criteria are then connected in the order of their polar angles with respect to the center of the reference point to create a polygonal representation of the roof boundary P . This polygonal representation is an approximation of the true roof boundary, so subsequent steps refine that boundary using level set evolution.

To provide the initial contour for the level set evolution algorithm, the polygonal representation is shrunk by 1 m on all sides. Shrinking the contour is necessary because the parameters of the energy functional are set to drive the contour outward, so the initial contour must lie within the true roof boundary. The level set evolution algorithm then iteratively perturbs this contour to minimize an energy functional that drives the contour to the image features in the mean-squared-error image.

Rooftop Candidates Assessment

Once the level set evolution algorithm has converged to a solution, the resulting rooftop boundary candidate is assessed, such that regions that exhibit some rooftop-like properties but which are not actual rooftops can be discarded. Each rooftop boundary candidate is assessed against several criteria:

1. Boundaries within which there is a lot of intensity variation in the Gaussian color invariance space are discarded, because true rooftops are relatively uniform in intensity within the Gaussian color invariance space.
2. Boundaries that have low solidity, defined as the ratio of their area to the area of their convex hull are also discarded.
3. Boundaries that do not comply closely with the edges in the edge image are also discarded.

4. When two or more detected boundaries are identical, only one is kept and the others are discarded.
5. Boundaries that are connected to other boundaries containing similar colour are joined and their centroid is used for a second pass of the algorithm, because such boundaries likely correspond to different parts of the same rooftop.

3.3 Determining the Position of the Sun

Aerial and satellite imagery is often accompanied by meta-data describing the time and date of the image acquisition, as well as the orientation of the camera when the image was taken. Using this information, it is possible to calculate the angle of the sun's rays relative to the Earth, and therefore also relative to the buildings photographed. Figure 3.6 illustrates how the position of the sun can be described using polar coordinates.

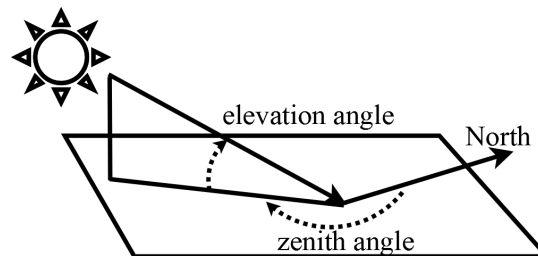


Figure 3.6: Position of the sun can be described using polar coordinates.

3.4 Removing Insignificant Rooftop Details

Some relatively small rooftop details, such as vents, chimneys, shingles, and skylights, are not necessary for estimating the the larger-scale three-dimensional shape of the rooftop. Rooftop ridges and boundaries are sufficient for estimating the three-dimensional shape of the rooftop, and other details simply interfere with the robust detection of rooftop ridges and boundaries.

For instance, skylights and rooftop vents frequently appear white or black from aerial or satellite imagery. Therefore, when we use a Canny edge detector [5] to identify the strong edges in the image, the edges from skylights and rooftop vents appear predominantly in the edge map. In some cases, these vents or skylights are far enough from other sources

of edges, like rooftop ridges, that they could be identified and removed on the basis that they are not connected to other edges in the edge map. However, vents and skylights are frequently near rooftop ridges, or positioned in a line on the rooftop, such that subsequent edge linking steps are unable to differentiate between edges corresponding to a line of vents or skylights and edges corresponding to rooftop ridges.

3.4.1 Identification and Removal of Vents, Skylights, and Other Fixtures

Removing outlier regions (corresponding to vents, skylights, etc.) involves performing a threshold-based operation to find very bright or very dark small regions within the rooftop area. These regions typically correspond to rooftop vents or skylights, which if not removed, may negatively affect subsequent segmentation steps.

To identify the outlier regions, we convert the colour image given as the input to a grayscale image, with each pixel's grayscale intensity ranging from 0 (black) to 255 (white). We then perform histogram equalization on the image to stretch its contrast. In a histogram-equalized image, we found that pixels with intensities greater than 180 on a 0 to 255 scale typically corresponded with white plastic vents, and pixels with intensities less than 90 typically corresponded to black plastic vents or skylights.

Once these outlier regions have been identified, extremely small regions (those with areas of less than 0.5m^2) are removed by a morphological opening operation. The remaining regions are then used as a mask to identify the regions that must be inpainted (a technique typically applied to photo restoration that replaces masked-off regions of an image with content generated from the surrounding pixels) [37]. After inpainting, the vents and skylights on the roof appear to have disappeared, because the surrounding pixels have been used to fill in the area that they previously occupied. Figure 3.7 illustrates this inpainting algorithm.

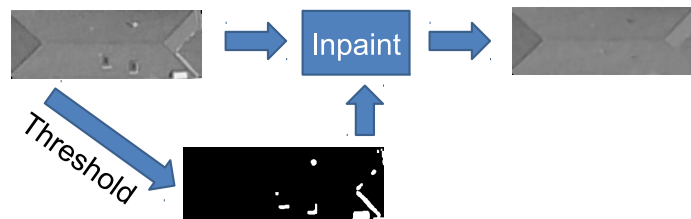


Figure 3.7: Illustration of how thresholding and inpainting are used together to remove vents and skylights.

There are a variety of inpainting algorithms available but we chose to use a relatively

simple but effective algorithm that was developed by Alexandru Telea [37]. Appendix A describes the Telea inpainting algorithm in more detail.

Figure 3.8 illustrates the output of this vent removal process.

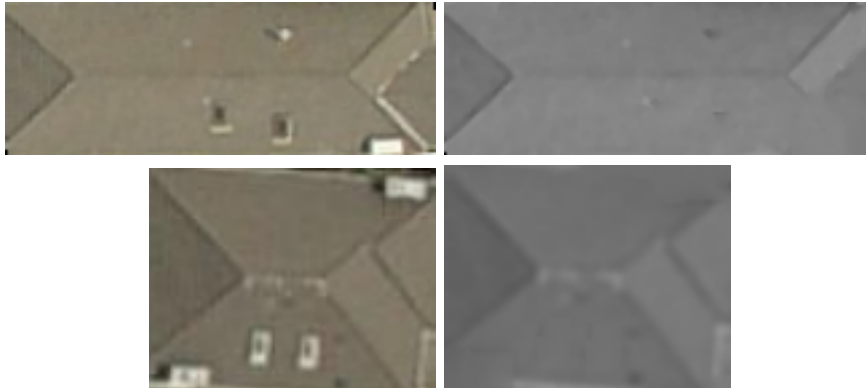


Figure 3.8: Two examples of an original rooftop image (left) and the processed image with the vents removed (right).

3.4.2 Smoothing Texture While Extracting Differences Due to Shading

Next, we apply an edge preserving filter to reduce the rough texture of the rooftop while maintaining the important edges in the image. The classic median filter produces good results but both mean-shift filtering [6] and bilateral filtering qualitatively produced images that better captured the relevant edges while smoothing irrelevant details. In the implementation of this algorithm, we experimented with both mean-shift filtering and bilateral filter, but chose the bilateral filtering because of its slightly better runtime performance. This difference in runtime performance is the result the bilateral filter only requiring one pass over the image, while mean-shift filtering requires several iterations to converge on a solution. Appendix B describes the bilateral filtering algorithm in finer detail.

We may quantitatively evaluate the effect of various filters by computing the mean-squared-error (MSE) or peak signal-to-noise ratio (PSNR) of the smoothed image relative to the original image. A filter that qualitatively smooths insignificant details while either minimizing the MSE or maximizing the PSNR is ideal, because such a filter enables the smoothed image to retain important information. Equation 3.6 shows how the MSE may

be computed for two images, I and K , both of which are $m \times n$ in size.

$$MSE = \frac{1}{m n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (3.6)$$

The peak signal-to-noise ratio (PSNR) of a smoothed image may be computed by comparing the smoothed image to the original image using the formula

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \quad (3.7)$$

$$= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \quad (3.8)$$

$$= 20 \cdot \log_{10} (MAX_I) - 10 \cdot \log_{10} (MSE) \quad (3.9)$$

Table 3.1 compares the PSNR of blurred rooftop images using a Gaussian filter and compares them with the PSNR of images blurred using a bilateral filter. The images blurred with a bilateral filter have a higher PSNR than those blurred with the Gaussian filter, which is a consequence of the bilateral filter keeping more image detail around edges than the Gaussian filter, which blurs all parts of the image equally.

Finally, we apply a histogram equalization to the image to increase the contrast within the image. This step is important for the following steps, which rely on the contrast between rooftop regions to identify edges.

Histogram equalization Histogram equalization proceeds by first calculating the histogram for the grey-scale image. Next, it computes the cumulative distribution function (cdf) for the histogram, such that $cdf(v)$ denotes the number of pixels in the image whose intensity is less than or equal to v . Using $cdf(v)$, we are then able to map each pixel value v in the image to a new pixel value, $h(v)$, using the formula

$$h(v) = \text{round} \left(\frac{cdf(v) - cdf_{min}}{cdf_{max} - cdf_{min}} \times 255 \right) \quad (3.10)$$

where cdf_{min} is the minimum value of the cumulative distribution function, and cdf_{max} denotes the number of pixels in the image.

3.5 Identifying Planar Rooftop Surfaces

Pitched roofs (as opposed to flat roofs) consist of a set of planar intersecting surfaces that slope downward. Where these planar surfaces intersect, there is a rooftop ridge. The

| Image # | Bilateral filter PSNR | Gaussian filter PSNR |
|---------|-----------------------|----------------------|
| 1 | 21.34 | 17.76 |
| 2 | 18.75 | 16.44 |
| 3 | 21.58 | 19.14 |
| 4 | 21.71 | 19.17 |
| 5 | 21.40 | 18.86 |
| 6 | 20.57 | 17.49 |
| 7 | 19.56 | 16.52 |
| 8 | 21.30 | 17.87 |
| 9 | 19.23 | 15.83 |
| 10 | 22.47 | 18.96 |
| 11 | 23.68 | 19.96 |
| 12 | 24.13 | 20.81 |
| 13 | 15.13 | 13.58 |
| 14 | 23.06 | 19.94 |
| 15 | 21.05 | 17.45 |
| 16 | 21.95 | 18.60 |
| 17 | 21.29 | 17.54 |
| 18 | 19.15 | 15.68 |
| 19 | 21.33 | 18.39 |
| 20 | 20.63 | 17.58 |
| 21 | 20.88 | 18.35 |
| 22 | 21.10 | 18.59 |
| 23 | 20.00 | 17.20 |
| 24 | 21.15 | 19.06 |
| Average | 20.94 | 17.95 |

Table 3.1: Peak signal-to-noise-ratio (PSNR) of an image blurred with a 5×5 Gaussian filter vs. an image blurred with a bilateral filter with a 5×5 spatial kernel and a 3×3 colour kernel.

simplest pitched roof is a gabled roof, consisting of just two flat surfaces, joined together to form a single rooftop ridge. Slightly more complicated pitched roofs include the hipped roof, mansard roof, and half-hip roof, but many other variants also exist. Figure 3.9 shows some common types of pitched roofs.

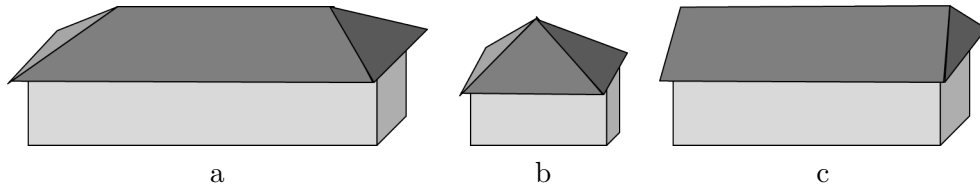


Figure 3.9: (a) A hip roof, (b) a square hip roof, and (c) a gabled roof.

3.5.1 Roof Ridge Identification and Linking

Segmentation starts by identifying straight edges in the image. We used the Canny algorithm for edge detection with a lower threshold of 250 and an upper threshold of 300 [5]. The Canny edge detector is applied to the histogram equalized image, described in Section 3.4.2.

Canny edge detector To understand the significance of these two thresholds, it is important to first understand how the Canny algorithm for edge detection identifies edges in an image. The Canny algorithm first computes the intensity gradient in the image by convolving the image with a discrete differentiation filter; in the proposed algorithm, a Sobel filter was used. Next, the Canny algorithm performs non-maxima suppression, in which it computes the angle of the gradient at each point, and zeros the gradient magnitude of all pixels that are not local maxima. This step has the effect of thinning all of the edges identified in the gradient magnitude image such that each is only one pixel thick. The final step in the Canny algorithm—and the one that utilizes the lower and upper threshold values—is hysteresis thresholding. At this step, each pixel with a gradient magnitude greater than the upper threshold is identified as an edge. Connected pixels whose gradient magnitude is greater than the lower threshold are also identified as being part of an edge.

