

Revisiting Normalized Cross-Correlation for Accurate Camera Pose Estimation and Accurate Real-Time Multiple View Stereo

by

Eduardo de Brito Almeida

B. S., Universidade Federal do Ceará, 2004

Sc. M. in Engineering, Brown University, 2010

Sc. M. in Applied Mathematics, Brown University, 2010

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in the
School of Engineering at Brown University

Providence, Rhode Island

May 2015

© Copyright 2015 by Eduardo de Brito Almeida

This dissertation by Eduardo de Brito Almeida is accepted in its present form by
the School of Engineering as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____

David B. Cooper, Advisor
School of Engineering

Recommended to the Graduate Council

Date _____

Joseph L. Mundy, Reader

Date _____

Pedro F. Felzenszwalb, Reader

Approved by the Graduate Council

Date _____

Peter M. Weber
Dean of the Graduate School

Curriculum Vitae

Eduardo de Brito Almeida was born in Stony Brook, New York on July 12th, 1981. He graduated from Federal University of Ceará in Fortaleza, Ceará, Brazil in December 2004 with a Bachelor of Science degree in Electrical Engineering. In the Summer of 2006, Eduardo began his graduate studies at Brown University in Providence, Rhode Island. Since then he has been at the Laboratory for Engineering Man-Machine Systems (LEMS) at Brown University working as a research assistant in the field of computer vision under the supervision of Prof. David B. Cooper. Eduardo earned Master of Science degrees in Engineering and in Applied Mathematics from Brown University in 2010. He is a NASA Rhode Island Space Grant Fellow and a NASA Harriett G. Jenkins Pre-Doctoral Fellow. In 2009, 2010 and 2012, Eduardo interned at NASA's Jet Propulsion Laboratory (JPL), a division of California Institute of Technology located in Pasadena, California, where he worked on projects related to image-based 3-d reconstruction under the direction of Dr. Curtis W. Padgett. He received a Space Act Award from NASA Inventions and Contributions Board in recognition of his development of significant scientific contribution that has been determined to be of value in the conduct of aeronautical activities of NASA. His research interests include computer vision, pattern recognition, applied probability and statistics, change detection, image-based 3D reconstruction, multiple view geometry, structure from motion and GPU acceleration.

Acknowledgements

I owe many people a debt of gratitude for supporting me and making this dissertation possible. In particular, I would like to express my sincere appreciation to my advisor Prof. David B. Cooper for the essential guidance, advices and support he gave me during the time I was working on this thesis and throughout all of my graduate studies. His great experience has always positively influenced my professional career in countless aspects. He has a great sense of humor and he also has been a wonderful friend, who, for instance, has introduced me to the excitement of sailing out of Boston harbor.

I am deeply thankful to Joseph L. Mundy and Pedro F. Felzenszwalb for serving in my thesis committee and providing many insightful comments and suggestions in this work. In addition, I am also very honored to have interacted with Prof. Gabriel Taubin, Prof. Benjamin Kimia and other professors at Brown University who have taught me and guided my academic work.

Particular thanks goes to professors Fátima Sombra, Basilis Gidas and Rui Seara whose encouragement inspired me to go after my dream of pursuing a doctoral degree in the United States.

This dissertation is a culmination of several years of research sponsored by Brown University and its School of Engineering; by the Mobility and Robotics Section at the Jet Propulsion Laboratory (JPL), California Institute of Technology; by the National Aeronautics and Space Administration (NASA); by the NASA Harriett G. Jenkins Pre-doctoral Fellowship Program (JFPF); by the Rhode Island Space Grant Consortium (RISG); and by the student programs from JPL Education Office.

I gratefully acknowledge my JPL mentor Dr. Curtis W. Padgett, the RISG director Dr. Peter H. Schultz, and all JFPF program members. Their invaluable support in awarding me a total of two NASA fellowships and three exciting internships have greatly enhanced my academic experience. I

have tremendously benefited by collaborating with Dr. Padgett and his group at JPL, where the ideas for the foundations of this thesis were consolidated.

There are several friends who I would like to thank. Gilson Lima, John Raiti, Brandon Mayer, Ricardo Fabbri, Kilho Son, Isabel Restrepo and David Borton for the amazing friendship and all the countless advices. Ming-Ching, Nhon, Osman, Vishal, Fatih, Amir, Anil, Shubao, Maruthi, Daniel Crispell, Ozge, Daniel Moreno, Matthew, Krishna, Ibrahim and all other labmates for their sympathy and for providing a pleasant work environment at LEMS.

I would like to give *very special* thanks to my father, my mother, my brother and my sister-in-law, for their constant support and endless love that always encouraged me to reach my goals and pursue my graduate school ambitions. This thesis is dedicated to them. I would also like to thank all my relatives for always being so kind and supportive. Their presence is constantly felt despite the physical distance. I am also grateful to Juliana, Carol, Vanessa, Kárida, Sílvia and Wallace Casaca for the memorable companionship during my last years at Brown, and Fr. Bodah for the friendship.

Above all else, I acknowledge the blessings of God, to whom I owe everything.

Contents

List of Tables	xiii
List of Figures	xv
1 Introduction	1
1.1 Goals and requirements	8
1.2 Contributions	10
1.3 Outline	11
2 Related work	13
2.1 Feature detection and matching	13
2.2 Dense reconstruction	18
2.2.1 Real-time 3-d reconstruction	20
2.3 Disambiguation of NCC correspondences	24
2.3.1 Disambiguating dense repeating features	25
3 Review of basic algorithms	31
3.1 Rectification	31
3.2 Weighted normalized cross-correlation	34
3.2.1 Size relations	36
3.2.2 Why is weighting important?	36
3.2.3 Why is normalization important?	37

3.3	Delaunay triangulation	38
3.4	Camera estimation	40
3.4.1	Bundle adjustment	40
3.4.2	Reprojection error	41
4	Near Real-time Camera Pose Estimation	43
4.1	Matching overview	44
4.1.1	Displaying image matches	45
4.2	Interest points	50
4.3	Matching process	55
4.3.1	Correlation peaks	57
4.3.2	Weak corner removal	60
4.3.3	Collection of sets of match candidates	61
4.3.4	Initial low-resolution matching	62
4.3.5	Native resolution matching	64
4.4	Disambiguation process	64
4.4.1	Overview	64
4.4.2	Fundamental matrix estimation	66
4.4.3	Ratio-inequalities test	67
4.4.4	Highest scores ratio module	68
4.4.5	Epipolar constraint module	68
4.4.6	Epipolar sectors module	68
4.4.7	Multi-view disambiguation modules	70
4.4.8	Indirect matches	71
4.4.9	Affine transfer modules	72
4.4.10	Match topology module	74
4.4.11	Epipolar transfer module	75
4.5	Conclusion	79

5	Planar grid matching	81
5.1	Overview	83
5.1.1	Motivation for modeling planar surfaces	87
5.2	Grid points detection	90
5.2.1	Finding grid seeds	90
5.2.2	Augmented clones set	92
5.2.3	Lattice model	93
5.2.4	Iterative outlier removal	97
5.2.5	Defining grid main directions	100
5.3	Grid lattice estimation	100
5.3.1	Connecting vertices into line segments	102
5.3.2	Defining line segments neighborhood	103
5.3.3	Merging line segments	103
5.3.4	Organizing lines into a 2-d lattice	106
5.4	Grid matching	106
5.4.1	Selection of grid points to match	106
5.4.2	Match grid points independently	109
5.4.3	Reduce ambiguities by matching grid lines	109
5.4.4	Topology-preserving incremental line matching	116
5.5	Wide-baseline matching via lattice tracking	121
5.5.1	Tracking lattice planes	122
5.6	Conclusion	125
6	Real-time Depth Estimation	126
6.1	Algorithm description	127
6.2	The ray grid	127
6.3	Rectification	128
6.4	Multiple view stereo matching	129
6.4.1	Similarity score along a ray	129

6.4.2	Peak representation	132
6.4.3	Score mapping from 2-d to 3-d	133
6.4.4	Background rays	133
6.4.5	Score fusion	133
6.4.6	Depth estimation	134
6.4.7	Peak refinement	137
6.4.8	Parameters	137
6.5	Contributions	138
6.6	Volume far sides	138
6.6.1	Far side of a voxel grid	141
6.6.2	Far side of a rectangular volume	141
6.6.3	Far side visibility	142
6.7	Surface reconstruction	145
6.7.1	Energy minimization using graph cuts	145
6.7.2	Graph construction	147
6.7.3	Contributions	148
7	GPU Accelerated Template Matching	151
7.1	General Purpose Graphics Hardware	152
7.1.1	Typical GPU Architecture	152
7.1.2	GPU memory model and cache	154
7.1.3	Parallel synchronization	157
7.1.4	Data-parallel instructions	159
7.1.5	Memory latency and performance	160
7.1.6	Active threads, warps and blocks.	161
7.1.7	Register spills	161
7.1.8	Parallel functions	162
7.1.9	Summary of performance requirements	162
7.2	Normalized cross-correlation	163

7.2.1	Expressing NCC in terms of sums	163
7.2.2	Sequential NCC algorithm complexity	164
7.2.3	Optimized CPU implementation	167
7.2.4	Disregarding integral images for parallel NCC	167
7.3	Proposed parallel NCC	168
7.3.1	Distribution of work	169
7.3.2	Memory allocations	171
7.3.3	Alternative parallel NCC implementations	172
7.4	Feature matching application	175
7.4.1	Overview of GPU function	175
7.4.2	Handling memory allocations	176
7.4.3	Finding correlation local maxima	176
7.4.4	Finding the highest peak	177
7.4.5	Prunning the peaks	177
7.4.6	Refining peak locations	177
7.4.7	Storing the correlation peaks	177
7.5	Multiple view stereo application	178
7.5.1	Image rectification	178
7.5.2	Weighted Normalized cross-correlation	178
7.5.3	Peak detection	179
7.5.4	Evaluating scores at each depth	179
7.5.5	Score fusion	180
7.5.6	Depth refinement	180
8	Experiments and Evaluation	181
8.1	Parallel NCC Performance	181
8.1.1	Alternative methods	182
8.1.2	Devices used in the experiments	183
8.1.3	Runtime of NCC on designated devices	183

8.1.4	Discussion	196
8.1.5	Comparison to additional NCC methods	197
8.2	NCC for camera pose estimation	198
8.3	Planar lattice matching and tracking	204
8.3.1	Results	204
8.4	Dense surface estimation	212
8.4.1	Evaluation datasets.	212
8.4.2	Quantitative evaluation.	213
8.4.3	Qualitative evaluation	222
8.5	Conclusions	226
9	Conclusions and future work	233
	Bibliography	236

List of Tables

4.1	Notation for proposed feature matching pipeline.	49
4.2	List of proposed feature matching pipeline parameters.	49
6.1	Notation for proposed depth estimation method.	130
6.2	List of depth estimation parameters used by proposed method.	137
7.1	Comparison of different GPU implementations of normalized cross-correlation.	174
8.1	Running times (ms) taken from figure 8.1.	184
8.2	Running times (ms) taken from figure 8.2.	185
8.3	Running times (ms) taken from figure 8.3.	186
8.4	Running times (ms) taken from figure 8.4.	187
8.5	Running times (ms) taken from figure 8.5.	188
8.6	Running times (ms) taken from figure 8.6.	189
8.7	Running times (ms) taken from figure 8.7.	190
8.8	Speedup ratio values taken from figure 8.8.	191
8.9	Speedup ratio values taken from figure 8.9.	192
8.10	Speedup ratio values taken from figure 8.10.	193
8.11	Speedup ratio values taken from figure 8.11.	194
8.12	Speedup ratio values taken from figure 8.12.	195
8.13	Mean squared reprojection error of SIFT matching.	198
8.14	Mean squared reprojection error of proposed feature matching method (chapter 4).	198

8.15 Mean squared reprojection error (MSE) for “capitol26” scene.	210
8.16 Mean squared reprojection error (MSE) for “downtown46” scene.	210
8.17 Statistics for proposed multiple view stereo pipeline.	221
8.18 Statistics of proposed surface extraction pipeline.	222
8.19 Accuracy results obtained from the evaluation of Seitz <i>et al.</i>	228

List of Figures

1.1	Image-based reconstruction pipeline.	2
1.2	A correspondence between two views of an object and the geometry of the associated observed surface point.	3
1.3	Illustration of a few corresponding feature locations that are connected by green lines.	3
1.4	Dense 3-d point cloud extracted from images of the scene shown in figure 1.3 via proposed dense matching method.	4
1.5	Illustration of an triangle mesh representing a building estimated via proposed approach.	5
1.6	Results of the proposed wide-baseline matching of dense repetitive features for building facades.	7
2.1	Feature detection and matching pipeline.	14
2.2	Interest points detected as SIFT features.	15
2.3	Performance of feature detection and matching for viewpoint change.	16
2.4	Dense stereo reconstruction pipeline.	17
2.5	Image rendering from 3-d models learned from multiple aerial images of an urban scene.	18
2.6	The two scenes from the multi-view stereo evaluation of Seitz <i>et al.</i>	19
2.7	Images taken from the twelve scenes of the multi-view stereo evaluation dataset of Strecha <i>et al.</i>	20
2.8	Images from some of the 80 scenes of the evaluation dataset of Jensen <i>et al.</i>	21
2.9	Large-scale reconstruction from Vu <i>et al.</i>	22

2.10	Facades of large buildings, which normally present piecewise planar grids of dense repeating features (windows).	26
2.11	Typical example of the model-based pairwise matching method of this thesis for dense repeating features.	26
2.12	Scene with replicated structures, which typically causes confusion in matching algorithms.	28
2.13	Automatic detection of deformed lattices in a single image.	29
3.1	Side by side rectified views.	31
3.2	Geometric relations between matching points on a stereo pair.	32
3.3	Illustration of undesired distortions due to rectification of an image pair where an epipole is located in the image plane.	34
3.4	Image matching via normalized cross-correlation.	35
3.5	A Delaunay triangulation and its corresponding Voronoi diagram.	39
4.1	Feature matching process and the scope of images taken from a sliding window moving through adjacent frames of a sequence.	45
4.2	Matching pipeline for an image stream.	46
4.3	Matching process for an image pair.	47
4.4	Disambiguation process block from figure 4.3.	48
4.5	Typical example of the pairwise matching method.	51
4.6	Detail of the same example as figure 4.5.	52
4.7	Edge responses on two aerial images of urban scenes.	54
4.8	Interest point detection.	56
4.9	Distributions of the correlation scores of all estimated matches of an aerial urban scene.	58
4.10	Domains of correlation windows.	59
4.11	Illustration of normalized cross-correlation peaks at homogeneous locations.	61
4.12	Comparison of spatial distribution and density of normalized cross-correlation function peaks.	63
4.13	Elimination of outlier matches from the epipolar sectors module.	69
4.14	Illustration of matches $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ from I_R to I_T and $\mathbf{x}_T \leftrightarrow \mathbf{x}_A$ from I_T to I_A	72