Alternative edge detectors were also explored. Combining several Prewitt edge filters [15] (each one oriented to find edges along likely angles like 0, 45, and 90 degrees) worked fairly well, as illustrated in Figure 3.10. However, the Canny edge detector, which does not rely on as many assumptions about the angle of the edges, worked well also, especially when

especially short and disconnected edges are removed [5]. Figure 3.11 shows the output of the Canny edge detector on the same image as in Figure 3.10. Although the lines detected are not as straight as those found using the Prewitt edge detector, they do not rely on knowing the orientation of the roofline ridges, as the Prewitt edge detector requires.

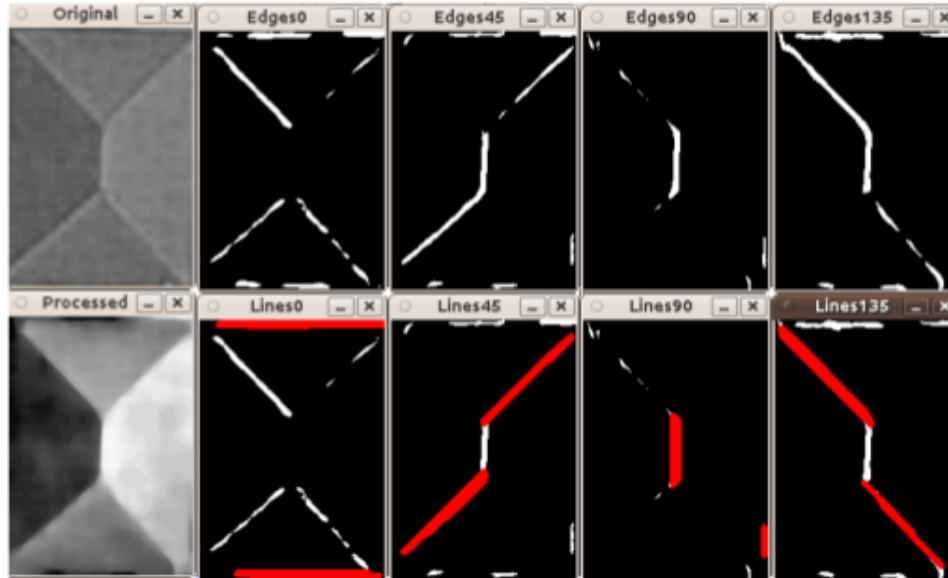


Figure 3.10: Edges and lines detected at several angles using a Prewitt edge detector (top row) and Hough line detector (bottom row).

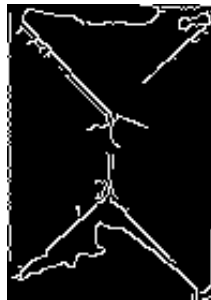


Figure 3.11: Edges detected by the Canny edge detector.

Next, we applied a morphological dilation operation to the binary edge image to connect nearby edges and identify large edge-less regions. These edge-less regions are then used as markers for a watershed segmentation [31] of the rooftop image, with the strong edges that

were previously identified used as barriers. Figure 3.12 illustrates the results of this segmentation step, and Appendix C describes the mark-based watershed segmentation algorithm in detail.

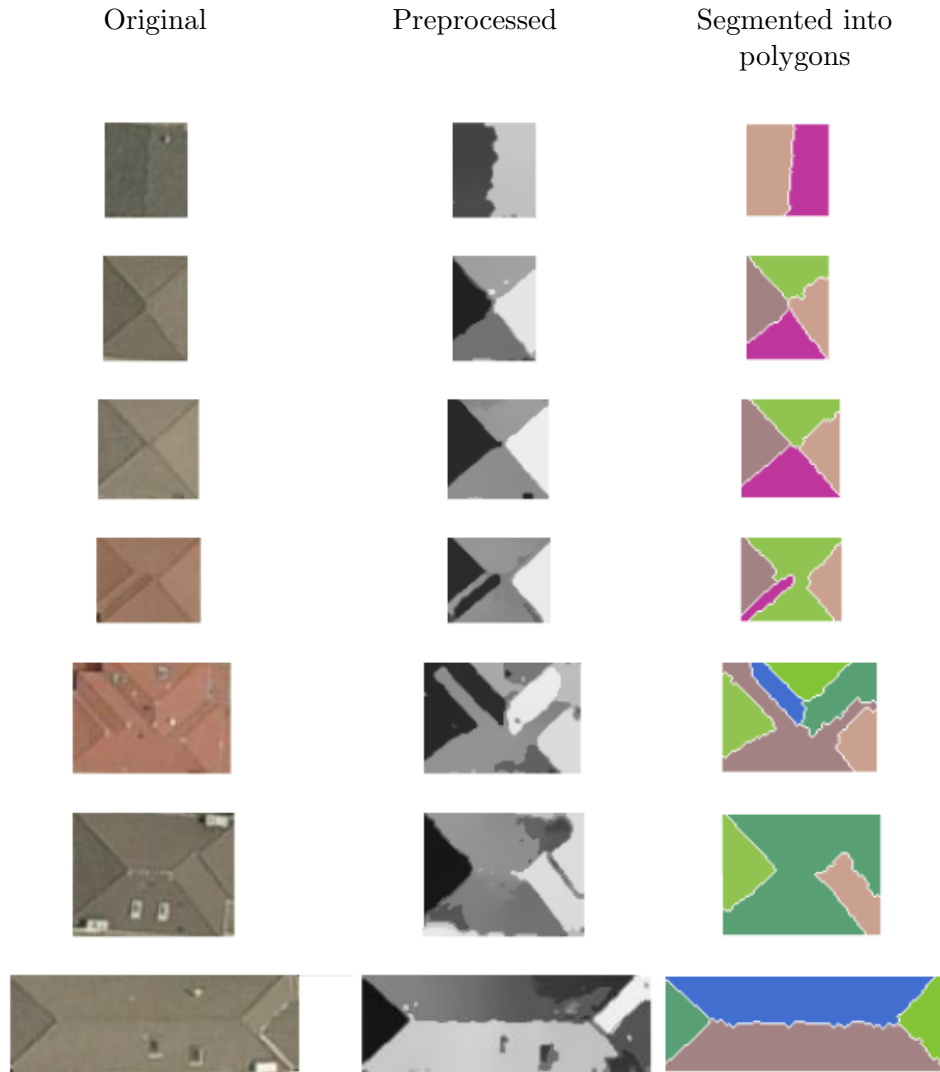


Figure 3.12: Some examples of original, preprocessed, and segmented rooftop images.

Marker-based Watershed segmentation Table 3.2 shows some examples of segmented images produced using this algorithm, along with some metrics relating to the segmentation accuracy.

| % of the incorrectly segmented pixels | Image size (pixels) | Elapsed time (s) | Original, ground truth, and segmented |
|---------------------------------------|---------------------|------------------|---------------------------------------|
| 4.20% | 64×64 | 0.019 | |
| 2.00% | 48×52 | 0.011 | |
| 6.04% | 48×60 | 0.008 | |
| 7.71% | 57×58 | 0.007 | |
| Average: 4.99% | | | |

Table 3.2: Percent of the incorrectly segmented pixels, relative to ground truth segmentations generated by manually segmenting the rooftop images.

3.5.2 Segmentation Into Polygonal Shapes

For each found segment using the watershed algorithm, we apply the Ramer-Douglas-Peucker algorithm [10] to approximate the contour with a polygon of just a few vertices.

Ramer-Douglas-Peucker algorithm The Ramer-Douglas-Peucker algorithm is an iterative algorithm that operates by progressively adding vertices to approximate the input contour until the maximum distance between the simplified and original contour has reached a threshold value. Generally, the algorithm may be described as a series of simple steps that are repeated recursively until an error threshold is met:

1. Find the point c on the actual contour C whose distance, d , is greatest from the approximated contour, \tilde{C} .
2. If d is less than the chosen threshold value t , terminate.
3. Insert c into the approximated contour \tilde{C} .
4. Return to step 1.

Figure 3.13 illustrates how the Ramer-Douglas-Peucker algorithm iteratively adds vertices to the approximated contour to progressively reduce the error in the approximation.

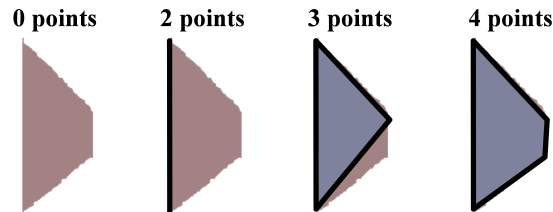


Figure 3.13: The iterative procedure of the Ramer-Douglas-Peucker algorithm as it adds progressively more vertices to approximate a contour.

With an appropriate threshold, the Ramer-Douglas-Peucker algorithm usually results in polygonal regions with three to five vertices, depending on the complexity of the contour. Choosing too small a threshold results in a closer approximation to the segmented contour but it also may result in more vertices in the simplified contour than are really necessary. Choosing too large a threshold, however, results in a poorly approximated contour. Figure 3.14 illustrates the result of using too large a threshold, using an appropriate threshold, and using too small a threshold.

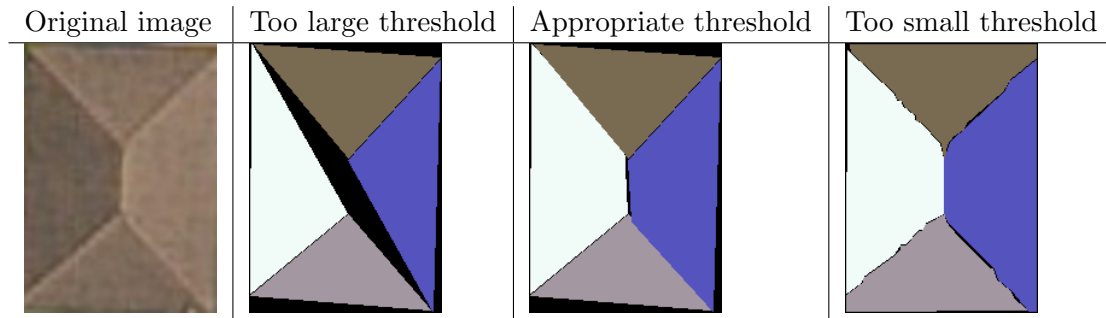


Figure 3.14: The effect of different thresholds chosen for the Ramer-Douglas-Peucker algorithm.

The result of this Ramer-Douglas-Peucker algorithm is a set of simple polygonal shapes, such as those shown in Figure 3.15, which illustrates the results of the edge-detection, segmentation, and shape simplification steps. After this set of polygons is computed, nearby vertices of adjacent polygons are combined into a single shared vertex, thereby producing a graph that partitions the entire rooftop.

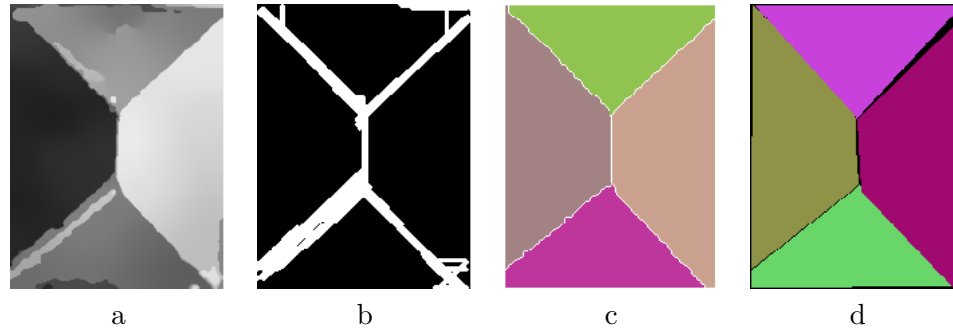


Figure 3.15: (a) The preprocessed image, (b) the image after edge detection, (c) the image after watershed segmentation, and (d) the image after simplifying the regions into simpler polygons.

3.5.3 Representing Rooftop Structure

Once all the polygons that partition the rooftop are identified, they must be represented by an appropriate data structure so that subsequent states can utilize them. An appropriate data structure consists of an ordered pair, $G = (V, E)$. The first set, V , is the set of vertices, each of which consists of a vector in (x, y, z) . The second set E is the set of edges joining vertices in V , and can be represented in software as a vector (e_1, e_2) , where e_1 and e_2 are indexes into a list representing V . This data structure $G = (V, E)$ is an undirected graph.

Because the graph G represents the geometrical arrangement of a rooftop, it is constrained in its structure. For instance, any valid graph G must be a planar graph, meaning that no two edges intersect one-another when the graph is embedded in a plane. Additionally, all of the vertices in the graph must have degree of two or more, because each simple cycle in the graph identifies a polygon, and each vertex is a member of a cycle.

3.6 Deducing Depth From Shading

In the previous section, we partitioned the rooftop into polygonal shapes, each of which represents a planar section of the rooftop surface. Without depth information, however, we cannot estimate the angles between adjacent polygons, and thus cannot estimate the height or pitch of the roof. To estimate these quantities, we must utilize lighting and shading information from the image and its associated acquisition geometry to estimate the surface normals of each polygon. Computer Aided Design (CAD) software and three-dimensional computer games regularly do the reverse of this process— computing the shading of a

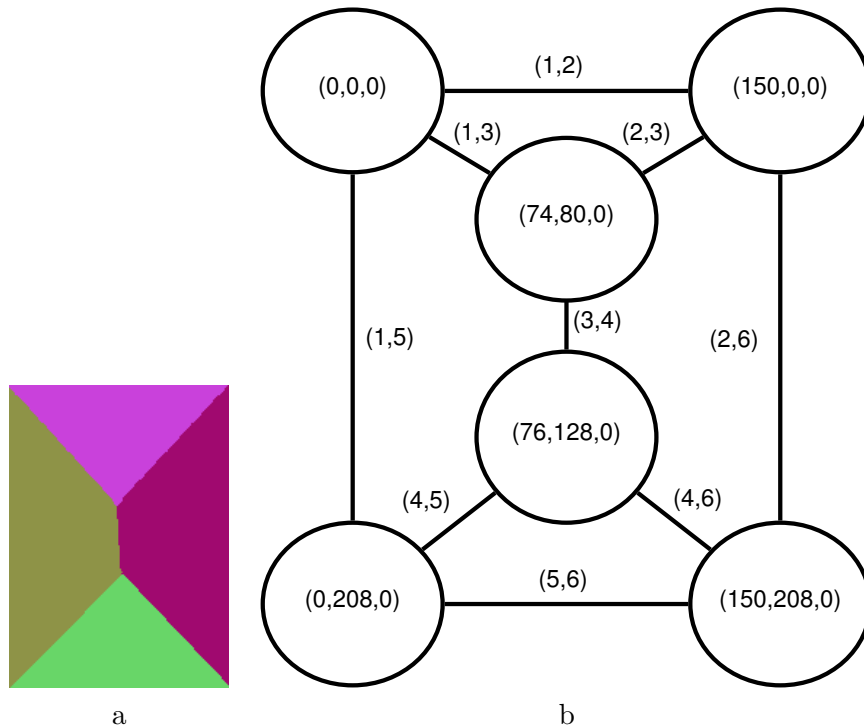


Figure 3.16: (a) An example segmented rooftop, and (b) the resulting graph data structure used to represent the segmentation.

polygon given its orientation—but because multiple possible orientations of a single polygon may result in the same shading, this problem is generally under-constrained. However, we can constrain the problem by considering the resulting shading of the entire rooftop, with all polygons connected, rather than solving for the orientation of each polygon individually.

3.6.1 Shape Models and the BRDF

In general, the shading of a particular material is determined by a BRDF, or Bidirectional Reflectance Distribution Function. Given a ray of incoming light, with radiance L_i , the BRDF calculates the distribution of angles along which the light is reflected from the surface. In general, the BRDF is a function with four parameters, because the amount of incident light that is reflected in a particular direction depends on the angle of the incident light, and the angle to the viewer, both of which consist of both an azimuth angle and a zenith angle.

BRDFs that depend on the value of the incoming and outgoing azimuth angles, rather

than merely their difference, are called anisotropic BRDFs, and are functions of four parameters. Anisotropic BRDFs approximate materials like brushed aluminum, which exhibit different reflectance properties if you were to rotate them on a plane. Many surfaces are well approximated with BRDFs that do not depend on the incoming and outgoing azimuth angles, however, and instead depend only on the difference between them. Such BRDFs are called isotropic, and are functions of only three parameters: the incident zenith angle, θ_i , the reflected zenith angle, θ_r and the difference between the incident and reflected azimuth angles, $\phi = (\phi_i - \phi_r)$. Isotropic BRDFs approximate the reflectance of most materials well, including rooftop shingles.

Another way to look at the BRDF is as a function that defines the ratio of the light incident on a surface from an angle ω_i that is then reflected from the surface in a particular viewing angle ω_o , both of which are measured from the normal of the surface:

$$f_r(\omega_i, \omega_o) = \frac{dL_r(\omega_o)}{dE_i(\omega_i) \frac{dL_r(\omega_o)}{L_i(\omega_i) \cos(\theta_i) d\omega_i}} \quad (3.11)$$

The BRDF for a particular material (e.g. rooftop shingles) can be measured using a device called a gonireflectometer [13], which consists of a movable light source, a movable light sensor, and a stationary pedestal on which to place the material to be measured.

Some research already exists where the BRDF of roofing shingles have been measured [27]. For instance Figure 3.17 shows a roofing shingle in a gonireflectometer ready to have its BRDF measured. Generally, however, measured BRDFs are unavailable or too complex for use in rendering, so certain phenomenological models are used instead to approximate real BRDFs with simpler mathematical functions.

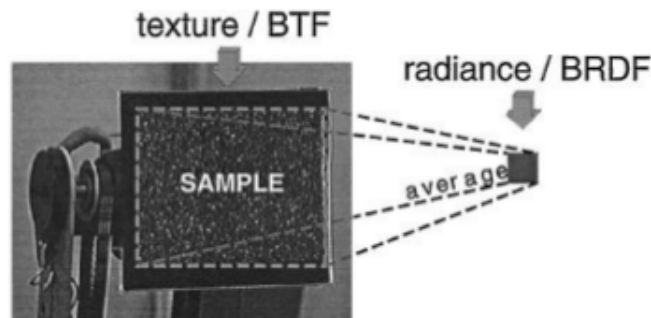


Figure 3.17: Gonireflectometer setup for measuring the BRDF of a rooftop shingle. Figure from [27].

Measurements of the BRDF for rooftop shingles wrapped on a sphere have shown that they have a very flat appearance, with virtually no specular component [27]. In fact, they are flatter than an ideal Lambertian surface, indicating that the Blinn-Phong shader [4] may not be able to account for such a surface. For instance, the spheres in the image below both have the same colour, with the one on the left exhibiting completely diffuse (Lambertian) reflection, and the one on the right exhibiting reflection similar to that seen on rooftop shingles, where the appearance is flatter than Lambertian[27].

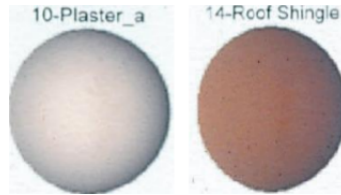


Figure 3.18: Comparison of measured BRDFs for a roughly Lambertian surface (plaster) and rooftop shingles. Figure from [27].

This additional flatness is a characteristic of the roughness of the rooftop shingles. An alternative model that better accounts for materials like rooftop shingles or sandpaper is the Oren-Nayar shading model[3]. The Oren/Nayar shading model incorporates an additional parameter, sigma, which is a measure of the roughness of a surface. The rougher the surface, the flatter it tends to appear on a sphere.

To estimate the depth from shading information, we need to first know the BRDF (Bidirectional Reflectance Distribution Function) that best models the rooftop material. By default, OpenGL (Open Graphics Library) uses a shading model called Blinn-Phong, which models a generic BRDF as the weighted sum of three components: an ambient component, a diffuse component, and a glossy specular component [43].

The Blinn-Phong model does not model the roughness of roof shingles well, however. The roughness tends to reduce the dependence of the perceived brightness of a surface on the viewing angle. A better model for rough surfaces is the Oren-Nayar model [41]. Here the reflected radiance L_r is a function of the incident and reflected angles, as well as two parameters, ρ , the albedo of the surface, and σ , the roughness of the surface. Roofing shingles are well modelled with $\rho = 0.20$ and $\sigma = 0.82$ [27].

The reflected radiance, L_r , then, is given by the formula,

$$L_r = \frac{\rho}{\pi} \cos(\theta_i) (A + (B \max(0, \cos(\phi_i - \phi_r)) \sin(\alpha) \tan(\beta)) L_i, \quad (3.12)$$

where L_i is the irradiance incident on the rooftop, and

$$A = 1 - 0.5 \frac{\sigma^2}{\sigma^2 + 0.33} \quad (3.13)$$

$$B = 0.45 \frac{\sigma^2}{\sigma^2 + 0.09} \quad (3.14)$$

$$\alpha = \max(\theta_i, \theta_r) \quad (3.15)$$

$$\beta = \max(\theta_i, \theta_r) \quad (3.16)$$

The angles θ_i and θ_r are the incident and reflected angles, respectively, both of which are measured from the normal of the surface. The angle θ_i is therefore determined by the position of the light source relative to the surface, and θ_r is determined by the position of the camera relative to the surface. Figure 3.19 illustrates this scenario.

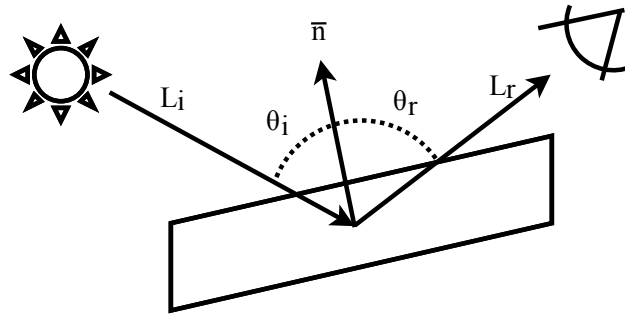


Figure 3.19: Reflected light from a rooftop showing L_i , θ_i , L_r , θ_r , and their relationship to the light source, surface, and viewer.

We extract the lighting information from the acquisition geometry included along with the image meta-data. This acquisition geometry describes the sensor and the sun azimuth and elevation angles [23].

3.6.2 Model Candidate Reconstructions

Once we have BRDF and lighting information, we can render our hypothesized three-dimensional shapes using OpenGL. The rendered image of the three-dimensional model as seen from above is then compared against the original image (with vents removed) to calculate a fitness measure of the hypothesized three-dimensional shape.

Initially, we start with a model where all of the flat surfaces identified earlier lie on the same plane. In subsequent steps, we adjust the z -value of certain vertices while maintaining

the flatness of each identified plane to arrive at alternative hypothesized three-dimensional shapes.

3.6.3 Evaluating Candidate Reconstructions for Goodness-of-Fit

In the algorithm, a subtraction-based correlation with the original image is utilized as a fitness measure, F , where

$$F = \sum_{(x,y) \in I_1} (I_1(x,y) - I_2(x,y))^2. \quad (3.17)$$

I_1 and I_2 are the original and rendered images, respectively. A smaller value for F indicates a better model.

3.6.4 Evolution of a Solution Using Gradient Descent

The algorithm then perturbs the z coordinates of the vertices in the rendered rooftop model slightly and renders it again to reduce F . Figure 3.20 illustrates how the z coordinates of the vertices of the rooftop may be varied to construct different hypothesized rooftop shapes.

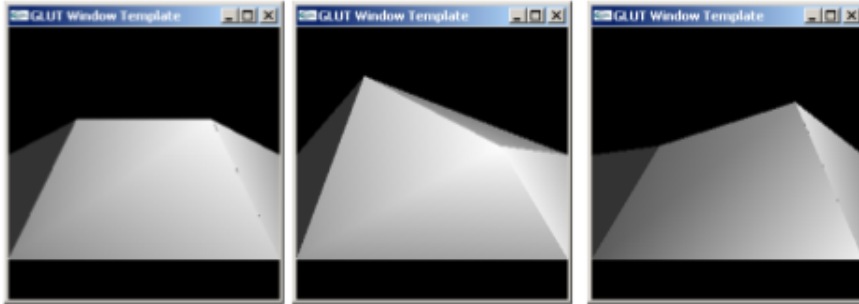


Figure 3.20: Example of a simple rooftop model (seen from the side), with varying z coordinates for its internal vertices.

The z coordinates are perturbed according to a gradient descent equation. The set of all z values for all of the vertices are used to form a vector \mathbf{z} . Thus, given a set of z values \mathbf{z}_i at the i th iteration, the set of z values at the $(i + 1)$ th iteration is given by,

$$\mathbf{z}_{i+1} = \mathbf{z}_i + \nabla F(\mathbf{z}_i) \quad (3.18)$$

where $\nabla F(\mathbf{z}_i)$ is computed by incrementing the z value of each vertex individually and recording the resulting change in the fitness measure F .

We also experimented with random perturbations of one vertex at a time, reverting any change that did not decrease the mean-squared error (MSE), and keeping any change that did decrease the mean-squared error. That approach worked nearly as well as the gradient descent method, but we found that the gradient descent method approached a solution in a more predictable time frame, so it was preferred. Figure 3.21 shows how the mean-squared error decreases with the number of iterations.