4.15	Illustration of match topology module disambiguation steps operating in two views.	76
5.1	Examples of facades of large buildings.	82
5.2	Matching pipeline for dense repetitive features.	83
5.3	Definition of planar lattice.	84
5.4	Typical example of the proposed model-based pairwise matching method.	85
5.5	Overview of the planar grid matching pipeline.	86
5.6	Multiple estimated lattices of a reference image.	88
5.7	Multiple lattice correspondences found in a single image pair.	89
5.8	Matching results for highly repetitive urban scene features.	91
5.9	Illustration of two grids with grid seeds and clone sets.	92
5.10	Example of Delaunay triangulation of grid features.	94
5.11	Example of common triangulation edges.	94
5.12	Illustration of feature vectors of Delaunay triangulation edges.	95
5.13	Distribution of the number of neighbors of a vertex of a Delaunay triangulation of a typical facade.	97
5.14	Iterative detection and removal of non-lattice points.	101
5.15	Examples of Delaunay triangulations computed from pruned estimated grid points.	102
5.16	Line segments estimated for distinct grids of windows of two buildings.	104
5.17	Illustration of parallel neighbors of a line segment.	105
5.18	Grid lines detection from merging line segments.	107
5.19	Examples of estimated 2-d planar grid lattices.	108
5.20	Histograms of the number of correlation matches of repeating grid points.	109
5.21	Examples of grid point matches computed individually.	110
5.22	Illustration of grid line matching.	112
5.23	A grid seen from an oblique viewing angle.	115
5.24	Erroneous corresponding lattices that would arise if neighbor point spacing scale was allowed to change drastically.	116

5.25	The solution of the proposed lattice correspondence problem is not unique under a degenerate configuration where an epipole and a lattice vanishing point coincide.	119
5.26	Examples of pairwise matches of building facade grid features seen from distinct angles.	120
5.27	A lattice connectivity estimated in an image (right) is transferred to another image (left).	121
5.28	Illustration of facade registration.	122
5.29	Result of the proposed wide-baseline matching of dense repetitive features.	123
5.30	Example very wide-baseline tracking of a planar facade.	124
6.1	Estimated depths computed in 1.1 seconds.	126
6.2	Comparison of normalized cross-correlation (NCC) with Gaussian-weighted normalized cross-correlation (WNCC).	131
6.3	Results of fusion methods for combining similarity scores along rays from multiple viewpoints.	135
6.4	Estimated depths from a given reference viewpoint of the “capitol” dataset.	136
6.5	Two-dimensional illustration of the <i>far side</i> of two convex shapes.	139
6.6	Two-dimensional representation of the <i>far side</i>	140
6.7	Two-dimensional representation of the <i>far side</i> of an axis-aligned rectangular grid.	141
6.8	Three-dimensional representation of the <i>far side</i> of a rectangular bounding box.	144
6.9	Illustration of a cut in a graph with six vertices and two special terminals.	146
7.1	GPU memory spaces.	155
7.2	Simple examples of global memory access patterns.	157
7.3	GPU memory data flow.	158
7.4	SIMD (Single Instruction, Multiple Data) multi-core computer architecture.	159
7.5	Schematic representation of the execution of warps on a single thread block.	160
7.6	Ratio of the algorithm complexities for computing the numerator of NCC.	166
7.7	Overview of proposed parallel NCC implementation.	168
7.8	An operation of the outermost loop of the proposed GPU implementation.	169
7.9	Operations of the intermediate loop of the proposed GPU implementation.	170
7.10	Operations of the sequential innermost loop of the proposed GPU implementation.	171

8.1	Runtime (ms) of frequency domain NCC implementation of OpenCV running on the “i7-950” CPU for $N \times N$ templates and $M \times M$ images.	184
8.2	Runtime (ms) of spatial domain NCC implementation of JPL running on the “i7-950” CPU for $N \times N$ templates and $M \times M$ images.	185
8.3	Runtime (ms) of hybrid domain NCC implementation of Matlab running on the “i7-950” CPU for $N \times N$ templates and $M \times M$ images.	186
8.4	Runtime (ms) of spatial domain NCC implementation of OpenCV running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images.	187
8.5	Runtime (ms) of frequency domain NCC implementation of OpenCV running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images.	188
8.6	Runtime (ms) of proposed spatial domain NCC implementation running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images.	189
8.7	Runtime (ms) of proposed spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images.	190
8.8	Speedup of proposed spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images over frequency domain NCC implementation of OpenCV running on the “i7-950” CPU.	191
8.9	Speedup of proposed spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images over spatial domain NCC implementation of JPL running on the “i7-950” CPU.	192
8.10	Speedup of proposed spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images over hybrid domain NCC implementation of Matlab running on the “i7-950” CPU.	193
8.11	Speedup of proposed spatial domain NCC implementation running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images over spatial domain NCC implementation of OpenCV running on the “GTX 275” GPU.	194

8.12	Speedup of proposed spatial domain NCC implementation running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images over frequency domain NCC implementation of OpenCV running on the “GTX 275” GPU.	195
8.13	Structure from motion reconstruction of the “capitol26” dataset.	199
8.14	Structure from motion reconstruction of the “horse21” dataset.	200
8.15	Structure from motion reconstruction of the “campus29” dataset.	201
8.16	Typical example of pairwise matching via the proposed NCC method.	202
8.17	Typical example of pairwise matching via traditional SIFT method.	203
8.18	Images of the “downtown46” dataset.	204
8.19	Results of proposed matching of dense repetitive features for adjacent frames.	205
8.20	Results of the proposed wide-baseline matching of dense repetitive features.	206
8.21	Additional results of the proposed wide-baseline matching of dense repetitive features.	207
8.22	Multiple estimated lattices on a single image.	208
8.23	Several multiple-lattice correspondences from different angles.	209
8.24	Comparison of 3-d points returned by bundle adjustment on the “downtown46” scene.	211
8.25	Datasets used for reconstruction evaluation.	214
8.26	Camera positions for multiple view stereo reconstruction evaluation of Jensen <i>et al.</i>	215
8.27	Highly dense ground truth point cloud of the “tools49” scene.	216
8.28	Images of the “smurf63” and “statuette64” datasets.	217
8.29	Reconstruction evaluations for scenes with well textured objects.	218
8.30	Reconstruction evaluations for scenes containing textureless or highly specular objects.	219
8.31	Overall reconstruction performance comparisons.	220
8.32	Surface reconstruction for thin geometry.	223
8.33	Performance evaluation comparison of the proposed MVS algorithm paired with the proposed surface estimation method and paired with Poisson reconstruction.	224
8.34	Robustness of proposed method to varying illumination.	225
8.35	Reconstruction of objects from datasets used in evaluations in section 8.4.2.	227
8.36	Running times of several multiple view reconstruction algorithms.	228

8.37 Surfaces estimated by proposed meshing algorithm running on aerial views.	229
8.38 Reconstruction of an indoor environment where radiometric variations are noticeable.	230
8.39 Reconstruction of thin geometry.	231
8.40 Example of failure regions on the “downtown46” dataset.	232

Chapter 1

Introduction

Reconstructing three-dimensional shapes from images is important in the field of computer vision and a variety of applications, such as augmented reality, entertainment industry, robotics, industrial inspection, digital archival and aerial cartography. Estimating 3-d models from images can be performed via stereo vision. The term *stereo* here is used as a short for stereopsis, the perception of depth and 3-d structure from observing a scene from different viewpoints. Stereo most often refers to binocular vision, *i.e.*, two eyes or two views, while multiple view stereo refers to the use of three or more images.

In binocular vision, objects at distinct depths project into two views with different relative positions, called disparities, which provide depth perception since objects that are closer present larger disparities than the ones that are farther. In order to process disparities, the correspondence between image locations must be known. Thus, estimating 3-d models from images has image correspondence and feature matching as a building block.

Features are points of interest in an image that differ in appearance from their local neighborhood. Ideally, a feature would be as a geometric point, *i.e.* have a location but no extent, yet in practice this is relaxed since images are discretized into pixels and a single pixel is a poor representation for a feature. Thus, typical features are small pieces of information in an image, *e.g.* a corner, an edge or a blob. Feature matching is the process of determining corresponding feature locations in images taken under different viewing conditions (see figure 1.2). In order to automatically estimate an underlying



Figure 1.1: Image-based reconstruction pipeline.

rigid 3-d model that was perceived through a number of images, there are three basic steps:

- Step 1:** Estimate sparse feature correspondences between images as illustrated in figure 1.3, which allows recovery of camera parameters for the cameras that acquired the images. Camera parameters include camera relative motion (location and orientation) and optical characteristics of the lens and imaging device.
- Step 2:** Estimate dense correspondences for the observed surface given camera geometry. Dense matching returns a set of 3-d points, denoted a *point cloud*, extracted from multiple overlapping image pairs taken around an object. The estimated points are dense samples of the observed surface of the scene, as in figure 1.4.
- Step 3:** Fit a surface to the dense unstructured point cloud in order to recover a complete representation of the true surface of the scene via a process called *meshing*, which represents surfaces as a polygon mesh (see figure 1.5).

It is possible to bypass the dense point representation and estimate surface directly using methods that implicitly aggregate estimated surface location data into a volumetric representation, or methods that assume known scene topology and possibly some other prior information. However, it is often possible to isolate the point estimation stage of a method from the surface extraction stage yielding the three aforementioned steps, which are illustrated in figure 1.1. Note, camera geometry estimated in step 1 is achieved via *structure from motion* (SFM), the problem of simultaneously estimating camera relative motion and scene structure (3-d points) from 2-d image correspondences. SFM normally includes a global refinement called *bundle adjustment* that simultaneously refines all scene geometry and camera parameters.

This thesis tackles the 3-d image modeling problem and divides the solution into the three aforementioned steps consisting, respectively, of camera estimation (step 1), dense stereo matching (step 2) and surface extraction (step 3), which are performed in this order. The problems arising here

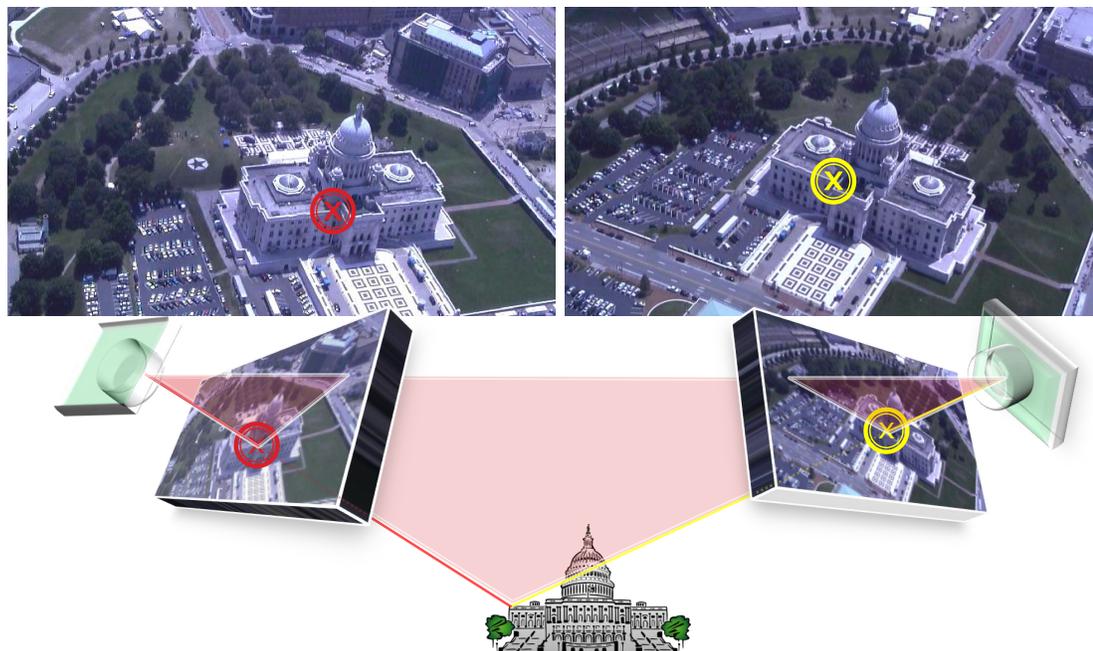


Figure 1.2: A correspondence between two views of an object and the geometry of the associated observed surface point. *Left*: pixel in a reference image. *Right*: matching location in another image. *Bottom*: illustration of a 3-d scene depicting camera locations, corresponding viewing rays, the associated observed 3-d point and its depths from each camera location.



Figure 1.3: Illustration of a few corresponding feature locations that are connected by green lines. The images are aerial views from a helicopter.

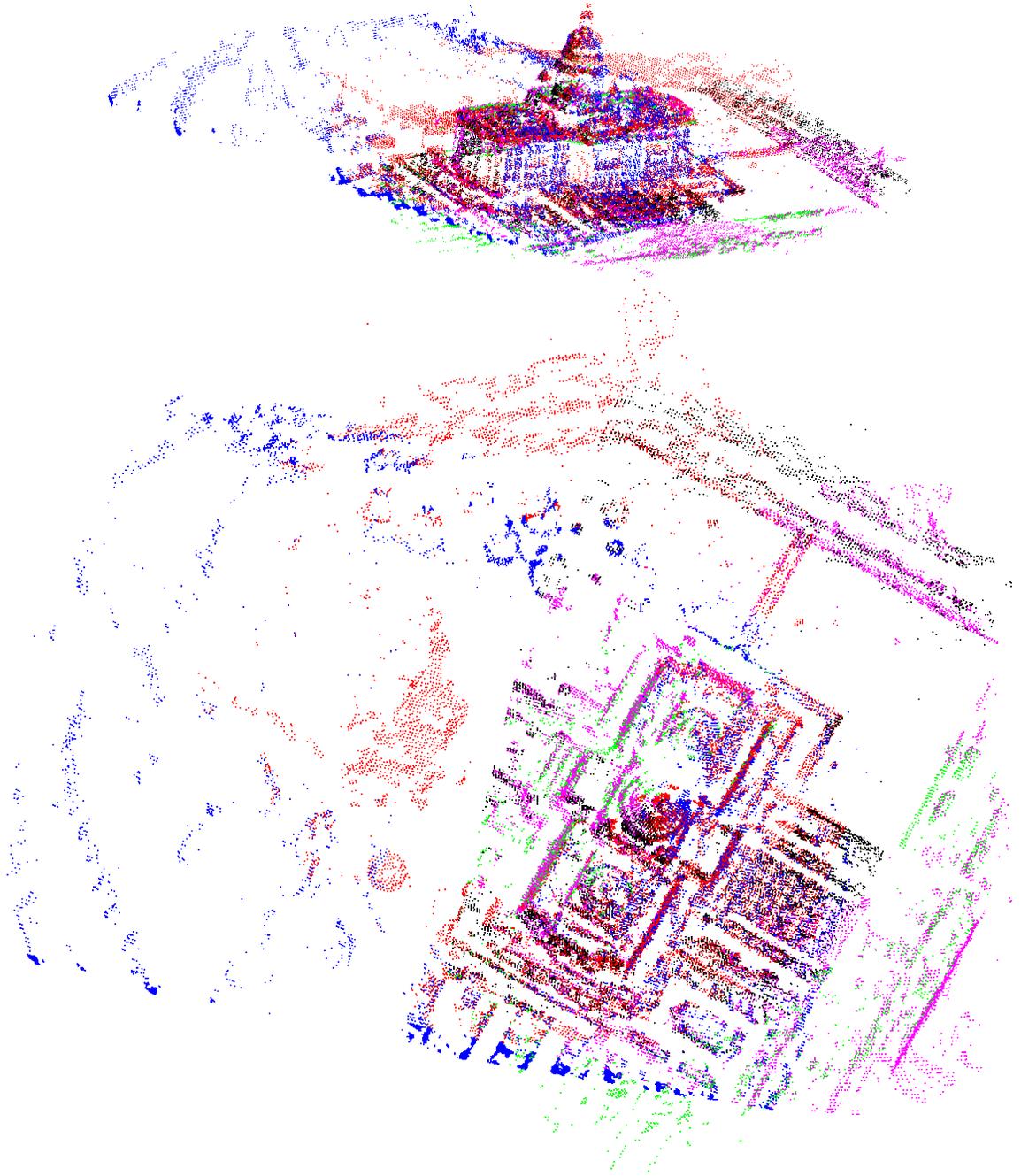


Figure 1.4: Dense 3-d point cloud extracted from images of the scene shown in figure 1.3 via proposed dense matching method. *Top*: a bird's-eye view of the point cloud. *Bottom*: a nadir view. In order to model the entire object it is observed from multiple angles. Each color represents points estimated from a given reference viewing location, *i.e.* from a depth map.