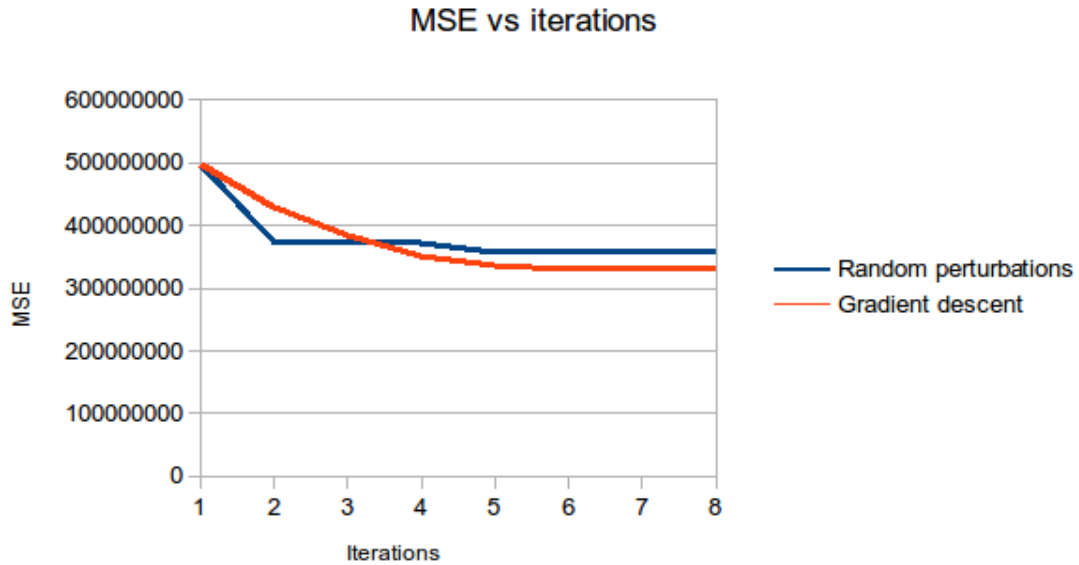


Figure 3.21: Mean-squared error (MSE) relative to the number of iterations.

Once a reconstruction is established that achieves a desired fitness measure, the algorithm stops iterating and the final image is rendered. Alternatively, if for eight consecutive iterations, the fitness measure does not decrease, or if the algorithm has already iterated a maximum number of times (50 iterations seemed to work well) the algorithm also stops. Two examples of reconstructed three-dimensional rooftops using this processes are shown in Figure 3.22. The results on the second row of this figure correspond to the example shown in Figure 3.22.

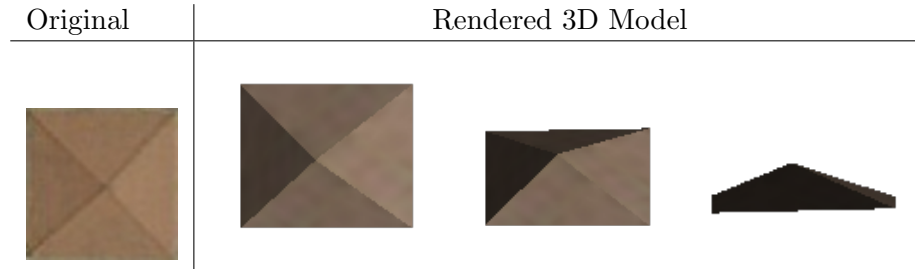


Figure 3.22: Example of the rendered 3D model generated for an image.

3.6.5 Incorporating Constraints Into the 3D Shapes

As the algorithm iteratively guesses and tries new hypothesized three-dimensional shapes, it applies certain general constraints to the three-dimensional shapes. For instance, hypothesized three-dimensional shapes whose exterior planes do not pitch downwards are discarded, because any realistically-shaped rooftop must pitch downwards to deflect rain and snow away from the building.

Similarly, regions where there are two adjacent planes whose shapes are near mirror-images of one another usually indicate that both regions have the same downward pitch. The simplest example is a pitched rooftop consisting of two rectangular planes that connect along one edge.

3.6.6 Why Not Use a System of Linear Equations?

The solution to our gradient descent algorithm is a vector of z coordinates, so an obvious question is to ask whether this problem could be reduced to a system of linear equations. Unfortunately, even the simplest BRDF is non-linear, so one cannot reduce the problem to a system of linear equations.

3.7 Software Implementation

3.7.1 Software Architecture

We implemented the algorithm described in this thesis as a single application written in C++, which utilized several libraries, including Qt, OpenGL, and OpenCV. We used Qt for developing the user interface, OpenGL for rendering hypothesized rooftops, and OpenCV for preprocessing and segmenting the satellite or aerial rooftop images.

The application consists of two important compilation units: `main.cpp`, and `roofrenderer.cpp`. The `main.cpp` file contains code that deals with segmenting the aerial or satellite images, and `roofrenderer.cpp` contains code that renders the hypothesized three-dimensional rooftop models.

3.7.2 Leveraging OpenCV for Rooftop Segmentation

OpenCV includes implementations of many useful computer vision and digital image processing algorithms. For instance, it provides a `cvInpaint()` function that implements the inpainting algorithm we used for removing rooftop vents and skylights.

3.7.3 Rendering Roofs in OpenGL

OpenGL (Open Graphics Library) enables rendering of three-dimensional models, generally using direct lighting, and utilizing graphics processing units (GPUs) on the host PC, if any are available. OpenGL is an open standard and can be used on all major platforms, including Windows, Linux, and Mac OSX. Unlike other image synthesis techniques such as ray tracing or ray casting that generally are incapable of rendering images in real-time, OpenGL can frequently render scenes within a small fraction of a second. For this reason, OpenGL has become popular among game developers, where the game-play consists of three-dimensional scenes rendered in real-time.

OpenGL, by default, defines a “viewing volume in the three-dimensional world space. If any object or any part of an object lies outside that viewing volume, it is clipped and not rendered in the resulting image. The default view volume is a cube centred at the origin, with each side having a length of 2.

OpenGL uses a right-handed coordinate system, meaning that the z-axis points out of the screen towards the viewer. Thus, with the camera located at the origin $((0, 0, 0))$, objects are only visible if they exist in the half of the viewing volume with negative z values. In other words, only those objects that exist entirely in the space where $(-1 < x < 1)$, $(-1 < y < 1)$, and $(-1 < z < 0)$ will not be clipped.

The solution to this problem is to either guarantee that any objects to be rendered fit completely within the default viewing volume, or to specify a different viewing volume by modifying the projection transformation matrix. The projection matrix defines how objects are projected onto the screen, and also defines the viewing volume, outside of which objects

are clipped.

I am able to specify a projection transformation matrix using code like that in Figure 3.23.

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();
```

Figure 3.23: C++ code snippet illustrating how one can specify the viewing volume.

The default projection matrix is the identity matrix but I can specify a custom one using the `glFrustum()` function.

Taking a flat region of the rooftop and applying it to an three-dimensional model in OpenGL as a texture results in a rendered image that looks very similar to the original rooftop image, as shown in Figure 3.24.

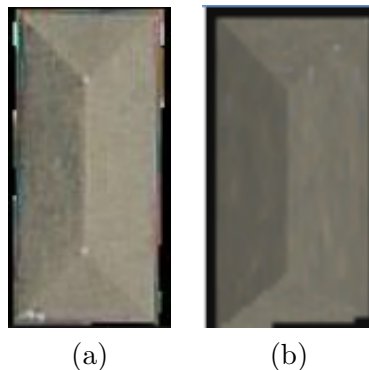


Figure 3.24: (a) the original rooftop image, and (b) the resulting image rendered in OpenGL.

An example of a simple OpenGL code snippet that would draw the rooftop above is shown in Figure 3.25.

It's interesting to note that for each vertex, one must specify the normal vector explicitly using the `glNormal3f()` function.

GL Shader Language (GLSL) and the programmable graphics pipeline When rendering the hypothesized rooftop models, it is important to colour each face of the rooftop appropriately. As mentioned earlier, the default OpenGL shader is based on Blinn-Phong shading, but we would like to use something that better approximates the appearance of a

```
glBegin(GL_QUADS);
// Top surface
glNormal3f(0.0f, 0.5f, 0.5f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, -1.0f, -0.5f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(0.5f, -1.0f, -0.5f);
glTexCoord2f(0.5f, 1.0f); glVertex3f(0.0f, -0.5f, 0.5f);
glTexCoord2f(0.5f, 1.01f); glVertex3f(0.0f, -0.501f, 0.5f);
glEnd();
glBegin(GL_QUADS);
// Right surface
glNormal3f(0.5f, 0.0f, 0.5f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(0.0f, -0.5f, 0.5f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(0.5f, -1.0f, -0.5f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(0.5f, 1.0f, -0.5f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(0.0f, 0.5f, 0.5f);
glEnd();
glBegin(GL_QUADS);
// Bottom surface
glNormal3f(0.0f, -0.5f, 0.5f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(0.0f, 0.5f, 0.5f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(0.5f, 1.0f, -0.5f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-0.5f, 1.0f, -0.5f);
glTexCoord2f(0.01f, 0.0f); glVertex3f(-0.501f, 1.0f, -0.5f);
glEnd();
glBegin(GL_QUADS);
// Left surface
glNormal3f(-0.5f, 0.0f, 0.5f);
glTexCoord2f(0.0f, 1.0f); glVertex3f(0.0f, 0.5f, 0.5f);
glTexCoord2f(1.0f, 1.0f); glVertex3f(-0.5f, 1.0f, -0.5f);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-0.5f, -1.0f, -0.5f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(0.0f, -0.5f, 0.5f);
glEnd();
```

Figure 3.25: C++ code snippet illustrating drawing a simple roof top.

real shingle rooftop (such as an Oren-Nayer shader). To do this, we have to write a custom shader program, which runs on the GPU during rendering.

Modern GPU hardware is heavily pipelined, enabling multiple units within the GPU to execute in parallel. Figure 3.26 illustrates how a set of vertices passed to the GPU are processed by several pipeline stages before eventually resulting in pixels being written to the framebuffer (for display on the screen).

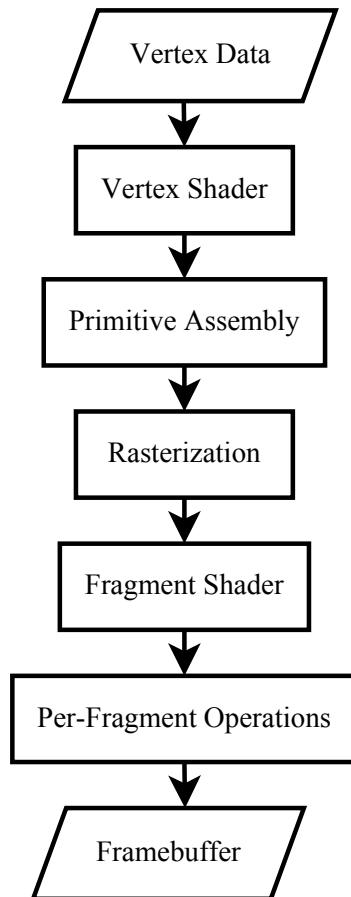


Figure 3.26: A typical graphics rendering pipeline for a modern graphics processing unit.

Most of these pipeline stages typically are fixed in function, but modern graphics cards now enable both the vertex shader and fragment shader to execute small, custom programs written in a C-like language called GLSL (GL Shader Language). GLSL vertex shaders

can be used to modify vertex attributes such as position and color, whereas GLSL fragment shaders are responsible for computing the final pixel color of each rendered pixel. Consequently, GLSL fragment shaders can be used to simulate the reflectance of a custom material.

```

uniform vec4 Ka;
uniform vec4 Kd;
uniform sampler2D lookup;

varying vec3 N;
varying vec3 V;

void main(void)
{
    vec3 Pn = -normalize(V);
    vec3 Nn = normalize(N);

    float roughnessSquared = 0.5;
    float A = 1. - (0.5*roughnessSquared)/(roughnessSquared+0.33);
    float B = 1. - (0.45*roughnessSquared)/(roughnessSquared+0.09);

    vec4 color = vec4(0., 0., 0., 0.);

    vec3 L = normalize(gl_LightSource[0].position.xyz - V);
    float VdotN = dot(Pn, Nn);
    float LdotN = dot(L, Nn);
    float irradiance = max(0., LdotN);
    float angleDifference = max(0., dot(normalize(Pn-Nn*VdotN),
        normalize(L-Nn*LdotN)));
    float lut_val = texture2D(lookup, vec2(VdotN, LdotN)*0.5+0.5).a; \
    color += gl_LightSource[0].ambient*Ka+gl_LightSource[0].diffuse*Kd
        *(A+B*angleDifference*lut_val)*irradiance;

    gl_FragColor = color;
}

```

Figure 3.27: GLSL fragment shader for implementing Oren-Nayer shading (adapted from glslang-library [7]).

3.7.4 Alternatives to OpenGL for Rendering Rooftops

OpenGL renders graphics using direct lighting only. Consequently, it cannot automatically render complex lighting conditions such as inter-reflections resulting from light bouncing off multiple surfaces. An alternative rendering approach, known as ray tracing, may produce much more photo-realistic images, at the expense of greatly increased rendering time.

To explore this possibility, we experimented with rendering hypothesized rooftops using an open-source ray tracing system called PBRT (physics based ray tracer). PBRT offers an unprecedented degree of flexibility for rendering, allowing for rendering rooftops with arbitrary BSDF (Bidirectional scattering distribution function) functions, lighting, haze, non-point-source light sources, and lens effects.

For example, Figure 3.28 shows the same building model rendered in two different ways. The first, rendered with OpenGL, is less realistic-looking than the second, which was rendered using PBRT (ray traced), but the OpenGL rendering took only a small fraction of the rendering time.

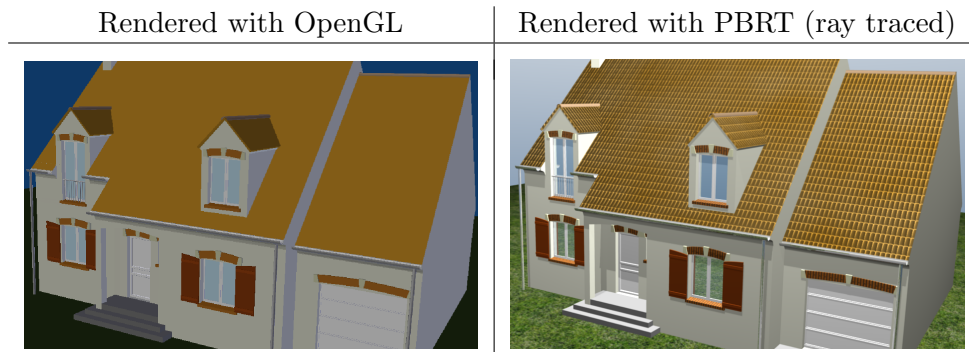


Figure 3.28: Comparison of the same building rendered using OpenGL and PBRT.

3.7.5 Combining OpenCV and OpenGL for Computer Vision

One of the first steps in integrating OpenGL and OpenCV (or any other image-processing library) is to determine how to retrieve a pointer to the image data rendered by OpenGL. This can be achieved using the code snippet in Figure 3.29.

The first line is important, because by default, when `glReadPixels()` aligns the data, it will write to the buffer on a 4-byte boundary. Thus, if the supplied buffer was not already aligned on a 4-byte boundary, `glReadPixels()` would skip up to three bytes at

```
// Tell OpenGL to use 1-byte alignment (i.e. no alignment)
glPixelStorei(GL_PACK_ALIGNMENT, 1);

// Allocate a buffer to store pixel data
std::size_t bufferSize = windowWidth * windowHeight * 3;
char* buf = new char[bufferSize];

// Copy the pixel data to the buffer for use by other libraries
glReadPixels(0, 0, windowWidth, windowHeight, GL_RGB, GL_BYTE, buf);
```

Figure 3.29: C++ code snippet illustrating how one can retrieve pixel data from an image rendered with OpenGL.

the start of the buffer and thus write past the end of the buffer by up to three bytes. The 4-byte alignment can be disabled using the statement below, which simply specifies 1-byte alignment (i.e. no alignment). `glPixelStorei(GL_PACK_ALIGNMENT, 1)`.

In Figure 3.29, we are using a pixel format identified by `GL_RGB` and `GL_BYTE`, which indicates that three bytes will be used for each pixel (one for red, one for green, and one for blue). Other options are available, but this format is convenient for use with a variety of libraries, including OpenCV.

3.7.6 Optimizing Tight Loops in C++ Image Processing Programs

Many computer-vision algorithms involve looping over each pixel in an image and performing a simple image processing operation. These types of algorithms are largely limited by memory latency because the computation done is trivial, but large amounts of data are processed. In this situation, it is important that the CPU's cache is able to effectively hide the slow DRAM (dynamic random access memory) memory latency and avoid cache misses.

For instance, a programmer might write the code in Figure 3.30 to apply a threshold operation to the image (setting all pixels with value less than 127 to 0, and all others to 255). The code works but it takes about 12 seconds to enhance just one image.

The iteration is much slower than necessary because it iterates over y in the inner loop instead of x , which means that memory locations in the image that are very far apart are accessed in rapid succession. The CPU relies on spatial locality in memory to choose what to put in the cache, so jumping around in memory means that there will be a lot of cache misses and performance will suffer as a result.

```
#include <stdio.h>
int main()
{
    const int width = 262144;
    const int height = 1024;
    unsigned char* image = new unsigned char[width*height];
    loadImage(image);
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (image[y*width + x] < 127)
                image[y*width + x] = 0;
            else
                image[y*width + x] = 255;
        }
    }
    saveImage(image);
    delete [] image;
    return 0;
}
```

Figure 3.30: Example of a C++ program that applies a threshold to an image inefficiently.

If, instead of iterating over y in the inner loop, we iterate over x , the memory will be accessed in consecutive locations, and the CPU cache will be able to effectively hide the memory latency and improve performance.

The original code took about 12 seconds to run but after the simple change below, it took about 2 seconds. Thus, this simple change can result in a substantial performance improvement.

These kinds of simple optimizations are something to be mindful of because they do not reduce the clarity of the algorithm, but substantially reduce the run time by more effectively utilizing CPU cache.

3.7.7 Utilizing Qt for Cross-Platform Applications

Qt (pronounced “cute” or “QT”) is a cross-platform C++ application development framework that was first publicly available in 1995 by Trolltech. Qt is available under commercial,