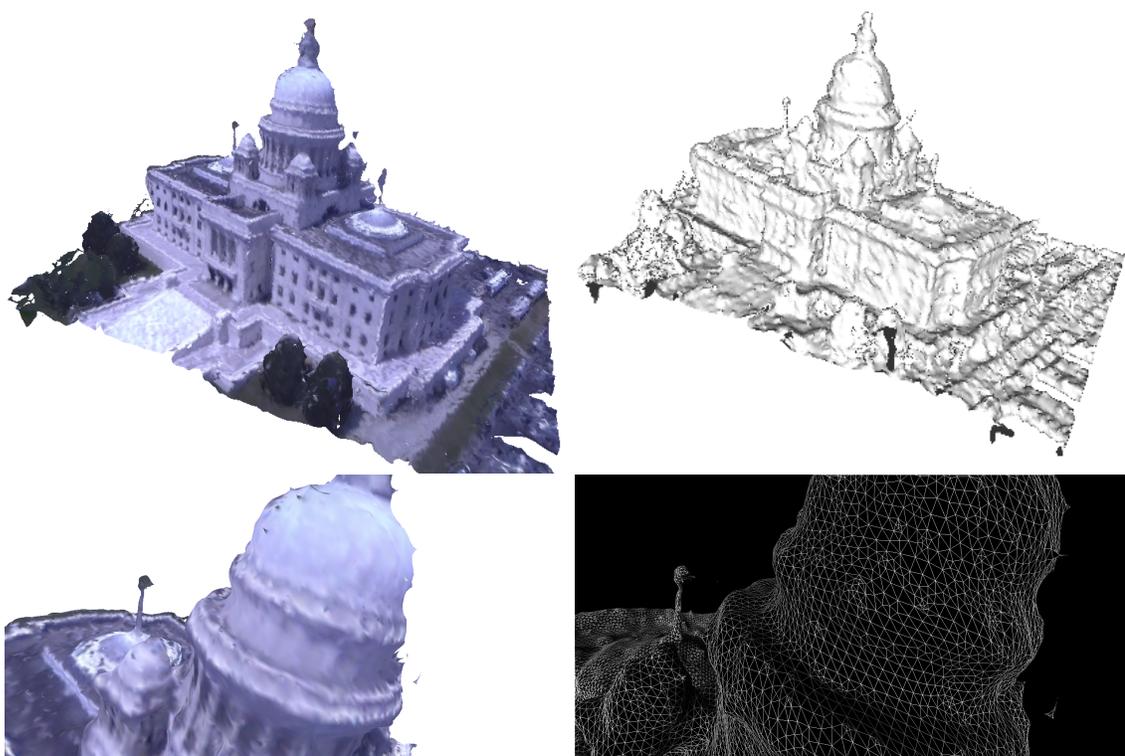


Figure 1.5: Illustration of a triangle mesh representing a building estimated via proposed approach. *Top left*: the mesh with colored faces. *Top right*: the geometry of the untextured mesh illuminated with artificial light. *Bottom left*: detail view of the mesh with colored faces. *Bottom right*: detail view of the mesh showing the faces, edges and vertices.

are far from being entirely solved since many factors introduce errors and degrade correspondence performance, as radiometric conditions, specularities, occlusion, perspective distortions and ambiguities. Illumination color, illumination direction and imaging devices are radiometric variations that affect the appearance of an object across images. Specularity effects cause surfaces to reflect more light in some directions than in others causing serious matching difficulties. In order to handle occlusion, a method must be robust to missing matches, presenting no correspondences when none is in fact visible. Another problem is ambiguities from duplicated structures requiring special disambiguation since more than one correspondence exist for such structures based solely on local appearance. A window array in the facade of a large building is an example of such duplicated structures. Regions with nearly constant appearance, also known as textureless regions, *e.g.* the sky, are an additional source of ambiguity. Matching textureless regions is much harder than when using their textured

counterparts, analogous to the difficulties found in solving jigsaw puzzles for pieces having no texture variation versus other puzzle pieces.

This thesis revisits the use of *normalized cross-correlation* as a similarity measure to be used for image matching in step 1 and step 2. Traditionally, the correspondence problem of camera estimation in step 1 has been broadly tackled using other robust feature descriptors, especially the widely popular scale-invariant feature transform (SIFT) descriptor [74]. However, it is shown that normalized cross-correlation estimates matching features with higher location accuracy than SIFT. Moreover, it is shown that the computationally more expensive computations of normalized cross-correlation for correspondence search can be completely mitigated by parallel implementations on commodity graphics hardware. This thesis proposes a high-performance correlation search parallel implementation that is used both in steps 1 and 2 for real-time computation of putative matches for many thousands of features per image pair of a dataset.

For step 1, the proposed real-time correlation matching is followed by an additional proposed serial disambiguation module that will be parallelized in future work. The disambiguation allows finding a large number of matches that, in general, surpass the number of matches found via SIFT. For step 2, an entire pipeline is implemented in parallel to retrieve 3-d point clouds from images in real-time via multiple view stereopsis (MVS). The MVS pipeline uses the same proposed parallel normalized cross-correlation implementation for matching, which is also used in step 1. For step 3, one may use any meshing technique available. Common ones are based on level sets, graph cuts optimization or signed distance functions. Meshing results are demonstrated using a proposed graph cuts approach that produces a watertight polygon mesh and works on uncontrolled environments. A mesh here is a set of vertices, edges and triangular faces defining a polyhedral solid that represents an approximation of a surface (see figure 1.5).

The proposed methods are experimentally validated on several aerial scene datasets. Aerial scenes are one of the most challenging datasets categories since virtually no conditions can be controlled, as opposed to datasets acquired in laboratory, where illumination, object materials, camera motion, camera stability and background may be fixed and chosen to support an algorithm. Results on indoor and controlled environments are also shown.

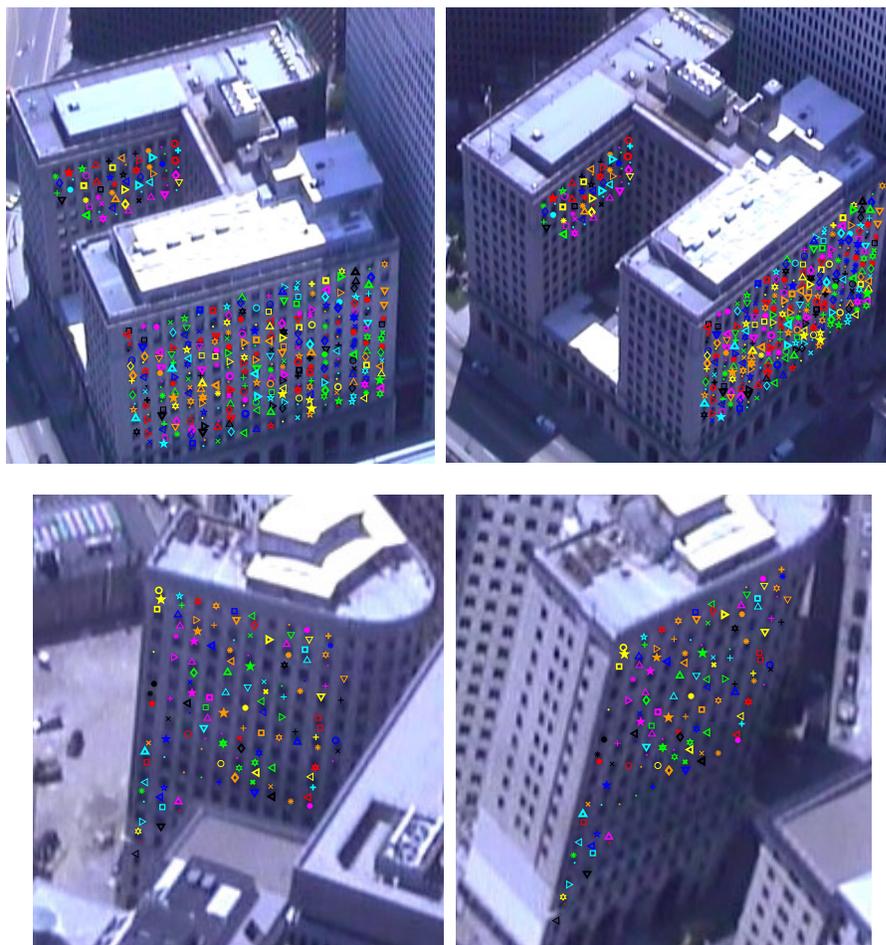


Figure 1.6: Results of the proposed wide-baseline matching of dense repetitive features for building facades. The features are expected to be on a plane and distributed on a regular grid. Correspondences are shown with the same marker. The top row shows robustness to occlusion and to multiple planar surfaces. The bottom row illustrates that the matching is also robust to missing grid matches.

Urban and architectural scenes are common in aerial views used for building 3-d maps of cities. Such scenes normally present dense feature repetition that often causes issues in matching algorithms, therefore it is a relevant topic in computer vision. This work also presents a novel solution to the very ambiguous problem of matching dense periodically repeating features found in such scenes, as windows on facades of large buildings (see images in figure 1.6 for examples). The matching of such features are hard to disambiguate due to their density and periodicity. The algorithm uses a plausible assumption that these repeating features are distributed in a regular grid on a planar surface that allows match disambiguation.

The proposed method for matching repeating features also exploits detected planar geometry for achieving wide-baseline matching, an important topic in 3-d reconstruction related to accuracy, and results are illustrated in figure 1.6. Baseline is the distance between two viewing locations. Short-baseline matches allow recovery of scene geometry usually subject to larger errors along the depth direction, while wide-baseline matches suffer from location bias due to the drastic change in appearance that affect local feature descriptors. Clearly there is a trade-off. Figure 1.2 illustrates the geometry of a wide-baseline match. The errors in short-baseline matching arise from the nearly degenerate geometry extracted from the image pairs. In a short-baseline example, the two viewing rays seen in figure 1.2 would be nearly parallel and their intersection (the reconstructed 3-d point) would be very sensitive to small errors.

Wide-baseline matches do not suffer from degeneracy problems, however it is harder to obtain them in general given the considerable appearance changes of feature points under a wide viewpoint change. The proposed solution for matching repeating features of planar facades is then extended for attaining wide-baseline matching, thus improving geometry estimation accuracy in urban scenes. In summary, this solution handles a problem where traditional local matching methods fail due to extreme ambiguity and can be used to correspond the planar facades from arbitrary viewpoints as long as they are visible (under ultra wide baseline changes). The proposed solution also uses normalized cross-correlation and achieves far superior geometry estimation accuracy when compared to short-baseline counterparts. A second benefit of detecting planar facades is a first step in the geometric interpretation of a 3-d scene.

1.1 Goals and requirements

This thesis revisits normalized cross-correlation to show that it is one of the most accurate feature descriptors for the purpose of camera estimation and proposes a massively parallel implementation of normalized cross-correlation to show feasibility of its use in the fundamental problem of real-time 3-d reconstruction from 2-d images. This is a core problem in computer vision and consists of estimating the elements that have generated the input images. The solution is geared towards very general and uncontrolled environments. The focus is on outdoor scenes such as ground level imagery from

moving hand-held camera or aerial imagery acquired from aircraft or satellite. The images may be taken from a video sequence or may be an unorganized image collection obtained from the Internet. The proposed algorithms operate on grayscale images but can be extended to use color.

Video sequences typically taken at 24 frames per second or higher rates provide small frame to frame motion and high temporal continuity. When compared to still images, video has lower quality in terms of image detail, due to lower resolution, motion blur and compression artifacts, however, high frame rate video provides a much larger amount of data, and, correspondence search between adjacent frames of such videos can be performed in small windows. In contrast, still images present larger disparities for correspondences between adjacent image pairs, which improves the overall accuracy of geometry estimation. One of the major challenges of still images are occlusions that will happen more often than in video due to the larger viewpoint change.

In this thesis, focus is given to high-resolution images for maximizing geometry estimation accuracy. A requirement of the proposed algorithms is that any two frames are connected via a sequence of small-baseline frames, otherwise they may not be automatically registered to each other in a common reference frame. This is a fair assumption used by most, if not all, general 3-d reconstruction techniques in order to perform reliable feature matching to register images to each other. Therefore, images are expected to be acquired at low frame rates from a smoothly moving camera, preferably of high resolution. Note, the proposed methods have no high frame rate continuity assumptions so images may be sampled to have a significant baseline change if taken from high frame rate video.

The current implementation of feature matching for step 1 does not handle a large 2-d image-rotation of adjacent pairs since normalized cross-correlation is not rotation-invariant. Noticeable 2-d rotations may arise in short-baseline pairs if a camera is significantly rotated about the viewing axis, whose direction is perpendicular to the image sensor. This short limitation can be easily fixed as future work by preprocessing an image pair to estimate and compensate for the relative image rotation by matching rotation-invariant features descriptors, *e.g.* SIFT or others. The proposed solution for multiple view stereo, step 2, does not have such limitation since an image rotation is compensated as a by-product of an algorithm simplification transformation called rectification used in step 2. Rectification of an image pair requires estimated matches or relative cameras and is discussed

in detail in chapter 3.

The proposed algorithms work under fixed or varying illumination conditions from frame to frame since normalized cross-correlation is invariant to linear changes in lighting. In addition one of the most important goals is to be able to run in real-time in significantly large images. Such data is commonly found in aerial views from aircraft and is useful for building detailed 3-d maps of ground terrain, real-time surveillance, and accurate unmanned aerial vehicle (UAV) navigation. Although UAVs can navigate using Global Positioning System (GPS) measurements, this navigation may be inaccurate, may fail on GPS-denied areas or fail due to faulty GPS, therefore real-time visual odometry navigation is important.

It must be stressed that real-time application implementations running on large images is a challenging problem and therefore many available state-of-the-art solutions do not apply since they are developed to run in an offline fashion and have slow performance. The rather recent availability of cheap parallel processing power on general purpose graphics hardware allowed the development of real-time solutions for the problems of interest given appropriate parallel algorithms, which are used in this thesis to accelerate program runtime. Processing large images is a challenge not only due to the amount of data to process, but also since they occupy precious space in the limited memory of current graphics hardware.

1.2 Contributions

The following are contributions of this thesis:

- A novel sparse feature matching method based on normalized cross-correlation (NCC) that presents significantly higher location accuracy and number of matches than state-of-the-art widely used SIFT matching. The correlation matching performs in real-time followed by a disambiguation.
- A model-based matching method for dense repeating features found in urban scenes, also based on NCC, that involves tracking planes to achieve wide-baseline matching and outperforms the location accuracy of SIFT by a substantial amount.
- A real-time implementation of multiple view stereo that recovers dense point clouds from

calibrated images with accuracy that is comparable to the slower state-of-the-art and speed that similar to the less accurate fastest methods. The images may be large and the entire pipeline is implemented on a graphics processing unit (GPU).

- A method to fit polygon mesh surfaces to the dense point cloud estimated by multiple view stereo using an approach designed to fit limited available memory and to perform well on real outdoor scenes. The method is volumetric and models space more densely than standard alternative approaches based on oriented points, thus estimating more representative surfaces.
- The proposed parallel implementations of the NCC that runs on a GPU and attains very high performance up to three orders of magnitude faster than alternative CPU methods and significantly faster than other parallel methods.
- The proposed MVS and feature matching approaches, which also run on a GPU exploiting the proposed parallel NCC method.
- All proposed approaches are automatic, work under rather general conditions and are described in detail.

1.3 Outline

This thesis is organized as follows.