```
#include <stdio.h>
int main()
{
    const int width = 262144;
    const int height = 1024;
    unsigned char* image = new unsigned char[width*height];
    loadImage(image);
    // Transposed order of iteration to reduce cache misses
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            // Calculate the offset into the array just once
            unsigned char& pixel = image[y*width + x];
            pixel = (pixel < 127) ? 0 : 255;
        }
    }
    saveImage(image);
    delete [] image;
    return 0;
}
```

Figure 3.31: Example of a C++ program that applies a threshold to an image efficiently.

GNU General Public License (GPL) and GNU Lesser General Public License (LGPL) licenses, making it free to use for non-commercial purposes, provided that the license and source are distributed according to the chosen non-commercial license.

Computer vision applications such as the one described in this thesis benefit from Qt for a number of reasons. Qt is free, open-source and cross-platform. Being cross-platform means that software can be developed just once, and may then be built for any of the most popular platforms without modifying the source code. For instance, Qt supports Microsoft Windows, Linux, Mac OS X, and even mobile platforms like Symbian and Android. Also, the Qt application framework is very comprehensive, and provides built-in support for displaying image data of a variety of formats.

Specifically, Qt provides a C++ class called `QImage`, which represents an image and can be painted on the screen or printed using Qt's `QPainter` class. `QImage` provides a constructor that accepts a pointer to a buffer containing image data, the pixel format, and the image resolution. This constructor provides a means to read and display image data

generated by a variety of systems, including OpenCV.

3.8 Utilizing GPU Hardware

A large class of image processing algorithms (e.g. blurring, sharpening, edge detection) perform a relatively simple operation on each pixel or neighbourhood of pixels within an image, thus exhibiting a large degree of data parallelism. Using technologies such as CUDA (Compute Unified Device Architecture), OpenCL (Open Compute Language), or Microsoft's DirectCompute, tremendous speedups can be achieved by executing these types of data-parallel image processing algorithms on the multiple cores available in GPUs.

In the implementation of this algorithm, we utilized OpenGL to offload computations pertaining to the rendering of the hypothesized three-dimensional rooftops to the GPU, if present. For platforms without a GPU, however, there is often a software fall-back that may be utilized, instead.

Chapter 4

Experimental Results

As was described in the previous sections, this algorithm consists of several steps: extracting rooftops from a larger image, removing insignificant details, segmenting the rooftop into planar sections, and reconstructing a three-dimensional model. Each of these steps may be considered separately in terms of the results it produces. For instance, it is helpful to consider the accuracy of the first step in its ability to precisely determine the boundaries of rooftops for cropping. Similarly, it is interesting to compare the quality of the segmentation achieved in the third step against a ground truth segmentation.

Additionally, the quality of the three-dimensional reconstructions may be evaluated using various different metrics. For instance, the accuracy of the roof pitch estimations is useful to quantify.

4.1 Examples of Rooftop Segmentation

Figure 4.1 shows several examples of rooftop segmentation performed by the algorithm proposed in this thesis.

4.2 Examples of 3D Reconstructions

Figure 4.2 shows some examples of the 3D reconstructions produced by this algorithm.

In comparison to the output of other shape from shading algorithms, the proposed algorithm tends produce much more accurate 3D models. For instance, Figure 4.3 compares

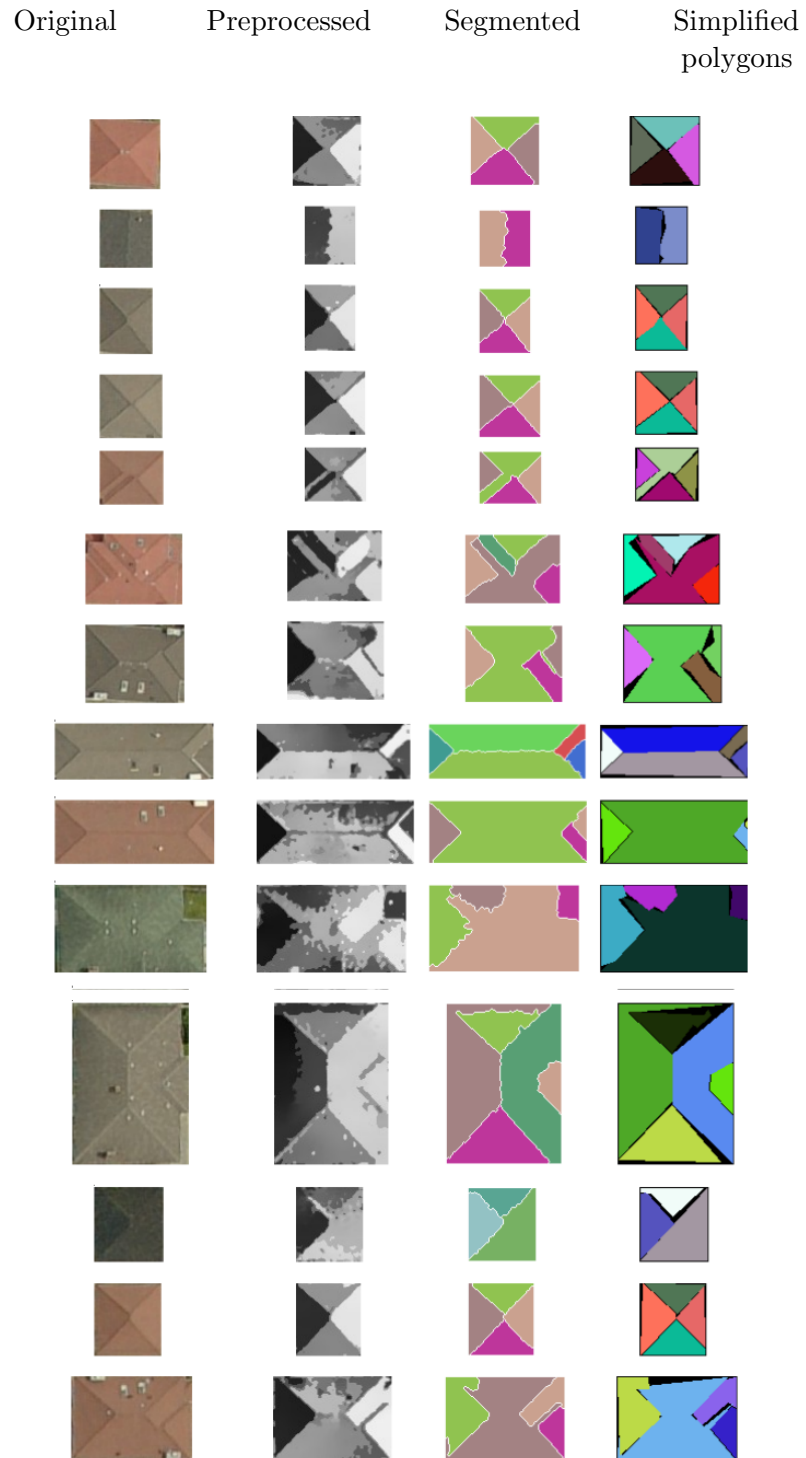


Figure 4.1: Examples of original, preprocessed, segmented, and simplified polygon images.

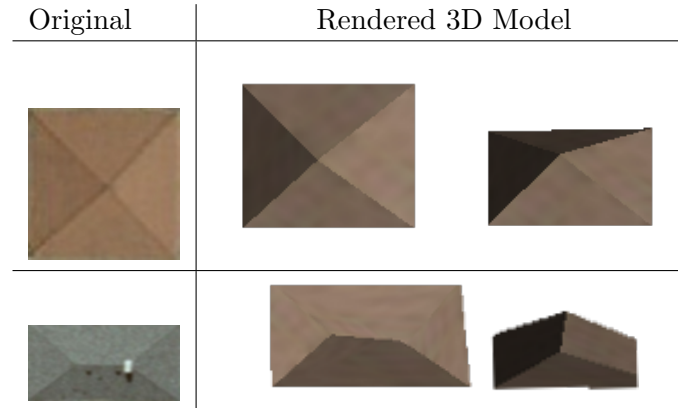


Figure 4.2: Examples of original images and reconstructed 3D models.

the 3D model generated by Bichsel and Pentland against the 3D model produced by this algorithm.

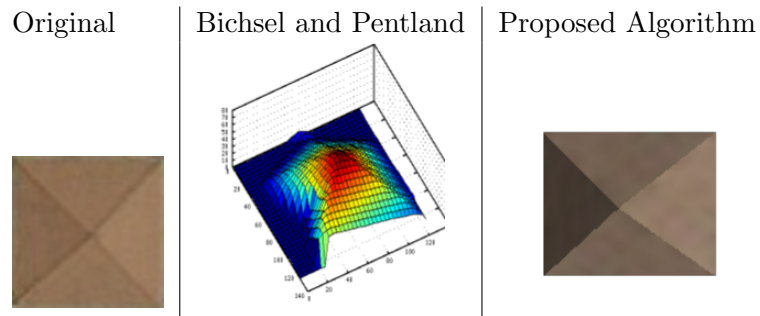


Figure 4.3: Comparison of 3D reconstruction for Bichsel and Penland's algorithm [2] and the proposed algorithm.

4.3 Rooftop Pitch Estimation

The accuracy of the roof pitch estimation was tested for several building examples. Our main constraint here was the ground truth had to be created manually by travelling to the site and physically measuring rooftop slopes. We found that the estimated roof pitches were usually within a few degrees of the true roof pitches.

Table 4.1 lists the true and estimated pitches for some of the tested buildings, and Figure 4.4 shows some of the imagery used.

One strategy used to measure the actual pitch or rooftops is to take photos of the houses,





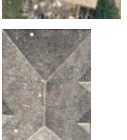
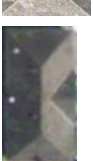
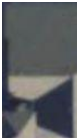



| Rooftop | Actual Pitch | Estimated Pitch | Image |
|----------------------------|--------------|-----------------|---|
| #1 (Chilcotin House) | 26 ° | 28 ° |  |
| #2 (Kelowna Res.) | 26 ° | 28 ° |  |
| #3 (Curtis Road #1) | 29 ° | 30 ° |  |
| #4 (Curtis Road #2) | 30 ° | 30 ° |  |
| #5 (House on 194A St) | 20 ° | 23 ° |  |
| #6 (Cross Twn Food Market) | 26 ° | 28 ° |  |
| #7 (Fort Lang. Hall) | 28 ° | 28 ° |  |
| #8 (Aldergrove Store) | 47 ° | 45 ° |  |
| #9 (Langley Hall) | 44 ° | 45 ° |  |
| #10 (C. & G. Howe School) | 16 ° | 18 ° |  |
| Average. Error: | | 1.5 ° | |

Table 4.1: Rooftop pitch estimation.

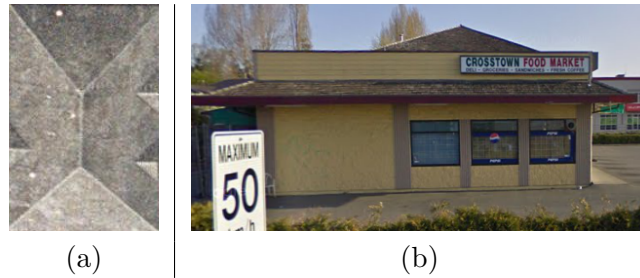


Figure 4.4: The Crosstown Food Market from (a) above and (b) street level. These images are from Google Maps [16].

and then using the measurement tool of the Gimp image processing application. This is shown in Figure 4.5. Using this approach, it is important to consider the impact of tilt or pan in the image relative to the rooftop surface.

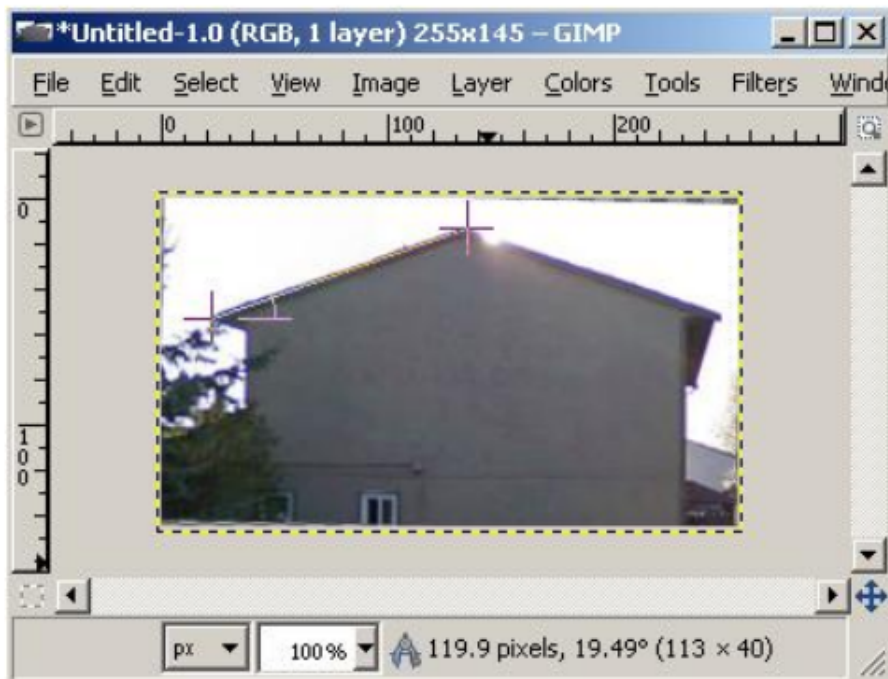


Figure 4.5: Measuring roof pitch using the Gimp.

To estimate rooftop pitches more accurately, the algorithm rounded its estimated pitches to one of the common rooftop pitches used in construction. Specifically, rooftop pitches are generally specified in terms of the integer number of inches the roof rises over 12 inches of horizontal run. Consequently, there are only 12 distinct roof pitches that are commonly

used, ranging from 1:12 to 12:12, with most roofs being pitched near the middle of that range. Figure 4.6 illustrates this discrete set of likely roof pitches.

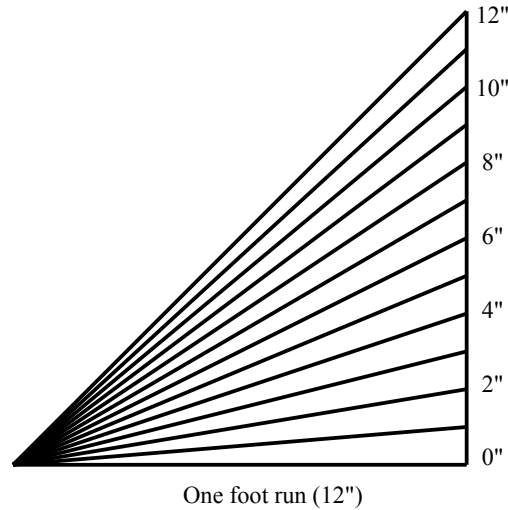


Figure 4.6: Roofs are designed to have only certain discrete pitches, identified by a whole number of inches in vertical rise over 12 inches of vertical run.

Consequently, given an estimated surface gradient, the algorithm computes the angle, θ between vertical and the surface normal, or, equivalently, the angle between the surface and horizontal. Using that angle, the algorithm rounds the pitch to the nearest pitch that can be expressed with the ratio $n/12$, where n is a positive integer. This is accomplished using the formula,

$$n = \lfloor 12.0 \tan(\theta) + 0.5 \rfloor. \quad (4.1)$$

4.4 Run-Time Considerations

The implementation of this algorithm operates in a reasonable time and with reasonable memory requirements. Our test machine was a Toshiba Satellite laptop with an AMD (Advanced Micro Devices) Athlon(tm) 64 X2 processor running at 800 MHz. Our implementation was not multi-threaded, so it utilized only one of the two cores available on the machine. For 160×160 pixel colour images, the algorithm takes about 5.8 seconds to complete.

| Rooftop Image | Runtime | Maximum memory usage |
|---------------|---------|----------------------|
| #1 (160x224) | 5.970s | 23860K |
| #2 (160x160) | 4.996s | 23632K |
| #3 (160x224) | 6.704s | 23856K |
| #4 (160x160) | 5.577s | 23632K |

Table 4.2: Processor and memory utilization.

As shown in Table 4.2, the algorithm took about 5.8s to complete, and used approximately 23 MB of memory. Most of this memory was accounted for by libraries linked to the executable (e.g. OpenCV, OpenGL, GLUT), which could be linked dynamically, instead, if multiple instances of the algorithm were to run concurrently.

4.5 Limitations and Assumptions

Minimum Image Resolution This algorithm requires sufficient resolution and contrast to identify rooftop features. We found that the algorithm works well with images that have a minimum resolution of about 0.2m per pixel. At that resolution, edges that are shorter than approximately 1 m are sometimes lost, because such short edges are ignored as potential noise or combined with other nearby edges.

Image Captured on a Clear Day Additionally, those images taken on clear days in the morning and evening typically have the highest contrast, and therefore result in more robust segmentation. On overcast days, especially around noon, the light incident on the rooftop is very diffuse and tends to illuminate all faces of a nearly equally, resulting in images that have too low contrast to be used by this algorithm.

Nadir Image Capture or Orthorectified Image The algorithm assumes that the input images are taken from directly above — or that the images have been orthorectified to remove any distortion due to perspective projection — such that the images contain approximately orthogonal projections of the buildings and rooftops.

Rooftop Complexity Additionally, the proposed algorithm currently works reliably only for relatively simple rooftop shapes, such as those with two to five planar regions that are connected edge-to-edge. Ledges and sharp discontinuities in the rooftop surface cannot easily be accounted for, because, in general, two disconnected rooftop surfaces that are

separated only by a vertical gap look exactly the same as two connected rooftop surfaces when seen from above. Therefore, the algorithm assumes that the rooftop is a continuous surface, without sheer vertical drops or ledges.

4.6 Sensitivity Analysis and Portability Issues

The algorithm utilizes several parameters that must be set to appropriate values based on the type of input images being processed. Table 4.3 describes these parameters, and lists the values that were found to work well for the images tested. Table 4.4 then lists the effects increasing or decreasing each parameter has on the execution of the algorithm.

| Section | Parameter | Description | Value |
|---------|------------------------------------|--|--|
| 3.4.1 | Vent high threshold | Pixel intensity (ranging from 0 to 255) threshold above which pixels are marked as belonging to a light-coloured vent. | 180 |
| 3.4.1 | Vent low threshold | Pixel intensity (ranging from 0 to 255) threshold below which pixels are marked as belonging to a dark-coloured vent | 90 |
| 3.4.1 | Inpaint radius | The radius of the region considered when inpainting each pixel | 7.0 |
| 3.4.2 | Bilateral filter aperture size | The size of the mask used when applying the bilateral filter to each pixel in the image. | 5×5 |
| 3.4.2 | Spatial sigma | Affects the degree to which the value of pixels far from the central pixel in the mask are considered. | 40.0 |
| 3.4.2 | Intensity sigma | Affects the degree to which the value of pixels that are very different in intensity to the central pixel are considered. | 5.0 |
| 3.5.1 | Canny low threshold | Threshold for labelling pixels adjacent to already-labelled strong edges as edges. | 150 |
| 3.5.1 | Canny high threshold | Threshold for labelling pixels as strong edges. | 300 |
| 3.5.1 | Structuring element | The size of the structuring element used for morphological erosion and dilation operations. | 3×3 |
| 3.5.1 | Dilation iterations | The number of times morphological dilation is applied to the binary image. | 8 |
| 3.5.1 | Watershed seed region minimum area | The minimum size of a connected component region that may be selected as a seed region for watershed segmentation. | 40 pixels squared |
| 3.5.2 | Approximation accuracy threshold | The minimum approximation accuracy required for the Ramer-Douglas-Peucker algorithm [10] to terminate. | $0.0035 \times (\text{contourlength})$ |
| 3.5.3 | Maximum vertex combining distance | The furthest that two points on adjacent polygons can be and still be combined into a single vertex in the graph partitioning the rooftop. | $0.0001 \times w \times h$ |
| 3.6.3 | Maximum iterations of algorithm | The maximum number of iterations the rendering stage of the algorithm will perform while attempting to minimize the mean-squared error between the rendered and original images. | 500 |

Table 4.3: Free parameters of the algorithm.

| Section | Parameter | Change Trend ↓ | Change Trend ↑ |
|---------|------------------------------------|--|--|
| 3.4.1 | Vent high threshold | Some light areas, typically surrounding light-coloured vents will be inpainted unnecessarily. | Some light-coloured vents may not be inpainted, and the edges of others may not be inpainted. |
| 3.4.1 | Vent low threshold | Some dark-coloured vents may not be inpainted, and the edges of others may not be inpainted. | Some dark areas, typically surrounding dark-coloured vents will be inpainted unnecessarily. |
| 3.4.1 | Inpaint radius | Minor details of the roof surrounded areas to be inpainted may be accentuated in the inpainted region. | The inpainted region may lose definition of sharp edges that pass through it. |
| 3.4.2 | Bilateral filter aperture size | The image will generally be smoothed less. | The image will generally be smoothed more. |
| 3.4.2 | Spatial sigma | The image will generally be smoothed less. | The image will generally be smoothed more. |
| 3.4.2 | Intensity sigma | Strong edges in the image will be smoothed more. | Strong edges in the image will be smoothed less. |
| 3.5.1 | Canny low threshold | More edges will be detected. | Fewer edges will be detected, and fewer strong edges will be linked. |
| 3.5.1 | Canny high threshold | More strong edges will be detected. | Fewer strong edges will be detected. |
| 3.5.1 | Structuring element | Fewer edges will be joined together and linked. | More edges will be joined together and linked, but some small regions may be erased. |
| 3.5.1 | Dilation iterations | Fewer edges will be joined together and linked. | More edges will be joined together and linked, but some small regions may be erased. |
| 3.5.1 | Watershed seed region minimum area | Smaller rooftop regions may be identified. | Some small rooftop regions may be misidentified and included in the segmentation. |
| 3.5.2 | Approximation accuracy threshold | The perimeter of each planar region identified will consist of more points. | The perimeter of each planar region identified will consist of fewer points, making rendering and optimization faster. |
| 3.5.3 | Maximum vertex combining distance | The graph of the rooftop will be more complex. | The graph of the rooftop will be simpler and faster to render. |
| 3.6.3 | Maximum iterations of algorithm | A less accurate 3D model may be estimated. | A more optimal 3D model may be estimated, at the expense of greater computation time. |

Table 4.4: Change trends of the free parameters.

Chapter 5

Conclusions

5.1 Summary of Contributions

This thesis presented an algorithm for estimating the three-dimensional shape of a pitched roof, using just a single image. This approach differs from previous approaches, many of which require specialized equipment for laser ranging, or multiple images taken from different vantage points. We found that this algorithm usually accurately estimates the three-dimensional shape of gabled rooftops, especially when the images have high resolution and good contrast.

Our objective is to create the three-dimensional models of building rooftops given the orthographic view of a rooftop. This work adds on to the previous work in our group that addressed the height estimation of buildings (average height from the borders of the roof-lines to ground). Using the rooftop models, a more complete reconstruction using singular electro-optical imagery is possible.

5.2 Potential Applications

Estimating the three-dimensional shape of pitched roofs from nadir aerial or satellite imagery is of interest to both governments and industry. The most direct applications of this technology is to realistic three-dimensional map building, but other less obvious applications may also be possible, such as in architectural design, real estate marketing, and insurance assessment.

5.2.1 Map Generation and City Planning

Google Maps has enjoyed incredible success on the Web as a source of reliable and free location and direction information. This service combines street maps, satellite imagery, and aerial imagery from about 800–1500 feet from the ground into a relatively easy-to-use web application hosted on Google’s servers [16]. A closely related product, Google Earth, also incorporates three-dimensional building and landscape models, which are designed manually using 3D modelling software such as Google’s Building Maker [28].

With the increased prevalence of hand-held devices such as smart-phones and tablet computers, many users now access Google Maps using mobile devices. Consequently, in addition to their Google Maps web page, Google has also released a Google Maps application for Android-based smart phones. The screen-shot shown in Figure 5.1 was taken from an Android-based smart phone running the Google Maps application, and illustrates one way in which the Google Maps application uses 3D-rendered buildings to help users orient themselves in a map.

Similarly, a competing product to Google Maps, named Microsoft Bing Maps (formerly Microsoft Virtual Earth), also incorporates some three-dimensional building models into their map software. They have utilized software by Vexcel Imaging to combine thousands of aerial images acquired using multi-ray aerial photography to automatically generate three-dimensional building models for inclusion in Bing Maps [42].

5.2.2 Architectural Design

Some structural engineers and architects use software like “Truss 3D” from FINE civil engineering, shown in Figure 5.2, to design pitched roofs like those commonly found on wood-framed residences. These applications typically enable an engineer to view a rendered three-dimensional model of a building being designed. It may also be useful to estimate the three-dimensional model of a building in an aerial or satellite image so that it may be imported into the software for further design.

For instance, if a customer is working with an architect to design a new home, he or she may request that their new home have a rooftop shaped like another house they have seen. Normally, if the architect didn’t have plans for that house, they would have to manually enter the shape of the example rooftop into their design software.

Using a framework like the one proposed in this thesis, however, the architect may be

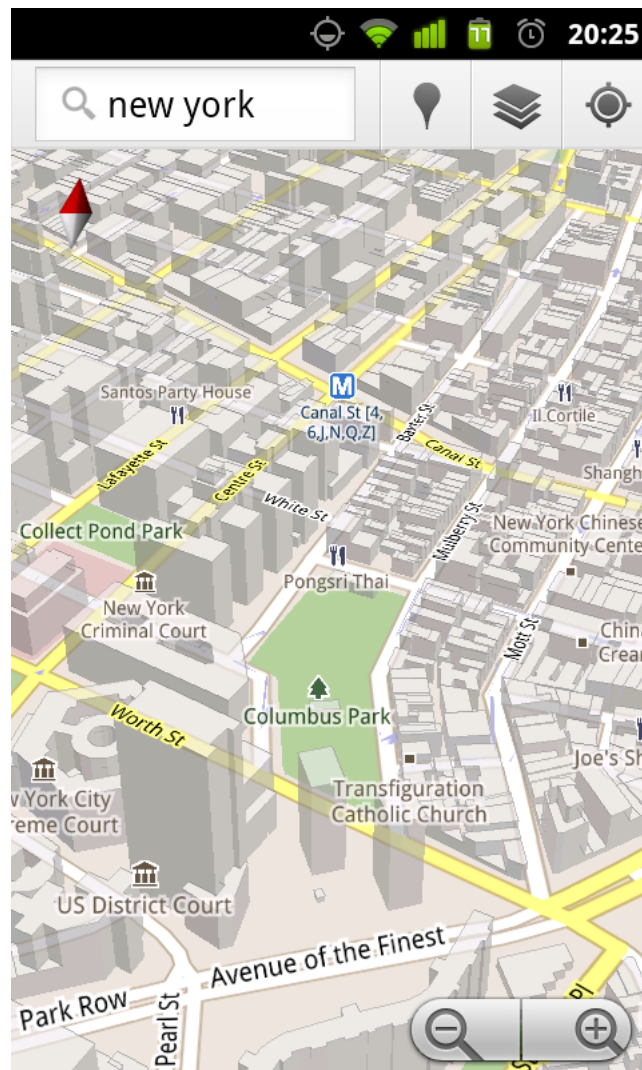


Figure 5.1: A screen-shot of the Android Google Maps 5.0 application, captured by user Wikipedia user 'Zouzzou'.

able to automatically import the estimated shape of the rooftop given a satellite or aerial image of the home, such as one found easily through Google Maps. Although the proposed framework relies on the availability of image acquisition geometry information (particularly the position of the sun relative to the image) to estimate the height and slopes of rooftop surfaces, if that information was not available, it could be estimated by a human operator to get an approximate three-dimensional reconstruction.

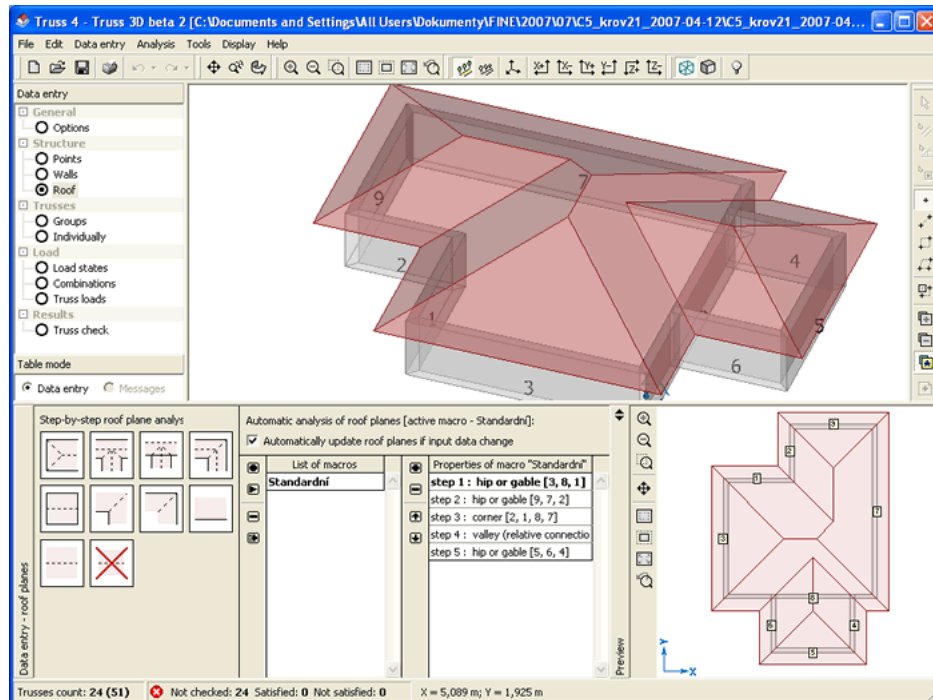


Figure 5.2: A screen-shot of the “Truss 3D” software from FINE civil engineering software [12].

5.2.3 Real Estate

There is a growing trend in real estate for real estate brokers to use the Internet, and especially the World Wide Web to communicate with clients and promote properties for sale [25]. The ability to automatically construct three-dimensional models of residential areas where pitched roofs are common would enable the creation of interactive tours of areas where real-estate is for sale to potential buyers. Web technologies with 3D graphics capabilities, such as VRML (virtual reality markup language), Adobe Flash, Microsoft Silverlight,

and WebGL (Web Graphics Library) all provide potential opportunities for sharing such immersive tours through buyers' web browsers.

5.2.4 Insurance Assessment

Insurance companies currently employ assessors to travel to insured properties to estimate building properties including rooftop models and slopes of each piece. The volume, variability, complexity of digital geo-spatial images, and the large liability for supporting assessor site visits mandate fully-automated three-dimensional rooftop modelling and reconstruction capabilities.

Some architectural design software enables the user to estimate the cost of building the home being designed. For instance, HomeDesign is a piece of software that offers a cost estimating feature [35]. If the size and shape of a home could be estimated using the algorithm described in this thesis, such information could be utilized — along with other relevant information such as flooring, lumber type, etc. — to estimate the cost of a building and spare the expense and liability of supporting assessor visits.

5.2.5 Population Estimation

Medical aid organizations utilize population estimates when deciding how to best distribute aide. Traditionally, if census data is unavailable, population estimates are computed after a time-consuming and costly survey performed by physically visiting each home. Many areas that are served by aide organizations are difficult to access, change frequently, and do not have accurate census data, so estimating population density using traditional methods is not ideal.

Automated systems that are capable of identifying the number and type of buildings in a region from satellite or aerial imagery can be an appealing alternative to manual surveys for rapidly and inexpensively estimating population density. The Medecins Sans Frontieres organization, for instance, relies on manual surveys for population estimates to plan its interventions [17].

5.3 Future Research

There are several opportunities for future work that derive from this work. For instance, the implemented algorithm currently requires that the direction of the light source (the sun) be specified. A desirable enhancement to this algorithm would be to remove that requirement, and instead automatically deduce the direction of the light source (the sun), perhaps with the aide of several different images taken of the same buildings from different perspectives. Another possible enhancement would be to enable the algorithm to combine single images from different orientations or times of day to achieve additional accuracy. One of the primary benefits of this algorithm over others is that it only requires one image. However, often multiple images may be available, and it would be beneficial to be able to utilize them.

Appendix A

Telea Inpainting

This method calculates the value of each pixel a in the region to be inpainted, Ω , by using a weighted average of the estimates computed considering each pixel b in the surrounding neighbourhood B . Figure A.1 illustrates this setup involving a , Ω , b , and B .

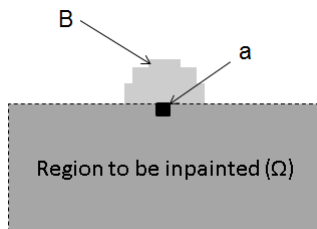


Figure A.1: Telea inpainting.