- Chapter 2 provides a comprehensive review of the literature associated to the topics covered by this thesis: sparse feature matching, multiple view stereo and match disambiguation in the presence of duplicate structure.
- Chapter 3 reviews basic concepts common in computer vision that are used throughout this thesis.
- Chapter 4 describes the proposed sparse feature matching method based on normalized cross-correlation that achieves better accuracy than SIFT according to a feature location error criterion. The chapter also discusses a disambiguation procedure that allows the method to produce high numbers of correspondence estimates that are higher than those of SIFT matching. Such estimated correspondences are used for camera estimation. The cameras are refined by global optimization and employed in dense reconstruction of chapter 6.

- Chapter 5 presents a novel algorithm to provide correspondences for dense repeating feature patterns seen under significant viewpoint changes. The solution augments traditional matching on urban scenes with large buildings containing many windows since windows in a facade all have the same texture and, therefore, cannot be distinguished from one another based solely on appearance. This chapter describes the automatic detection and automatic matching of such repeating patterns between views.
- Chapter 6 discusses the proposed dense reconstruction approach using multiple view stereo that achieves real-time performance and extends a state-of-the-art method to deal with 3-d reconstruction in uncontrolled environments such as aerial scenes. This dense matching method is based in normalized cross-correlation, assumes cameras are given and does not require fixed illumination. The entire pipeline of multiple view stereo is implemented on a GPU. The chapter also proposes a fast method that estimates a surface as a mesh that fits the estimated point cloud. This complementary meshing method does not run in real-time since it was implemented on a single-core CPU.
- Chapter 7 provides details of the real-time implementations discussed in this thesis, including a description of the parallel algorithms and a review of programming on a GPU. The parallel methods are implemented in cross-platform language geared towards GPUs and were tested in multiple GPUs from multiple vendors. Details are given for all the pipelines with emphasis on the proposed parallel implementation of normalized cross-correlation, which achieves remarkable performance given sophisticated parallel programming design and the associated use of massive GPU parallelism.
- Chapter 8 validates the claims of the proposed matching and reconstruction algorithms through a series of experiments on several datasets showcasing the ability of the methods to operate on large scenes, handle high ambiguities, achieve real-time performance, show robustness to illumination changes and provide high location accuracy in feature matching. Experiments also compare the proposed approaches to existing alternative techniques.
- Chapter 9 concludes this thesis and discusses future work.

Chapter 2

Related work

This thesis is related to the broad literature in scene reconstruction and is primarily associated to feature matching, dense reconstruction and match disambiguation with essence in normalized cross-correlation and massive parallel processing on GPU (graphics processing unit). This chapter provides a review of key developments of each field starting with feature matching, followed by multiple view stereo for dense 3-d point cloud reconstruction, and then normalized cross-correlation match disambiguation. Emphasis is given to both CPU and GPU methods.

2.1 Feature detection and matching

Feature detection and matching are essential components of 3-d reconstruction as well as other fields, *e.g.*, object recognition [74, 36], image registration [141], robot localization [109, 29] and camera estimation [106, 132]. Matched features support the identification of objects seen from different angles, and, allow recovery of scene structure and camera motion jointly [122, 105, 59, 98, 131]. Features generally are processed in three steps: detection, description and matching (see figure 2.1). Detection consists of finding interest points, a descriptor is a vector describing the visual appearance of an interest point, and matching involves finding corresponding descriptors. The most important property of feature detectors is repeatability, which indicates whether features can be reliably found in two views taken under varying viewing conditions. An important property of descriptors is distinctiveness,



Figure 2.1: Feature detection and matching pipeline.

i.e., descriptor vectors should be similar only when describing similar features. Likewise, it is desirable to have descriptors with robustness to noise, to illumination changes and to geometric deformations. Feature matching is typically accomplished by finding similar feature descriptors. A common measure of dissimilarity among features is the distance between their descriptors' vectors.

Several methods for local feature detection and/or description have been proposed such as the Harris detector [52], SIFT [74], the Shi and Tomasi detector [104], DAISY [115], kernel descriptors [13], SURF [10], MSER [78, 62], GLOH [80] and normalized cross-correlation [54, 75, 139, 119, 127]. Scale-invariant feature transform (SIFT) features are detected as the extrema of a difference of Gaussians function applied to an image scale-space pyramid. Scale-space is a formal theory to model different image scales using a family of smoothed images. In order to increase the quality of the SIFT features, locations of low contrast or low edge responses are discarded. SIFT has a high-dimensional descriptor consisting of a histogram of oriented gradients describing the edge distribution on a region. The SIFT descriptor, illustrated in figure 2.2, is oriented and normalized to reduce effect of rotation and illumination changes, and, by construction, is invariant to scale, orientation, illumination, and partially invariant to affine transformations. See [136, 78] for affine-invariant alternatives to SIFT. SIFT is therefore not invariant to distortions due to viewpoint changes, however, most descriptors do not perform well under viewpoint changes of more than 30° [83]. SIFT matching is based on nearest neighbors and only nearest descriptors whose distance is less than 0.8 times the distance of the second closest descriptor are considered, while other matches are deemed ambiguous and discarded. DAISY features [115] present a descriptor that is robust to scale, illumination and viewpoint changes and is designed to overcome problems in dense wide-baseline stereo where homogeneous areas, large perspective distortions and occlusions affect feature descriptor windows of significant size. DAISY and SIFT features produce very similar histograms of oriented gradients, however DAISY computes them more efficiently, which is suitable for dense-matching. The name DAISY arises from the shape



Figure 2.2: Interest points detected as SIFT features (images from Lowe [74]). *Left*: a 233×189 pixels original image. *Right*: 536 detected interest point locations with SIFT descriptor displayed as a vector indicating scale, orientation and position.

of the descriptor that appears similar to a flower. Similarly to SIFT and DAISY, speeded up robust features (SURF) is another scale- and rotation-invariant blob detector and descriptor that also focuses on efficiency, but introduces artifacts that degrade matching when used densely [115].

The Harris and Stephens detector [52], and, the Shi and Tomasi detector [104], are rotation-invariant detectors that efficiently describe a corner based on the eigenvalues of a matrix via a function based on the trace and determinant of that matrix without computing the eigenvalues directly, as introduced in [52]. These detectors are discussed in more detail in chapter 4. The Harris and Stephens detector [52], one of the most popular interest point detectors, is widely known as *Harris* detector.

Normalized cross-correlation (NCC) is one of the simplest descriptors and consists of local pixel intensities. These intensities are compared statistically via a correlation coefficient, thus the comparison is robust to linear illumination changes. Heo *et al.* [54] use an adaptive NCC measure developed to be insensitive to color radiometric variations using invariant color information instead of raw intensity (or color) and their method does not suffer from the fattening effect that object boundaries are not reconstructed correctly due to outliers in the correlation window existing due to viewpoint change. Fattening effect due to viewpoint change is handled in [54] using a weighting scheme where the weights are taken from a bilateral filter. An early approach [30] warps correlation

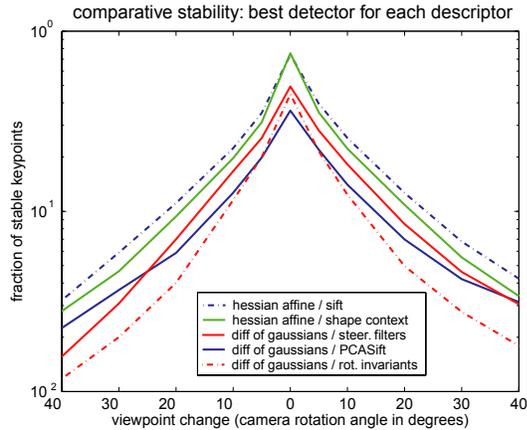


Figure 2.3: Performance of feature detection and matching for viewpoint change (image from Moreels and Perona [83]). The curves show stability results on a semi-log scale for several descriptors with the best performing detector for each descriptor. The *hessian affine* detector paired with a SIFT descriptor performed the best. The authors of [83] mention that no detector-descriptor combination performs well under viewpoint changes of more than 25–30°.

windows according to surface orientation that is estimated directly from stereo image pairs.

NCC is commonly used as a similarity or photo-consistency measure in dense stereo matching [124, 25, 126, 48]. Since NCC is the feature matching method of interest of this thesis, it is discussed in detail in chapter 3, while an efficient and speedy computation of NCC is proposed in chapter 7. Color images are converted to grayscale before computing NCC in this thesis.

All detector-descriptor combinations have advantages and disadvantages, therefore evaluation surveys have extensively tested descriptors in a wide range of conditions [80, 83, 121]. Mikolajczyk and Schmid [80] present an extensive evaluation of local feature descriptors that shows that SIFT and their proposed descriptor (GLOH) that is an extension of SIFT, present the best results in most tests. SIFT and GLOH robustness strength mostly relies on their histograms of oriented gradients, which are relatively robust to photometric and geometric transformations.

Moreels and Perona [83] explore the performance of feature detection and matching for a variety of descriptors under a variety of viewing conditions and their evaluation is more general than the one from Mikolajczyk and Schmid [80]. In [83], more general heterogeneous collection of 3-d objects are considered, while most transformations between views in the scenes of [80] could be modeled by a homography, *e.g.*, image rotations or planar geometry. Moreels and Perona [83] find that SIFT is

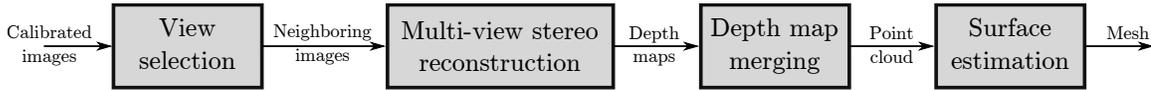


Figure 2.4: Dense stereo reconstruction pipeline.

one of the best overall choices and also favors SIFT for the specific case of stereo-vision applications (see figure 2.3). SIFT is then a natural choice for the benchmark feature use in the comparisons of this thesis.

Tuytelaars and Mikolajczyk [121] provide an comprehensive survey of stability for many feature detectors with respect to many properties such as repeatability, localization accuracy, distinctiveness, quantity and efficiency. SIFT was not surveyed in [121], however, the Harris detector [52] presented the highest localization accuracy and repeatability among rotation-invariant feature representations in the many tests performed in [121]. As NCC is a matching tool and not a feature detector, detection of interest points in this thesis are based on the Harris detector and more details about it are discussed in chapter 4. Interest points here are features that are easy to track using an NCC descriptor, such as corners of the same scale as the correlation template. In contrast, textureless regions are very difficult to match using NCC, or any local feature matching tool, and are not considered interest points here for the purpose of matching.

NCC is neither rotation- nor scale-invariant, a limitation that this thesis overcomes by assuming smooth inter-frame camera motion and using image registration techniques, such as rectification and planar surface tracking (discussed in detail in sections 3.1 and 5.5, respectively). The surveys of [121, 83] focus on invariant features and do not evaluate NCC. Mikolajczyk and Schmid [80] investigate cross-correlation and, in their analysis, NCC produces moderate results, *i.e.*, better than most of the surveyed feature descriptors in the context of matching, except against SIFT, one of the best performing, and some others similar to SIFT. In addition, the evaluation criterion of [80] is based only on number of correct matches and number of false matches, while localization accuracy is not evaluated. Moreover, no significant disambiguation of cross-correlation matches is performed, which would be fair as NCC has a very simple descriptor and does not benefit from stability enhancements built-in SIFT such as its invariance, its distinctiveness from its high dimension, and the elimination of detected points at low contrast or low edge response regions.

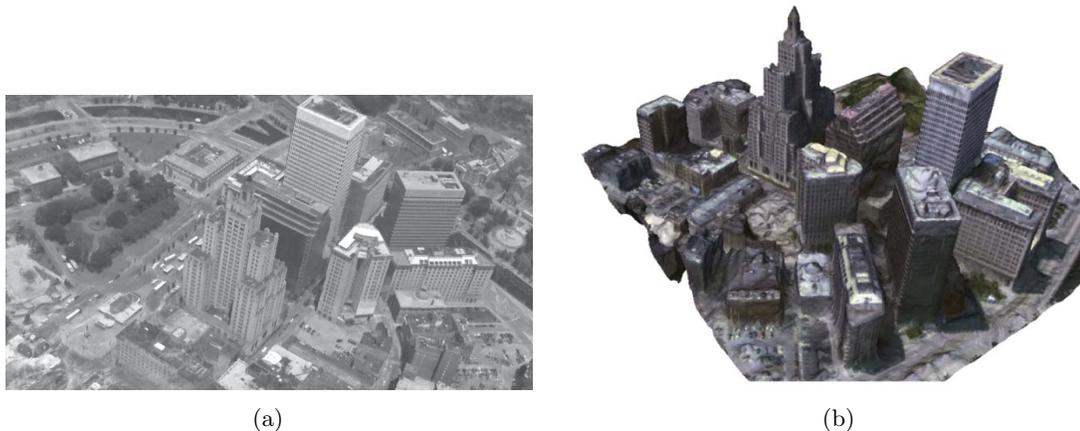


Figure 2.5: Image rendering from 3-d models learned from multiple aerial images of an urban scene. (a) An image rendered from a probabilistic volumetric model (image taken from Crispell [26]). The viewing location was not used when learning the model. (b) Reconstructed surface mesh that was textured using the learning images (image from Calakli *et al.* [21]).

This thesis does not revisit the experiments of Mikolajczyk and Schmid, but the proposed experiments show that the proposed NCC matching methods outperforms SIFT matching quality in terms of localization accuracy and in terms of number of matched features using the proposed novel disambiguation criteria (see chapter 4). This was expected by preliminary analysis [6].

This thesis then revisits NCC for its use in feature matching for accurate and real-time camera estimation applications, which is dominated by widely used SIFT [132, 2, 130, 105, 106, 46], a feature detection and matching method that has been successfully implemented on GPU [130, 129]. The proposed NCC computation parallel approach demonstrates superior localization accuracy and quantity of matches than SIFT, and, the associated proposed GPU implementation computes exact correlation values with phenomenal speedup w.r.t. CPUs.

2.2 Dense reconstruction

Given a set of images with calibrated cameras, most state-of-the-art reconstruction methods rely on a local measure of similarity to densely correspond pixels across images giving rise to dense 3-d geometry as point clouds and then impose global geometry constraints to fit a regularized surface via level sets [22, 63, 35, 89, 86, 61, 107], space carving [19, 65], dynamic programming [9, 73],



Figure 2.6: The two scenes from the multi-view stereo evaluation of [102]. *Left*: the “dino” object illustrated with one image and a shaded ground truth mesh. *Right*: similarly for the “temple” object.

graph cuts [124, 107, 66, 15, 85], belief propagation [37, 73], PDE [111], digital elevation models [5], parametric mesh evolution [33, 42, 32] or Expectation-maximization (EM) algorithm [110]. Geometry is usually estimated via analysis of multiple image pairs or groups. From each reference image and its neighboring images with an appropriate baseline, depth per pixel is perceived from the reference image viewpoint generating a depth image, denoted a *depth map*. The geometry of multiple depth maps can be pruned and combined into a single viewpoint-free representation by merging them into a 3-d point cloud such that its points are used as anchors to compute the full surface reconstruction (see figure 2.4).

Strecha [110] proposes a multiple view stereo (MVS) formulation that uses robust statistics to model visibility and depth jointly and independently of parameters, and includes a Bayesian generative model that has a process to generate the inliers and another for the outliers. Pollard and Mundy [94, 93] developed a probabilistic model that estimate surface appearance and occupancy in a volumetric framework via Bayesian learning that natively models geometric uncertainty and generates superior quality synthetic images from novel viewpoints. The discrete voxel model of [94] is extended to a continuous representation by Crispell [26, 27] using an octree data structure, which is suitable to reconstruct more detail by modeling the sparsity of the scene efficiently (see figure 2.5a). Miller *et al.* [81] provide a real-time implementation of [26]. Liu and Cooper [72, 73] use an optimized belief propagation algorithm to solve a volumetric Markov Random Field that jointly models all image formation constraints and returns a surface. The probabilistic optimization of [72, 73] deals well with matching ambiguities, wide-baseline matching and occlusion, generating fairly complete 3-d models.