For each individual pixel b in B , Telea's inpainting algorithm calculates an estimate $I_b(\tilde{a})$ for the value of the pixel to be inpainted, a , by performing a simple linear approximation involving the value of b and the image gradient $\Delta I(b)$,

$$I_b(\tilde{a}) = I(b) + \Delta I(b)(a - b). \quad (\text{A.1})$$

The algorithm then computes a weighted average of these estimates for each point b in B using the weights $w(a, b)$ and the equation

$$I(\tilde{a}) = \sum_{b \in B} w(a, b) I_b(\tilde{a}). \quad (\text{A.2})$$

such that

$$\sum_{b \in B} w(a, b) = 1.0. \quad (\text{A.3})$$

Applying this equation to each pixel inside the boundary of the region to be inpainted, Telea's inpainting algorithm is able to inpaint the outer-most one-pixel-thick strip within the region Ω . Consequently, Ω can be reduced in size by one pixel on all sides, such that it contains only those pixels that have yet to be inpainted. This advancement of the boundary of Ω , which we can denote $\delta\Omega$, into the interior of Ω is performed using the fast marching method (FMM). Telea's algorithm then repeats, until the inpainted region Ω contains no pixels.

Appendix B

Bilateral Filtering

Bilateral filtering convolves parts of the image without sharp discontinuities with a Gaussian kernel to smooth them, while convolving parts of the image with sharp discontinuities with a kernel that doesn't smooth those discontinuities [38]. To achieve this, the bilateral filter must recompute the weights in its convolution kernel for each pixel in the image, taking into account not only the Euclidean distance of each weight from the centre of the kernel (as in a Gaussian filter) but also the differences in pixel intensity of the central pixel relative to neighbouring pixels. Consequently, pixels with intensities that are very different from the intensity of the central pixel receive a smaller weight in the convolution kernel even though their Euclidean distance to the central pixel may be small.

The bilateral filtering of a pixel a in an image A is given by

$$I(\tilde{a}) = \frac{1}{\eta} \sum_{b \in \Omega} w(a, b) \phi(I(a), I(b)) I(b) \quad (\text{B.1})$$

where

$$\eta = \sum_{b \in \Omega} w(a, b) \phi(I(a), I(b)) \quad (\text{B.2})$$

and Ω is the set of pixels surrounding a that fall under the mask being computed for the bilateral filter. For instance, if the bilateral filter was using a 5×5 mask size, then Ω would consist of a 5×5 set of pixels centred on a . The function $w(a, b)$ measures the Euclidean distance between a and b , and $\phi(I(a), I(b))$ measures the difference in pixel intensity between a and b .

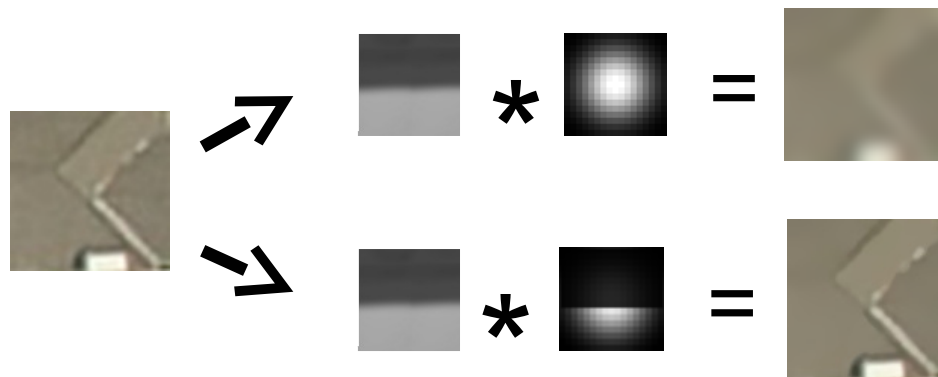


Figure B.1: Comparison of the preservation of strong edges by the Gaussian filter (top) and Bilateral filter (bottom).

Appendix C

Marker-based Watershed Segmentation

Watershed segmentation operates on the monochrome gradient image, $|grad(I)|$, of the rooftop. Edges, or regions of high variability appear lighter, and flat regions appear darker in the gradient image. This gradient image may be visualized as a topographic relief, with valleys and ridges corresponding to the dark and light regions of the image, respectively. The altitude of a point in the topographic relief is therefore proportional to the intensity value of the corresponding pixel in the gradient image. Conceptually, watershed segmentation operates by emulating the way water would flow into catchment basins in the topographic relief. Each edge-less marker region identified earlier may be thought of as a coloured stream of water being poured on this topographic relief from the position of the marker. As the basins fill with differently-coloured water, the water level in each basin rises, and eventually adjacent basins begin to overflow. At that point, the watershed segmentation algorithm defines a barrier between the two adjacent basins to prevent them from mixing. As this process continues, eventually all of the basins overflow, such that the entire topographic relief is covered with differently-coloured basins of water separated by barriers. When this occurs, the entire image can be thought of as being segmented.

We used a particular variant of marker-based watershed segmentation developed by F. Meyer [31]. In this implementation, the algorithm proceeds by executing the following steps:

1. The set of marker regions, $\{M\}$, to start the segmentation process is selected.
2. Each marker region, M_i , is assigned a unique label, and each pixel within the region

is also assigned that label.

3. The neighbouring pixels of each marker region, S_i , are inserted into a priority queue, S , with the priority corresponding to the intensity of the pixel.
4. The pixel, s with the highest priority in S (corresponding to the darkest pixel in the gradient image) is selected.
5. If the pixels that are adjacent to s and labelled all have the same label, s is assigned that label.
6. The non-labelled neighbours of s are then added to S with priorities corresponding to their intensity levels.
7. The previous three steps are repeated until S is empty.

Bibliography

- [1] H. S. Alhichri and M. Kamel. Integrated image and graphics technologies. chapter Multi-resolution image registration using multi-class Hausdorff fraction, pages 393–405. Kluwer Academic Publishers, Norwell, MA, USA, 2004.
- [2] M. Bichsel and A. P. Pentland. A simple algorithm for shape from shading. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pages 459–465, 1992.
- [3] Z. Blair and P. Saeedi. Towards automatic 3d reconstruction of pitched roofs in monocular satellite/aerial images. In *International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 2012.
- [4] J. F. Blinn. Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198, 1977.
- [5] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):679–698, 1986.
- [6] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *International Conference on Computer Vision (ICCV)*, pages 1197–1203, 1999.
- [7] Various contributors. The glslang shader library. <http://code.google.com/p/glslang-library/>, 2012.
- [8] M. Cote and P. Saeedi. Automatic rooftop extraction in nadir aerial imagery of suburban regions using corners and variational level set evolution. *IEEE transactions on geoscience and remote sensing*, 99:1–16, 2012.
- [9] M. Cote and P. Saeedi. A star-corner algorithm for building extraction in satellite/aerial images. In *International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 2012.
- [10] 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. *Canadian Cartographer*, 10:112–122, 1973.

- [11] C. Feng. Semi-automatic 3d reconstruction of piecewise planar building models from single image. In *International Conference on Construction Applications of Virtual Reality (CONVR)*, 2010.
- [12] FINE. Truss 3d — roof truss design truss4 — fine. <http://www.finesoftware.eu/roof-truss-design/truss-3d/>, 2012.
- [13] S. C. Foo. A gonioreflectometer for measuring the bidirectional reflectance of materials for use in illumination computations. Master’s thesis, Cornell University, 1997.
- [14] J. M. Geusebroek, R. van den Boomgaard, A. W. M. Smeulders, and H. Geerts. Color invariance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1338–1350, 2001.
- [15] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2001.
- [16] Google. Blurry or outdated imagery. <http://support.google.com/earth/bin/answer.py?hl=en&answer=21417>, 2012.
- [17] C. Grundy. Validation of satellite imagery methods to estimate population size. *New Scientist*, 214(2867):18, 2012.
- [18] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.
- [19] L. Hazelhoff and P. H. N. de With. Robust model-based detection of gable roofs in very-high-resolution aerial images. In *Proceedings of the 14th international conference on Computer analysis of images and patterns - Volume Part I, CAIP’11*, pages 598–605, Berlin, Heidelberg, 2011.
- [20] C. Heipke, C. Steger, and R. Multhammer. A hierarchical approach to automatic road extraction from aerial imagery. In *Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision II*, pages 222–231, 1995.
- [21] B. K. P. Horn. Obtaining shape from shading information. In *Shape from Shading*, pages 123–173, 1975.
- [22] B. K. P. Horn. *Robot Vision (MIT Electrical Engineering and Computer Science)*. The MIT Press, mit press ed edition, 1986.
- [23] M. Izadi and P. Saeedi. 3d polygonal building detection in monocular satellite images. In *IEEE Transactions on Geoscience and Remote Sensing (IEEE-TGARS)*, 2012.
- [24] R. T. Whitaker J. E. Cates and G. M. Jones. Case study: an evaluation of user-assisted hierarchical watershed segmentation. *Medical image analysis*, 9(6):566–78, December 2005.

- [25] G. D. Jud, D. T. Winkler, and G. S. Sirmans. The impact of information technology on real estate licensee income. *Journal of Real Estate Practice and Education*, 5(1):1–16, 2002.
- [26] D. A. Kleffner and V. S. Ramachandran. On the perception of shape from shading. *Perception & Psychophysics*, 52:18–36, 1992.
- [27] J. J. Koenderink, S. K. Nayar, B. van Ginneken, and K. J. Dana. Reflectance and texture of real-world surfaces. In *Image Understanding Workshop*, pages 1419–1424, 1997.
- [28] M. Limber and M. Simpson. Introducing google building maker. <http://googleblog.blogspot.ca/2009/10/introducing-google-building-maker.html>, 2009.
- [29] H. Liu. Derivation of surface topography and terrain parameters from single satellite image using shape-from-shading technique. *Computers & Geosciences*, 29(10):1229–1239, 2003.
- [30] F. Meyer. Color image segmentation. In *Proc. of the 4th Conference on Image Processing and its Applications*, pages 302–306, 1992.
- [31] F. Meyer. Integrals, gradients and watershed lines. In J. Serra and P. Salembier, editors, *Mathematical morphology and its applications to signal processing*, pages 70–75, 1993.
- [32] Y. Morgenstern, R. F. Murray, and L. R. Harris. The human visual systems assumption that light comes from above is weak. *Proceedings of the National Academy of Sciences (PNAS)*, 108(30):12551–12553, 2011.
- [33] L. M. Moskal. Lidar fundamentals, part one and part two. Technical report, Center for Urban Horticulture, College of Forest Resources, Univ of Washington, Seattle, WA., 2008.
- [34] A. P. Pentland. Linear shape from shading. *International Journal of Computer Vision (IJCV)*, 4(2):153–162, 1990.
- [35] Chief Architect Software. Chief architect home designer. <http://www.homedesignersoftware.com/homedesign/>, 2012.
- [36] I. Suveg and G. Vosselman. 3d building reconstruction by map based generation and evaluation of hypotheses. In *British Machine Vision Conference (BMVC)*, 2001.
- [37] A. Telea. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools (JGT)*, 9(1):23–34, 2004.
- [38] R. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision, ICCV '98*, page 839, Washington, DC, USA, 1998. IEEE Computer Society.

- [39] P. S. Tsai and M. Shah. Shape from shading using linear approximation. Technical report, 1992.
- [40] P. S. Tsai and M. Shah. Shape from shading with variable albedo. *Optical Engineering*, 37(4):1212–1220, 1998.
- [41] S. H. Westin. A comparison of four brdf models. Technical report, Cornell University, 2004.
- [42] A. Wiechert and M. Gruber. Aerial perspective: Photogrammetry versus lidar. *Professional Surveyor Magazine*, 29(8):1–3, 2009.
- [43] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.
- [44] J. Zhang, F. Mai, Y. S. Hung, and G. Chesi. 3d model reconstruction from turntable sequence with multiple -view triangulation. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*, ISVC '09, pages 470–479, Berlin, Heidelberg, 2009.
- [45] R. Zhang, P. Tsai, J. E. Cryer, and M. Shah. Commercial satellite imagery comes of age. *Issues in Science and Technology*, 16(1), 1999.
- [46] R. Zhang, P. Tsai, J. E. Cryer, and M. Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.