Figure 2.7: Images taken from the twelve scenes of the multi-view stereo evaluation dataset of [112].

Calakli and Taubin [20, 22] use a smooth signed distance function to estimate surfaces as watertight meshes given oriented point clouds that can be obtained from MVS. The solution has been generalized to be applied to probabilistic volumes in [21]. Kazhdan *et al.* [61] also solve the surface fit problem given oriented points via a level set method called Poisson reconstruction. A benchmark for surface reconstruction is provided by Berger *et al.* [11] and show that, among surface fitting methods, the global ones are very robust to noise, whereas local methods applied to clean data are highly accurate.

Evaluations benchmarks have been created to compare the reconstruction quality of many MVS algorithms [112, 102] and they have boosted the quality of the proposed algorithms in terms of computational complexity, *accuracy* and *completeness*. Accuracy measures how close a reconstructed point is to the true one, whereas completeness establishes how much of the true surface is modeled by the reconstruction. The evaluations of [112, 102] consist of a small number of scenes (see figures 2.6 and 2.7). A more recent setup [58, 57] for large scale MVS evaluation includes eighty scenes with much higher diversity of objects than the ones seen in [112, 102]. Examples of such scenes are given in figure 2.8. The setup of [58, 57] provides the ground truth data used in this thesis to obtain quantitative results to judge the quality of the proposed MVS algorithm. The “ground truth” data of [58] comes from reference structure light scans that are in fact very dense and accurate physical measurements showing errors reported to be from 0 to roughly 0.28 mm.

2.2.1 Real-time 3-d reconstruction

This thesis is also related to real-time 3-d reconstruction. Commodity graphics hardware has been used to accelerate 3-d reconstruction from images since the computational complexity of the problem is considerably high and its optimization space is large. Izadi *et al.* [56] propose a system that reconstructs a scene using a depth sensor based on a limited-range infrared structured light. This

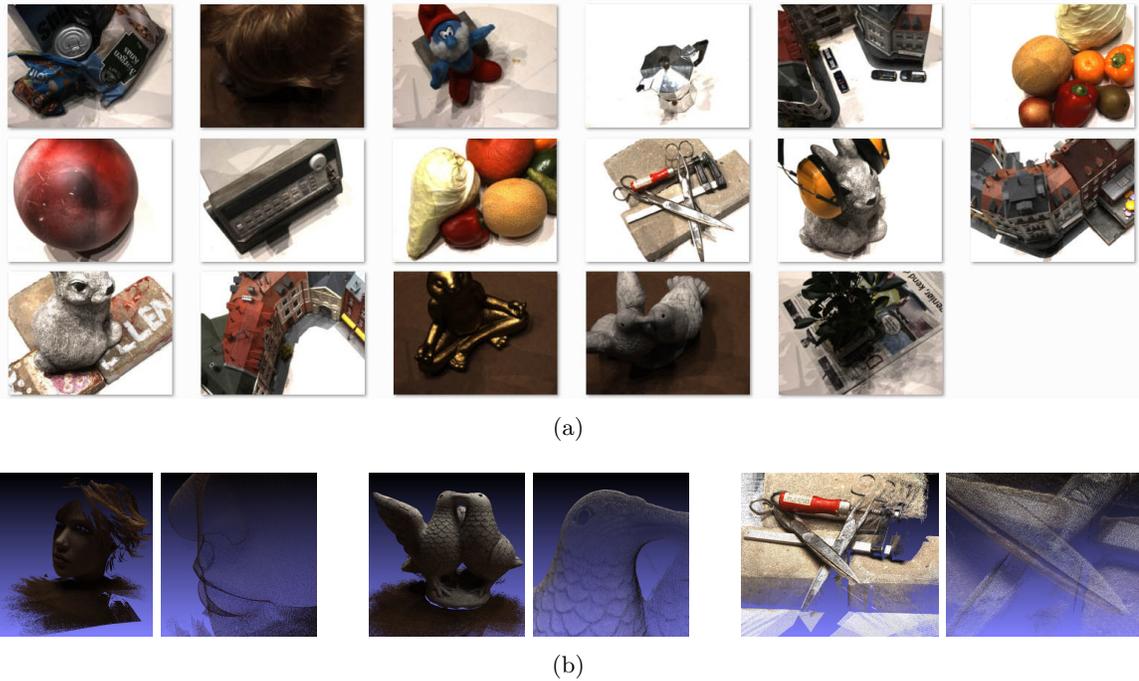


Figure 2.8: (a) Images from some of the 80 scenes of the evaluation dataset of [58] illustrating the variety of objects and surface materials that also includes specular surfaces. (b) Ground truth point clouds for three of the 80 scenes of [58] with detail to visually illustrate measurement accuracy.

type of method retrieves geometry without need for images, however it is mostly suitable for indoor environments with little to no feasibility for outdoor scenes and large scale aerial reconstruction. Zhao and Taubin [140] implemented real-time stereo technique on a GPU that models objects that move in front of the two-camera stereo setup.

There exists numerous general multiple view stereo algorithms [102], however, the number of methods that offer real-time performance is relatively small compared to the total number of methods. Furthermore, algorithm evaluations are often performed in controlled indoor environment with simple objects [102]. MVS algorithms that run in real-time are typically processed on GPUs [126, 81, 25, 125, 84, 67]. Vogiatzis and Hernández [125] developed a video-based real-time multiple view stereo reconstruction and show that the denser coverage of the object provided by the video improves accuracy but lacks completeness compared to similar methods. Vu *et al.* [126] have a system that works well on large-scale high-resolution imagery of relatively uncontrolled environments producing detailed reconstructions, as shown in figure 2.9, and a refined mesh obtained from dense point



Figure 2.9: Large-scale reconstruction from Vu *et al.* [126].

clouds using graph cuts and variational optimizations. Similarly, Miller *et al.* [81] also estimate a very detailed volumetric model in relatively unconstrained conditions using an efficient sparse volumetric representation from which a mesh surface can be extracted using the technique from [21] (see figure 2.5b). Chang *et al.* [25] introduce a novel MVS method that runs entirely on GPU and is based on surface elements that are estimated via parallel voting of position and orientation for multiple depths. Other algorithms [84, 67] are designed for efficiency and their accuracy is slightly degraded. Evaluations discussed in chapter 8 indicate that the accuracy and speed of the proposed reconstruction method is comparable to or better than [126, 21, 25].

The design of the real-time MVS method of this thesis is motivated by state-of-the-art methods originally proposed for CPU [124, 48], which are generalized and combined into a single novel framework in this thesis that runs on GPU to meet accuracy and speed constraints simultaneously, and perform well in uncontrolled outdoor or aerial scenes. Vogiatzis *et al.* [124] propose an MVS method robust to occlusions that estimates depth based on correlation peaks in a photo-consistency similarity-function along rays and uses graph cuts to solve for a minimal surface that is a result of the fusion of several depth maps. Reconstruction via graph cuts is discussed in detail in section 6.7. Goesele *et al.* [48] propose a similar MVS framework where only the most reliable correlation peaks are considered, which tends to increase accuracy and remove spurious reconstructions at a slight risk of decreasing completeness. These methods are relevant for this thesis goals and contributions since they offer quality reconstructions according to [101, 102] and a solution based on NCC that can be

parallelized given that scene geometry can be learned from images in an online fashion. Additional improvements are proposed to generalize the proposed method for general large-scale real outdoor scenes that may have complex geometry and occlusions, such as aerial scenes of urban areas with tall buildings. Note, the methods described in [124, 48, 134, 25] show results performing only in indoor or controlled environments, some other real-time approaches are video-based [125, 95], and the state-of-the-art MVS pipeline of Furukawa and Ponce [41] use a rather sequential region-growing approach that is not entirely suitable for a parallel implementation.

The implementation of Chang *et al.* [25] share some similarities with the proposed one since they describe an MVS step to estimate multiple depth maps taken from [124] and use graph cuts to extract smooth and clean surface elements from a set that may contain outliers. The several differences are: (1) their GPU implementation of the method of [124] is faithful to the original method, while the proposed one modifies [124] to make it even more robust to occlusions; (2) their use of graph cuts for regularization is completely novel, whereas the proposed one is an improvement over [124]; (3) their high-performance and feasibility of using graph cuts optimization on a 3-d volume is achieved using impressive and elaborate implementation using an octree partition of 3-d space and their proposed parallel graph cuts algorithm for GPU, while the proposed one greatly simplifies the optimization complexity to achieve fast results using fixed voxel grid and serial graph cuts on CPU; (4) their NCC operations run on GPU processors via a scalar function that computes the correlation coefficients independently, whereas the proposed method access images more efficiently and less redundantly for overlapping correlation patches common in dense MVS and general applications; (5) no tests on uncontrolled environments such as aerial imagery are reported by [25]; (6) their implementation is in CUDA, a language that runs exclusively on Nvidia GPUs, whereas the proposed one is written in OpenCL, a cross-platform framework that runs in Nvidia, AMD, Intel and other chips; and (7) their GPU pipeline speedup is in the order of 100X faster than their CPU implementation, while the proposed NCC computation speedup can reach 1000X or more (see chapter 8). NCC is the dominant computation of the proposed pipeline in terms of complexity, therefore such speedup is important.

The dense NCC matching performed here uses the same general parallel implementation used in the aforementioned proposed sparse feature matching algorithm (section 2.1). Chapter 4 presents

the sparse feature matching method, chapter 6 discusses the proposed MVS method and their shared parallel NCC implementation is discussed in detail in chapter 7.

Another general contribution of the proposed MVS approach w.r.t. alternative methods [124, 48, 25] is the use of weighted NCC such that pixels near the center of corresponding image patches are weighted more heavily during the matching process, a desired effect since the center is not affected by geometric distortions. The weighting mitigates a trade-off between bias and variance on correspondence estimation given the size of the feature descriptor. Small template patches are more accurate but noisier (more incorrect matches), while larger ones have high discriminative power but present location bias. This trade-off is discussed in detail in section 3.2.2. The problems are mitigated by using medium sized *weighted* correlation patches.

In summary, this thesis presents an MVS method that take advantage of modern GPUs to achieve real-time dense multi-view stereo matching for general large-scale outdoor scenes generalizing other algorithms designed for a sequence of still images. Additionally, it uses an efficient graph cuts methods to fit a surface, on CPU, to the estimated point clouds.

2.3 Disambiguation of NCC correspondences

Correspondences found via NCC matching that are assigned to points with the highest correlation scores are not particularly reliable as in SIFT matching since the NCC descriptors lacks high distinctiveness. Some reliability pruning is recommended, *e.g.* rejecting points in low contrast areas and considering ratios of distances between the nearest descriptor and the second nearest descriptor to express match quality. These procedures help disambiguate correspondences and clean them from outliers.

Disambiguation is often necessary for the case of sparse matching for camera estimation. Normally, only the epipolar geometry [53] is exploited, as in [125, 103]. Epipolar geometry is discussed in detail in chapter 3 and simplifies correspondence search to be simply along a line in an image (epipolar line) associated to the viewing ray of the corresponding feature in another image. For practical purposes, most existing work merely consider looking for features in a fixed distance from epipolar lines. Unlike traditional approaches, Fabbri and Kimia [34] propose a distinct correspondence disambiguation

based on differential curve properties such as torsion and curvature derivatives. Son *et al.* [108] disambiguate 3-d curve matching using object global constraints. Bhattacharya and Gavrilova [12] reject point matches based on the topology of neighboring matches. Almeida and Cooper [4] use planar models and epipolar geometry to resolve the very ambiguous problem of matching dense repeating features that are common in urban scenes (discussed in more detail in chapter 5).

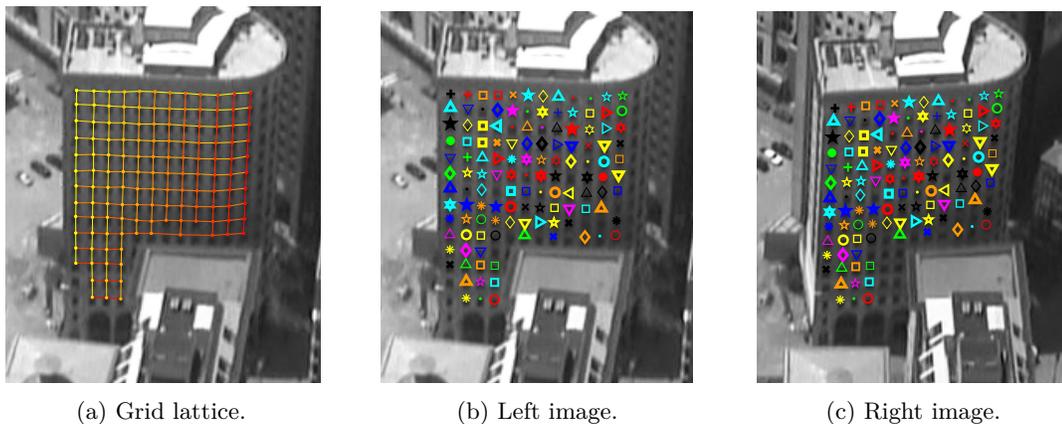
This thesis provides two novel disambiguation methods for NCC correspondences: (1) a general model-free disambiguation pipeline for camera pose estimation (chapter 4), and (2) a model-based disambiguation for dense repeating features (chapter 5). The proposed model-free procedure of chapter 4 uses ratio tests, epipolar geometry, three-view geometry and match topology consistency tests. As in [4, 103], epipolar geometry is estimated from the unambiguous matches and used to guide matching of other points that may have more than one similar match near an epipolar line. The main novelty is the adaptive distance threshold used to reject or accept correspondences based on their distance to an epipolar line. This threshold—computed for every interest point—is based on data and is iteratively refined. Note, the disambiguation problem when using NCC is different than when SIFT features are used [106, 103]. SIFT feature points are found *independently* in an image, then a set of SIFT points found in one image are compared to all other SIFT points found in another image. In the proposed NCC matching case, interest points are found in a reference image and a correspondence search for each point is carried out once in another image using NCC and this search may be reduced given epipolar geometry. No detection of interest points in the other image is necessary for matching.

2.3.1 Disambiguating dense repeating features

The proposed disambiguation scheme of chapter 4 is not sufficient to handle very dense repetition such as windows on large buildings in aerial views of urban scenes. The windows are seen very close to each other in large dense clusters, as seen in figure 2.10. Duplicated structures are ubiquitous in urban scenes and handling their inherent match ambiguity has been an important topic in computer vision. Scene structure duplicates usually consist of man-made objects and are commonly found in architectural scenes. Facades and windows are among the most common repeating elements in urban



Figure 2.10: Facades of large buildings, which normally present piecewise planar grids of dense repeating features (windows).



(a) Grid lattice.

(b) Left image.

(c) Right image.

Figure 2.11: Typical example of the model-based pairwise matching method of this thesis for dense repeating features. Correspondences are disambiguated by detecting the underlying planar grid of features (a), and using its associated global constraints to select the correct matches (b,c). Corresponding points have identical markers.

scenes and epipolar geometry may reduce ambiguity but does not eliminate it since there is still numerous match candidates near an epipolar line. In order to deal with general urban scenes, the author proposes a solution in chapter 5, published in [4], to handle such ambiguous cases under the reasonable assumption that such dense repeating points are on a planar surface and distributed in a grid, which is normally true (see figure 2.11). Chapter 8 has experiments showing that this method improves the accuracy of camera estimation and recover a high number of repeating features from building windows, a rich set of correspondences that is usually lost by traditional matching.

Dealing with replicated objects is important in computer vision since they often cause issues in *structure from motion* (SFM), the problem of simultaneously estimating camera poses (motion) and 3-d points (scene structure) from 2-d image correspondences alone [53, 122]. Duplicated objects often give rise to erroneous or ambiguous matches and their detection and disambiguation must rely on global consistency measures. For instance, consider a simple scene with two identical oatmeal boxes, box 1 and box 2, as illustrated in figure 2.12a, and a reference image sees only box 1, as in figure 2.12b. This scene illustrates two common possible ambiguities types. Traditional matching, *e.g.* via SIFT [74], may incorrectly associate the image objects if the sensed image sees only box 2 (*type I* ambiguity), as in figure 2.12c. SIFT will likely fail to provide correspondences from the boxes if the sensed image sees both of them, as it cannot disambiguate the nearly identical local descriptors (*type II* ambiguity). Note, *type I* ambiguity causes an error, whereas *type II* leads to missing matches (see figure 2.12). These issues have motivated work in scene disambiguation to determine the *correct associations*, *i.e.* matching features that correspond to the same 3-d points. Most existing work focus on *type I* ambiguity such that repeating elements are *large* objects repeating only a few times in a scene. Progress has been made in terms of detection and matching in the presence of such objects [98, 77, 113, 51, 138, 59]. Incorrect associations due to *type I* ambiguity may cause erroneous 3-d reconstructions, such as phantom surfaces and superimposed structures, *e.g.*, when all instances of a repeating object is reconstructed only once in 3-d [98, 128]. Such folded reconstructions may occur if repeating objects produce more features than the rest of the scene and dominate the SFM process causing complete failure. In the context of robotics, disambiguating repeating object associations is crucial for autonomous navigation in determining whether a detected landmark has been previously seen (loop closure) or it is new [51, 97, 47].

Specialized SFM algorithms for handling repeated objects in a scene have been proposed [138, 98, 59, 128, 24] and typically use SIFT matching for finding correspondences. Zach *et al.* [138] examine correspondence triplets to find erroneous associations, since triplets allow predicting a feature location in a third image. Both [138] and [98] note that if a large set of points, located in a foreground object and the background, match in two views, but not in a third, then the third view may be seeing a different instance of the object. In [128], repeating architectural features found on large outdoor



Figure 2.12: Scene with replicated structures, which typically causes confusion in matching algorithms. (a) Each red oatmeal box has several distinct features also found on the other oatmeal box. The image is from [98]. (b) An image that sees the oatmeal box on the left, but not the other. (c) An image that sees the oatmeal box on the right, but not the other. Traditional matching may erroneously match several features of the different oatmeal boxes if applied to (b) and (c), or fail to decide where to match, in image (a), the well-defined features of the oatmeal box seen in (b).

unordered images are disambiguated considering their local visibility structure via a non-geometric graph-topological model. [24] couples symmetry information with SFM to resolve ambiguities in matching building facades using an interactive system where the user segments the repeating element. Another category of similar problems is to parse building facades into semantic regions or to detect facades [77, 113, 71, 31]. Most existing work [77, 113, 133] exploits facade regularity exclusively through datasets showing ground level fronto-parallel (rectified) facade views. Mobahi *et al.* [82] presents a 3-d reconstruction of urban buildings that relies directly on semi-global or global image patches and not in traditional local features. Wu *et al.* [133] exploit facade repetition patterns for stereo reconstruction directly from a single image of multiple repeating elements. These prior methods have been applied exclusively to repeating elements found in large objects that appear a few times in a scene. Thus, the repeating objects are in high resolution and their features also repeat only a few times in a scene. Moreover, the methods focus on the *removal* of wrong associations, rather than on simultaneous stereo matching of multiple duplicates.

In contrast, this thesis proposes a solution for *type II* ambiguity by tackling repeating elements displaying *massive dense repetition*, such as hundreds of nearly identical features in a single image. The proposed approach robustly match the dense repeating features under viewpoint change, as in figure 2.11. Additionally, the features are tracked even after very wide-baseline motion through a sequence of smaller motions (discussed in detail in section 5.5). The features are assumed to be



Figure 2.13: Automatic detection of deformed lattices in a single image (images from Park *et al.* [90]).

organized in a planar grid and the grid is distant from the camera such that perspective distortions are approximately affine, which is realistic in urban scenes. Under these assumptions, viewpoint change transformations of these features can be modeled by a homography that can be estimated once the matches are disambiguated. The regularity and topological properties of grids are used for single view grid detection and also for matching grids in other views. There are no image resolution requirements, however the method targets small, dense and highly repetitive elements, where match disambiguation often fails *e.g.* simply by enforcing the epipolar constraint, and would be very challenging without reasoning on the global properties of the estimated planar grid lattice. Epipolar constraint and other spatial cues often suffice to disambiguate matches of sparsely repeating elements. In order to deal with inevitable occlusions of aerial views, the detection and matching are robust to incomplete detected grids and also missing matches.

Park *et al.* [90] relax the planar assumption and detect a deformed lattice from repeated patterns in a single image, but do not propose stereo matching. Deformed lattice matching is usually achieved using spatial-temporal tracking assuming negligible inter-frame motion and do not handle significant baselines [70]. The real-time tracking algorithm of Lin *et al.* [92] for non-rigid surfaces handles wide-baselines, but not the ambiguity of repeated patterns, as they cause a serious correspondence problem. Schindler *et al.* [100] match 3-d patterns from a pre-existing database of geolocated planar facades to newly detected facades exploiting the repeating nature of buildings to match the lattice rather than individual points.

Recent work of Liu and Liu [71] detects and associates facades on unconstrained aerial views and it is more directly related to this work. [71] uses the regularity of edge maps of the facade windows to detect lattices via common edge orientations and achieves good results in detecting multiple facades per image and in associating detected facade regions in different viewpoints, *i.e.*, they match the

facades but not the repeating points.

Note, in one-shot 3-d scanning with structured-light [60, 123], a similar matching problem exists, where an object is illuminated by a single set of known color-coded grid lines. Correspondences must be established between grid points and their location in an image of the illuminated object. This problem differs from the proposed one in some aspects: (1) the matching is independent of object appearance and is driven by known grid shape and colors; (2) in general, the grid lines projected onto the object form no lattice; and (3) it is not suitable for outdoor scenes. Furthermore, the grid pattern is assumed known as well as the camera and the projector intrinsic parameters and relative pose.

Unlike previous work, the proposed method focuses on dense repeating elements, *i.e.* type II ambiguity, and handles *detection* and *matching* under viewpoint changes with a significant baseline. The procedure targets aerial views of tall building facades, where the common repeating elements are hundreds of windows per facade, but it is not limited to such objects or imagery. Windows are commonly seen as either corners, short edges or very small parallelograms. A global assumption is necessary to handle this extremely high level of ambiguity. Thus, the features are expected to be in a flat lattice only a few pixels away from each other, a realistic on aerial urban scenes.

The proposed pipeline uses regularity constraints to detect a grid repeating-elements and estimate the hypothesized underlying planar lattice, a single-image process. Grid point correspondences are then found using NCC and they are extremely ambiguous. Matches are disambiguated based on the epipolar constraint and global properties of lattice points such as piecewise collinearity, even spacings and line intersections. Images are assumed to be ordered and only adjacent frames are matched. No prior information about camera parameters is assumed, yet the aerial views must have enough unambiguous points to allow estimating epipolar geometry. The framework handles images with *multiple* lattices, each lattice with possibly hundreds of nearly identical features, as in figure 1.6.

In summary, the main contribution of this disambiguation strategy is the novel planar lattice model to handle a very ambiguous image correspondence problem on matching a large number of similar features, which allows the registration of the planar surfaces via homographies to achieve straightforward ultra wide baseline matching via NCC, and improve the accuracy of SFM. Experiments validating these claims are introduced in chapter 8.

Chapter 3

Review of basic algorithms

This chapter reviews fundamental algorithms used throughout the thesis.

3.1 Rectification

Rectification consists of image transformations that simplify implementing a search for corresponding points in two views of the same scene. In general, the search can be reduced to a slanted line in the image, which is mapped to a horizontal line (an image row) after rectification. This geometric constraint is derived from epipolar geometry, which is briefly reviewed in the next paragraph.

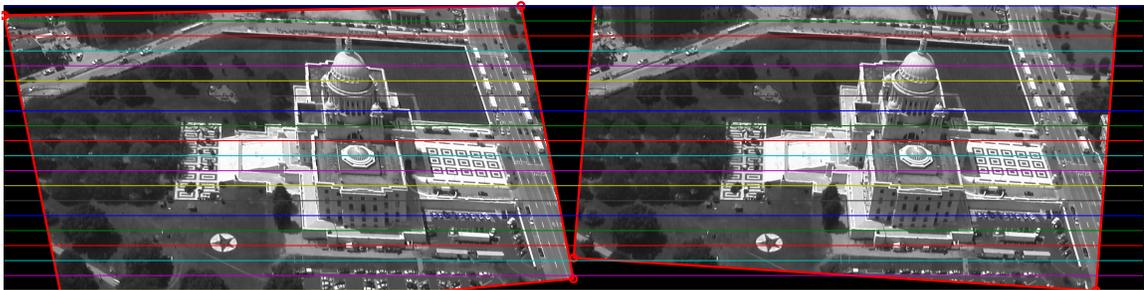


Figure 3.1: Side by side rectified views. Matching features lie on the same image row, since epipolar lines are horizontal and there is no vertical disparity.

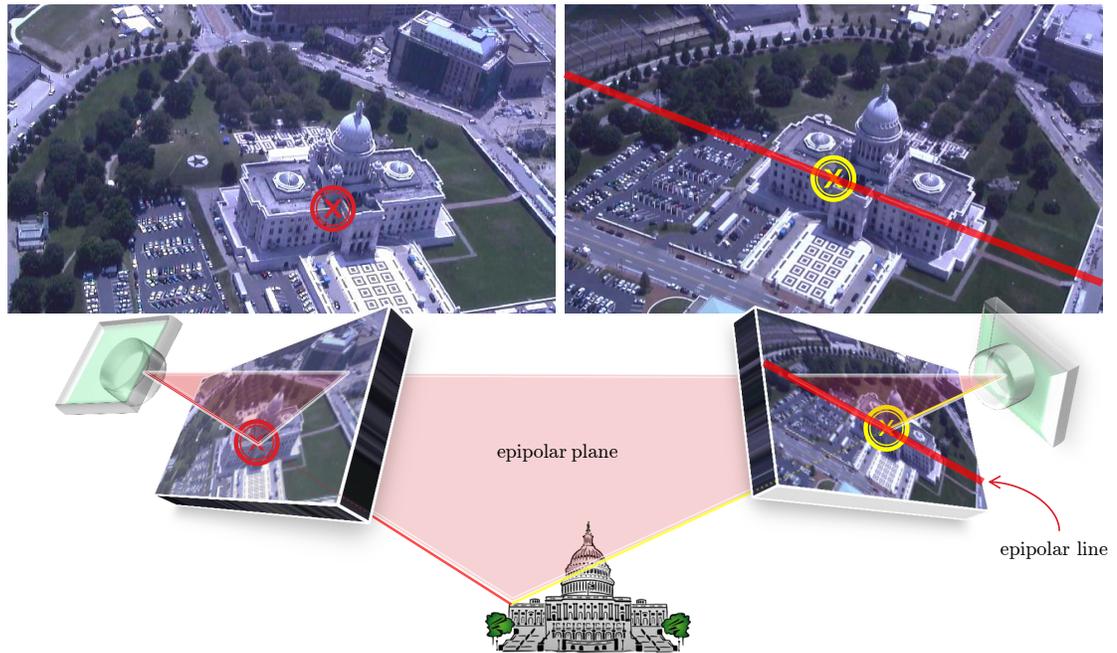


Figure 3.2: Geometric relations between matching points on a stereo pair. *Left*: pixel in a reference image. *Right*: matching point and associated epipolar line in other image. *Bottom*: 3-d scene showing camera centers, epipolar lines, image planes, the associated epipolar plane, a 3-d point and its depths from each camera center.

Epipolar geometry concepts. According to the epipolar geometry of two views [53], *baseline* is the 3-d line connecting the two camera centers, *epipole* is the intersection of an image plane and the baseline, *epipolar plane* is a plane containing the baseline, and *epipolar line* is the intersection of an epipolar plane and an image plane (figure 3.2). For a given baseline, there is a pencil of epipolar planes whose intersections with the image plane define a pencil of epipolar lines, all meeting at the epipole. Geometrically, the image projection of an arbitrary 3-d point in the scene lies on an epipolar line associated with the epipolar plane defined by the 3-d point and the baseline. Thus, given an arbitrary image point, its corresponding point in the other view must be constrained to the associated epipolar line. This property, called the *epipolar constraint*, reduces correspondence searches from the entire image to a single line. The fundamental matrix \mathcal{F} is a 3×3 rank 2 matrix that captures this intrinsic geometry.

Let \mathbf{x}_R and \mathbf{x}_T be corresponding locations in two images represented in homogeneous coordinates:

$$\mathbf{x}_R = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad (3.1)$$

$$\mathbf{x}_T = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}. \quad (3.2)$$

It is well known that the epipolar line of \mathbf{x}_R is computed by multiplying the fundamental matrix \mathcal{F} by the vector \mathbf{x}_R . For any \mathbf{x}_R other than the epipole \mathbf{e} , let

$$\mathbf{l} = (\mathcal{F}\mathbf{x}_R) \quad (3.3)$$

be the epipolar line of \mathbf{x}_R represented by the three coefficients in \mathbf{l} . The epipolar line \mathbf{l}' of \mathbf{x}_T is found in a similar way:

$$\mathbf{l}'^\top = (\mathbf{x}_T^\top \mathcal{F}) \quad (3.4)$$

From the definition of epipolar lines, \mathbf{x}_T lies on \mathbf{l} , then

$$\mathbf{x}_T \mathbf{l} = 0, \quad (3.5)$$

and similarly, \mathbf{x}_R lies on \mathbf{l}' implying $\mathbf{l}'^\top \mathbf{x}_R = 0$. The epipolar constraint follows from equations (3.3) and (3.5):

$$\mathbf{x}_T^\top \mathcal{F} \mathbf{x}_R = 0, \quad (3.6)$$

where \mathcal{F} is the fundamental matrix of the image pair where corresponding points \mathbf{x}_R and \mathbf{x}_T lie. The estimation of \mathcal{F} is discussed in section 4.4.2). Note, $\mathbf{l} = (\mathcal{F}\mathbf{x}_R)$ always contains the epipole \mathbf{e}' . It follows that $(\mathbf{e}'^\top \mathcal{F})\mathbf{x}_R = 0$ for all \mathbf{x}_R , then \mathbf{e}' is the left null-vector of \mathcal{F} and similarly, \mathbf{e} is the right null-vector of \mathcal{F} . Transposing equation (3.6) gives that \mathcal{F}^\top is the fundamental matrix for the views in the opposite order with \mathbf{e} , \mathbf{e}' as the left and right null vectors, respectively.

Image Rectification. The search for image correspondences would be one-dimensional and along horizontal epipolar lines, if two cameras were coplanar (placed side by side), with the images horizontal axes parallel to the baseline. Moreover, the epipoles would be at infinity along the horizontal axis. Even if two cameras are not coplanar, their images may be warped to emulate this condition by mapping them to a common plane. This process is called image *rectification* and is motivated by the

fact that it is computationally cheaper to search for corresponding points on the a line aligned with the rectangular image grid [53], as well as the fact that rectification is a much cheaper computation than the correspondence search.

The rectification process consists of finding the epipoles and defining projective transformations that map them to infinity along the baseline direction and rotating the images such that this axis is horizontal. The transformations warp and re-sample the images such that epipolar lines are aligned exactly to image rows and corresponding points have no disparity in the vertical direction as in figure 3.1. Note, if an epipole is inside an image, rectification may result in undesired distortions, since the transformation is continuous and will cause the neighborhood surrounding the epipole to also tend to infinity, resulting in potentially disconnected mappings (see figure 3.3).

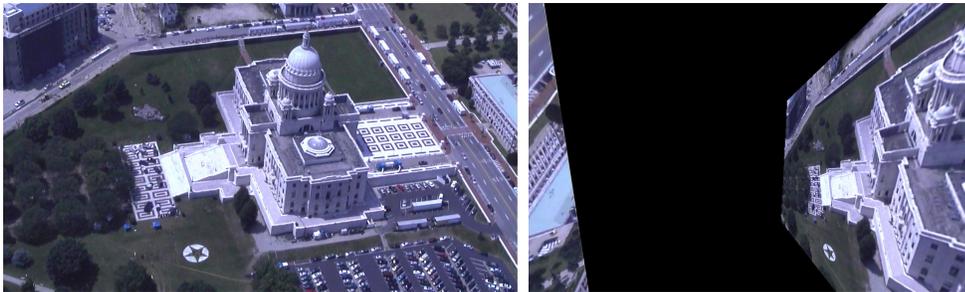


Figure 3.3: Illustration of undesired distortions due to rectification of an image pair where an epipole is located in the image plane (only one image is shown). *Left*: a regular image taken from a camera. *Right*: a rectification of the image on the left when the epipole is located inside the image. In this case, the rectified image is disjoint with two components.

3.2 Weighted normalized cross-correlation

Normalized cross-correlation is a popular measure of similarity between the appearance of a feature point in two different views. It is used in this thesis both for feature matching to estimate camera poses (chapters 4 and 5) and for dense multi-view stereo (chapter 6). Normalized cross-correlation is robust to changes in brightness due to lighting and exposure conditions.

The weighted normalized cross-correlation of a template image patch $t(x, y)$ with a subimage $S(x, y)$ consists of a 2-d array $C(u, v)$ of weighted correlation coefficients computed as $t(x, y)$ slides

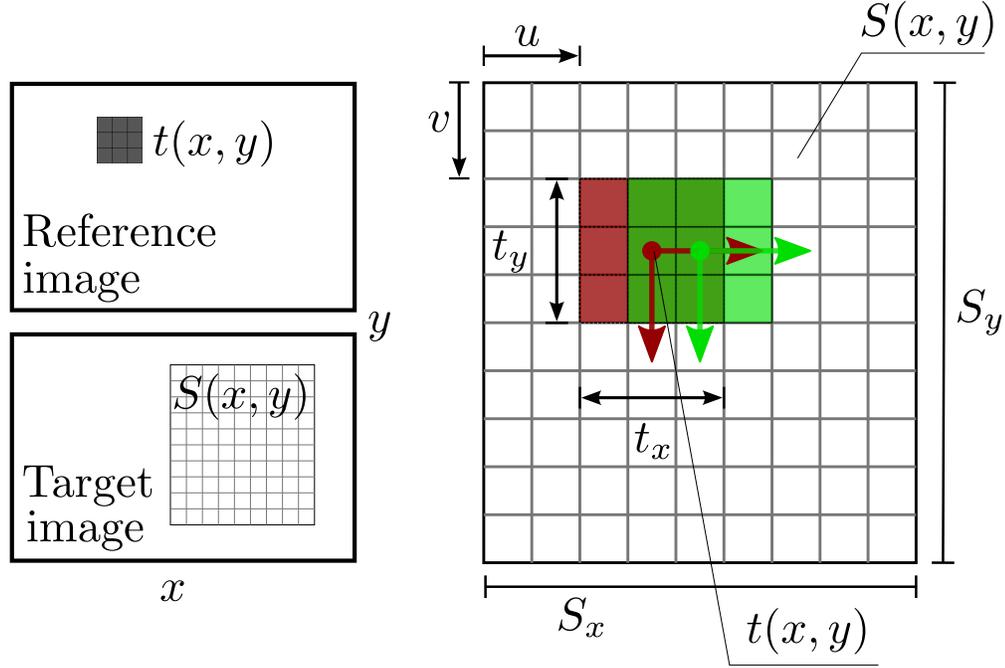


Figure 3.4: Image matching via normalized cross-correlation. A template $t(x, y)$ taken from a reference image is correlated with a larger subimage $S(x, y)$ from a second image. Arrows depict the relative 2-d motion of the subimages $t(x, y)$ and $S(x, y)$. For each pixel location the template is shifted, given by u and v , a new similarity coefficient $C(u, v)$ is computed.

over $S(x, y)$ at one pixel steps. Given a weight function $w(x, y)$, the array C is defined as

$$C(u, v) = \frac{\sum_{x,y} \left\{ w(x, y) [S(x+u, y+v) - \overline{S_w(u, v)}] [t(x, y) - \overline{t_w}] \right\}}{\sqrt{\sum_{x,y} \left\{ w(x, y) [S(x+u, y+v) - \overline{S_w(u, v)}]^2 \right\} \sum_{x,y} \left\{ w(x, y) [t(x, y) - \overline{t_w}]^2 \right\}}}, \quad (3.7)$$

where $\overline{t_w}$ and $\overline{S_w}$, expressed as

$$\overline{t_w} = \sum_{x,y} w(x, y) t(x, y), \quad (3.8)$$

$$\overline{S_w(u, v)} = \sum_{x,y} w(x, y) S(x+u, y+v), \quad (3.9)$$

are respectively the weighted mean of the template and the weighted mean of the portion of $S(x, y)$ under the template (see figure 3.4). Subtracting averages and dividing the signals by the weighted standard deviations as in equation (3.7) normalizes the data to have zero mean and variance one, so $C(u, v) \in [-1, 1]$ are correlation coefficients invariant to brightness changes. The weight function

$w(x, y)$ satisfies

$$\sum_{x,y} w(x, y) = 1, \quad (3.10)$$

$$w(x, y) > 0. \quad (3.11)$$

In this thesis, $w(x, y)$ is chosen as a 2-d Gaussian weight function centered at the template centroid with a diagonal covariance matrix as follows:

$$\Sigma_w = \text{diag}(\sigma_x^2, \sigma_y^2). \quad (3.12)$$

The Gaussian function constructed using Σ_w from equation (3.12) induces the dependency of $C(u, v)$ in $t(x, y)$ to evenly decay away from the template center (discussed in more detail in section 3.2.2). If $w(x, y)$ is constant, the weighting is uniform and the similarity measure is simply called *normalized cross-correlation*.

3.2.1 Size relations

The sizes of the template image $t(x, y)$, the subimage $S(x, y)$ and the correlation array $C(u, v)$ are such that

- $t(x, y)$ size is $t_x \times t_y$,
- $S(x, y)$ size is $S_x \times S_y$,
- $C(u, v)$ size is $C_u \times C_v$,
- $S_x \geq t_x$ and $S_y \geq t_y$,
- $S_x = C_u + t_x - 1$,
- $S_y = C_v + t_y - 1$.

3.2.2 Why is weighting important?

Weighting normalized cross-correlation admits a simultaneous reduction of bias and noise in image matching. When comparing the correlation patches centered at two perfectly matching points in two distinct viewpoints, the only truly matching pixel is usually the center, due to distortions from viewpoint change. Pixels away from the center may cause location bias, expressed as a small shift of

the correlation peak (apparent match location) away from the true match position. Smaller templates exhibit smaller biases, but more noisy correlation values, whereas large templates exhibit more bias and less noise. Isotropic Gaussian-weighted correlations achieve a reduction in bias of smaller templates due to the decay away from the center, while preserving the smaller noise levels from larger templates. An isotropic Gaussian kernel is not necessarily optimal, but probably suboptimal for weighting correlation, and an example of the good empirical benefits of using an isotropic Gaussian weight function is given in figure 6.2.

3.2.3 Why is normalization important?

The coefficient $C(u, v)$ computed in equation (3.7) is normalized by subtracting the means of the signals and dividing by their standard deviations. The coefficient is used as a measure of similarity between two signals. The higher $C(u, v)$ is, the more correlated the signals are. Since $C(u, v)$ is in the range $[-1, 1]$, thresholding reliable matches is as simple as $C(u, v) > \tau$, for some quality parameter τ .

If the similarity function was only the numerator of equation (3.7), the range of values would be undefined and vary with the amplitude of the variances of the signals. In this unnormalized case, it would be difficult to distinguish low similarities due to wrong matches from the ones due to true correspondences that have smaller variances. Similarly, high-variance unnormalized signals may present high scores even if highly uncorrelated. The normalization of $C(u, v)$ is stable for a broad range of variances, except for extremely low ones where there may not be enough numerical precision to represent signal fluctuations. Nevertheless, the instability of $C(u, v)$ on very low variance cases is due to signal representation and not the normalization, which still guarantees a coefficient in the range $[-1, 1]$.

In general, normalization is crucial for template matching in real-world scenes. Surface reflectance can be approximated by a Lambertian model with an albedo. The albedo is the actual signal of interest one tries to match as it is an intrinsic property, invariant to viewpoint and illumination. However, the albedo is not measured directly by pixel intensities, but it is coupled with illumination, which affects observations in different ways depending whether the illumination has direct or diffuse components. Illumination of outdoor scenes is often a mixture of a broad spectrum of direct sunlight, diffuse

skylight and interreflections. In a dynamic scene where images are collected sparsely, illumination components may change locally or globally from one image frame to the next, including image brightness, contrast and saturation. Shadows arise from blocking a direct component reducing the magnitude of the signal. An unnormalized similarity measure is not suitable for matching in these aforementioned cases.

In the case of high frame rate video sequences, where illumination and viewpoint are essentially constant on adjacent frames, other similarity measures are recommended, such as *sum of squared differences* (SSD). When illumination is fixed, both normalized cross-correlation and SSD provide high matching scores to true corresponding points. However, SSD will assign a low matching score to features that are not views of the same 3-d point, but are similar and are seen under distinct brightnesses. Normalized cross-correlation would provide a high matching score for such similar features, generating incorrect matching candidates that could be avoided by using the computationally cheaper SSD if illumination is known to be fixed.

In this thesis, the focus is on scenes where data is captured sparsely, with noticeable viewpoint changes, with possible illumination variations and with possible time-lapses between adjacent frames, including, but not limited to, high-resolution imagery from satellite and aerial vehicles. Radiometric factors such as imaging device changes, illumination color and illumination direction may affect intensity consistency between images [54] and normalized cross-correlation is robust to different radiometric conditions, unlike SSD. For instance, surfaces that are considered primarily Lambertian, may in practice still exhibit significant specular effects and reflect more light in some directions than in others, causing noticeable radiometric variations if seen from viewpoints along such directions.

3.3 Delaunay triangulation

A 2-d Delaunay triangulation consists of a subdivision of a plane into triangles and it is the dual of the 2-d Voronoi diagram, which partitions a plane into polygonal regions. These subdivisions of space operate in a set of points, called *seeds*. For every seed, the Voronoi diagram assigns a unique region consisting of all points that are closer (in Euclidian metric) to the given seed than to any other seed (identical to decision boundaries of the nearest neighbor classifier). These regions are

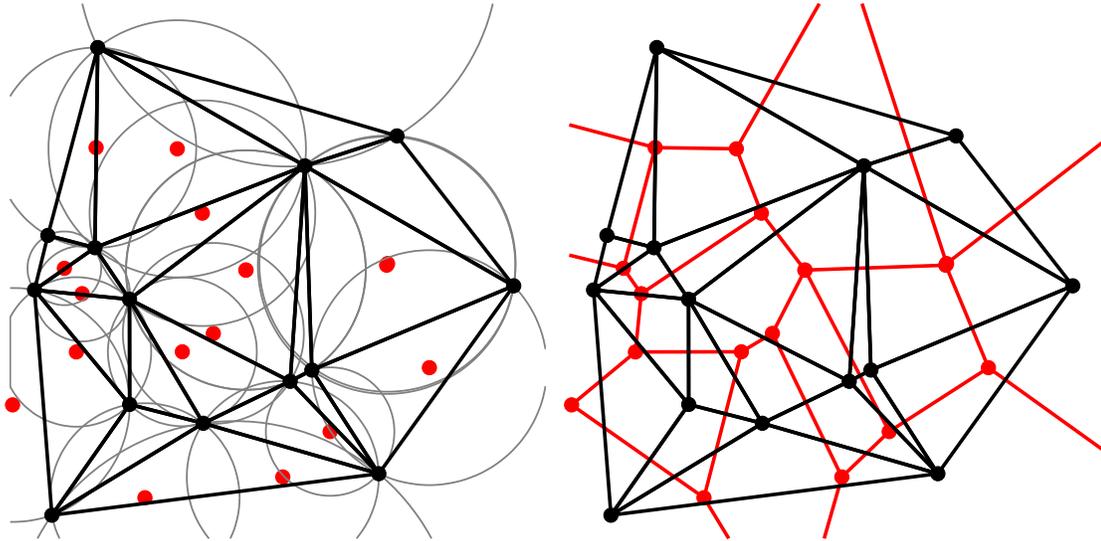


Figure 3.5: A Delaunay triangulation and its corresponding Voronoi diagram. Seed points are shown as black dots. *Left*: Delaunay triangulation shown in black with circumcircles in gray and their centers in red. *Right*: the corresponding Voronoi diagram shown in red produced by connecting centers of the circumcircles.

denoted Voronoi cells and each cell has a unique seed. The Delaunay triangulation is constructed by connecting the seeds in adjacent Voronoi cells. The Delaunay triangulation has unique properties when compared to other possible triangulations:

- The Delaunay triangulation maximizes the minimum angle of the triangles in the triangulation.
- The union of all triangles in the triangulation is the 2-d convex hull of the seeds.
- A circle circumscribing any Delaunay triangle contains no seed in its interior.
- The centers of Delaunay triangles circumcircles are the vertices of Voronoi cells.
- The Delaunay triangulation is unique assuming no degeneracies.
- Each seed has 6 surrounding triangles on average (for large number of seeds).

Delaunay triangulations and Voronoi diagrams are widely used in mathematics, computational geometry and other fields, *e.g.* in modeling terrains from a set of points or deriving capacities in wireless networks.

In this thesis, Delaunay triangulations are used to represent an array of points and their neighbors in chapters 4 and 5. The tasks from these chapters require connectivity graphs that supply neighbors

vertices in all directions and disfavor skinny triangles. The Delaunay triangulation is chosen since it discourages narrow triangles, due to its minimum angle maximization, and provides surrounding neighborhoods.

3.4 Camera estimation

A *camera* denotes a mathematical object that models an image sensor and how an image is formed via a projection of a 3-d scene into the plane of the image sensor. A camera is represented by a set of parameters that define the spatial position, spatial orientation and a set of intrinsic properties of an actual physical camera. The camera focal length, radial lens distortion, tangential lens distortion, the aspect ratio of image pixels, the geometric image center and the pose of the image sensor w.r.t. the lens are among common intrinsic camera parameters. The camera *extrinsic parameters* are the position and orientation of the camera, also denoted as *camera pose*, which define a coordinate system for the camera w.r.t. a reference frame called the *world coordinate frame*.

Camera calibration is the process of determining the camera intrinsic and extrinsic parameters. Common solutions to estimate intrinsic parameters, *e.g.* the one from Bouguet [14], use calibration objects. Some techniques denoted *self-calibration* use no calibration objects and rely on the rigidity of the scene to use only image information to calibrate cameras. The applications of this thesis use moving cameras where no information is known *a priori* about the camera parameters of each acquired image and where calibration objects are not practical, as in aerial views. Such cameras are called uncalibrated cameras. For such applications, self-calibration techniques are employed to calibrate cameras using an iterative nonlinear optimization called *bundle adjustment*.

3.4.1 Bundle adjustment

Bundle adjustment retrieves camera parameters and 3-d points in the scene from image correspondences. The reconstruction of the scene and camera geometry are up to a similarity transformation [76]. Common software packages to perform bundle adjustment are Bundler [105] and VisualSFM [130, 132]. Bundle adjustment consists of minimizing the *reprojection error*, which is the Euclidean distance between a projected point and a measured one. In general, the method reconstructs correspondences

in 3-d given camera parameters, forward projects the 3-d points back in the images, compute the errors of how far the projections are from the measured correspondences and finally refine the camera parameters to reduce the errors, then repeats. The outcome is globally optimized camera parameters, in addition to a reconstruction of sparse 3-d points.

3.4.2 Reprojection error

Bundle adjustment consists of estimating camera parameters to find a local minimum of the *reprojection error* of all image correspondences, which is the Euclidean distance measured in image pixels between a projected point $\hat{\mathbf{x}}$ predicted by image correspondences and the measured correspondences \mathbf{x} . The reprojection error quantifies how closely imperfect image correspondences \mathbf{x} represent the true projection of the 3-d point \mathbf{X} estimated from the correspondences. The projection of a 3-d point \mathbf{X} into a 2-d image coordinate is in general a nonlinear function of all the camera calibration parameters, where the linear part of the projection is encapsulated by the *projection matrix* [53] \mathbf{P} . Disregarding nonlinearities, the projections are given by

$$\mathbf{x} = \mathbf{P}\mathbf{X}. \quad (3.13)$$

The squared reprojection error \mathcal{R}^2 of a correspondence $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ is given by

$$\mathcal{R}^2 = \|\mathbf{x}_R - \hat{\mathbf{x}}_R\|^2 + \|\mathbf{x}_T - \hat{\mathbf{x}}_T\|^2, \quad (3.14)$$

where $\|\cdot\|$ represents Euclidean distance, and, $\hat{\mathbf{x}}_R$ and $\hat{\mathbf{x}}_T$ are location vectors obtained by projecting (via projections \mathbf{P}_R and \mathbf{P}_T) an associated 3-d point \mathbf{X} back into the images giving rise to $\hat{\mathbf{x}}_R = \mathbf{P}_R\mathbf{X}$ and $\hat{\mathbf{x}}_T = \mathbf{P}_T\mathbf{X}$. The point \mathbf{X} , which is the scene location observed at \mathbf{x}_R and \mathbf{x}_T , is computed by intersecting the 3-d rays emanating from respective camera centers and going through respective pixel locations \mathbf{x}_R and \mathbf{x}_T in a *reference* and *target* images. Since rays in three dimensions estimated from approximate pixel locations often do not exactly intersect, the best approximate ray intersection is defined as the 3-d point that minimizes the distances from itself to the rays. The mean squared reprojection error *MSE* of all correspondences of all images pairs of an image sequence is given by the average of the reprojection errors of the correspondences:

$$MSE = \frac{1}{N} \sum_{i=1}^N \mathcal{R}_i^2 \quad (3.15)$$

The MSE quantifies the deviation of the measured correspondence locations to the true locations. Bundle adjustment minimization simultaneously refines camera parameters and the 3-d coordinates representing scene geometry giving rise to the associated predicted image coordinates, which are preferably near observed coordinates. The smaller the MSE is, the more accurate the estimation of camera parameters is expected to be.

Chapter 4

Near Real-time Camera Pose Estimation

Camera estimation is the first step in an image-based surface reconstruction pipeline to allow incoming images to be registered and analyzed together. The most general case is addressed, where no information is known *a priori* about the cameras, except that inter-frame camera motion is small. Image correspondences are required to estimate cameras. This chapter proposes a novel set of matching tools using normalized cross-correlation that achieves **higher accuracy** than SIFT matching in terms of reprojection error of 3-d points computed from estimated matches and cameras. In addition, the number of correspondences found by the proposed system is higher than the ones from SIFT matching. The estimated camera parameters are optimized by publicly available multi-core bundle adjustment software [132]. The proposed implementation is partially GPU-accelerated and reaches near real-time performance, but if totally optimized, it should achieve similar computational throughput to real-time state-of-the-art SIFT features detection and matching running on a GPU. The term *real-time* used here refers to a system capable of matching images as quickly as they are acquired.

4.1 Matching overview

The matching process automatically assigns correspondences to points of interest seen in images that are processed one pair at a time. Assume the images are from a video sequence and taken from a moving camera having small baselines between adjacent frames. The image sequence may be an indefinitely large stream, hence consider a sliding window \mathcal{W} of size R_f adjacent frames. Within \mathcal{W} , the narrowest baseline pairs are matched first, followed by increasing baselines. Interest points are selected in the views from a given \mathcal{W} and matched to their subsequent views. After all pairs in a window \mathcal{W} are matched, it slides one image forward in the timeline and the process repeats. For instance, if $R_f = 4$, the matching order for $\mathcal{W} = \{1, 2, 3, 4\}$ is $\{1-2, 2-3, 3-4, 1-3, 2-4, 1-4\}$, then the window shifts to $\mathcal{W} = \{2, 3, 4, 5\}$ and unmatched pairs are processed analogously. The narrowest baselines are matched first since their matching is easier and likely provides enough correspondences to estimate accurate fundamental matrices, which are essential for match disambiguation. Growing baselines are increasingly harder to match, but matching becomes easier if a fundamental matrix is given prior to matching. The matrix for wider baseline frames is estimated from the sequence of narrower baseline matches using method discussed in section 4.4.8.

The matching is performed via normalized cross-correlation (NCC). The matching pipeline for an image sequence is summarized in flowchart in figure 4.2. For every image pair in all \mathcal{W} , feature correspondences are assigned through the *matching process* subroutine. After all correspondences are computed among the views, bundle adjustment [118] is performed by feeding matches into the optimization subroutine of the VisualSFM application [132].

The matching process subroutine is specified in figure 4.3 flowchart. Interest points are detected in one image of a pair, I_R , and a search for their corresponding points is carried out on the second image, I_T . These images are denoted *reference* and *target*, respectively. In addition, *auxiliary* images, I_A , which have been matched to I_R in advance, will be used to check for geometric consistency of matches between I_R and I_T . The scope of an image as being a reference, a target or an auxiliary view changes as the sliding window moves (see figure 4.1).

Often multiple matching candidates arise. In order to disambiguate the matching, several tests are performed, *e.g.*, to enforce geometric relations and spatial consistency of matches. The *disambiguation*

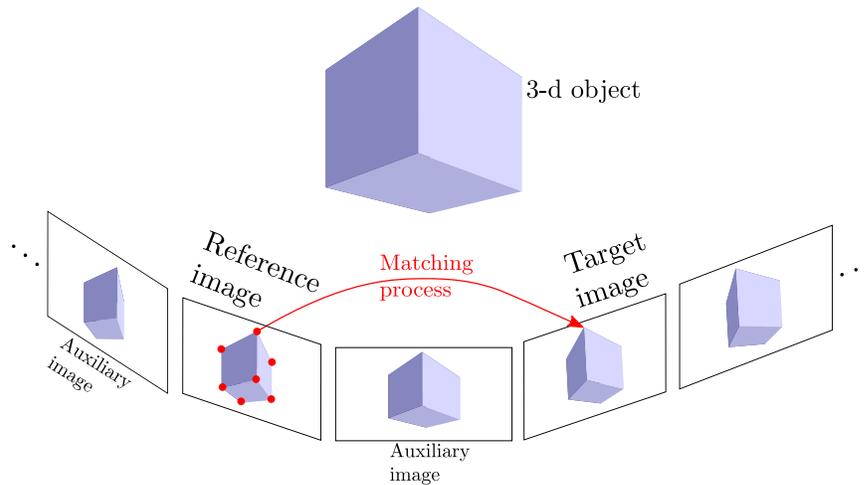


Figure 4.1: Feature matching process and the scope of images taken from a sliding window moving through adjacent frames of a sequence. The matching process establishes image point corresponds in the *target image* for interest points located in the *reference image*. *Auxiliary images* are nearby views used to assist on match disambiguation when multiple match candidates exist. The scope of an image changes as the matching process window moves. Auxiliary images are matched to the reference image in advance, in which case, they were reference and target, respectively.

process subroutine encapsulates these tests and is summarized as a flowchart in figure 4.4.

This chapter will discuss in detail the proposed feature matching pipeline from figure 4.3: detection of interest points (section 4.2), matching using normalized cross-correlation (section 4.3) and a series of multiple matches disambiguation steps (section 4.4).

Tables 4.1 and 4.2 provide a quick reference for the notation and parameters appearing in this chapter along with default parameter values. A method description may specify the values of its parameters, otherwise default values from table 4.2 are used.

4.1.1 Displaying image matches

A common way to display image matches is to display two images side-by-side and connect the matching points with a straight line. An alternative is to display the matches with a distinctive unique marker. Connecting lines are unambiguous, but can become very cluttered for thousands of matches. The markers are chosen for this thesis to prevent clutter.

Matching points are shown with identical markers sharing the same shape, width and color, which are picked randomly from predefined sets. The randomness does not guarantee uniqueness of the

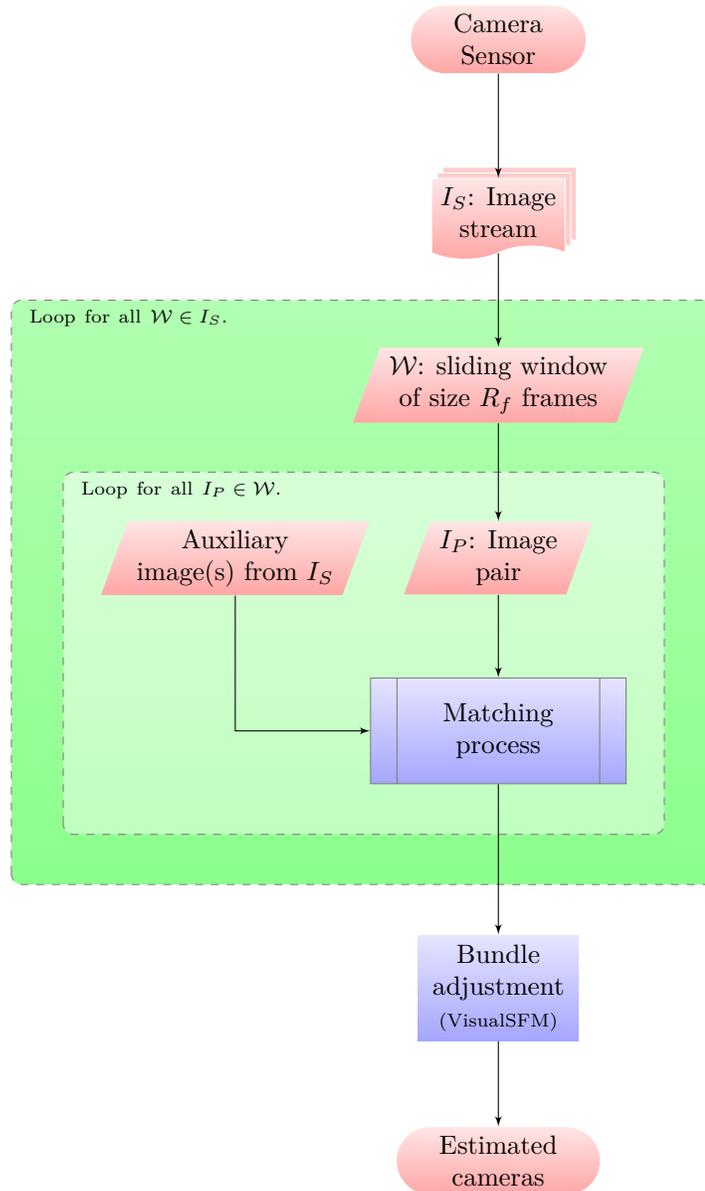


Figure 4.2: Matching pipeline for an image stream. See figure 4.3 for “Matching process” subroutine.

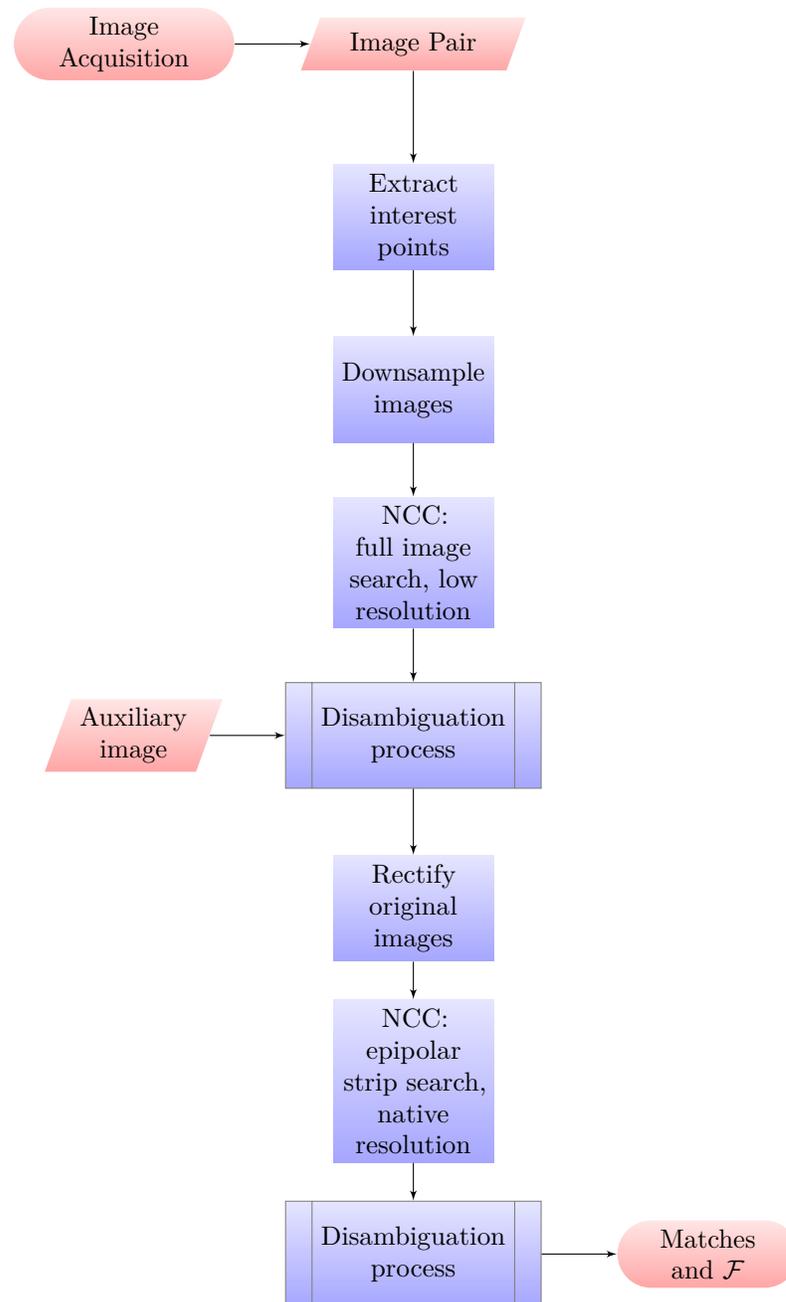


Figure 4.3: Matching process for an image pair. NCC stands for normalized cross-correlation matching. See figure 4.4 for “Disambiguation process” subroutine.

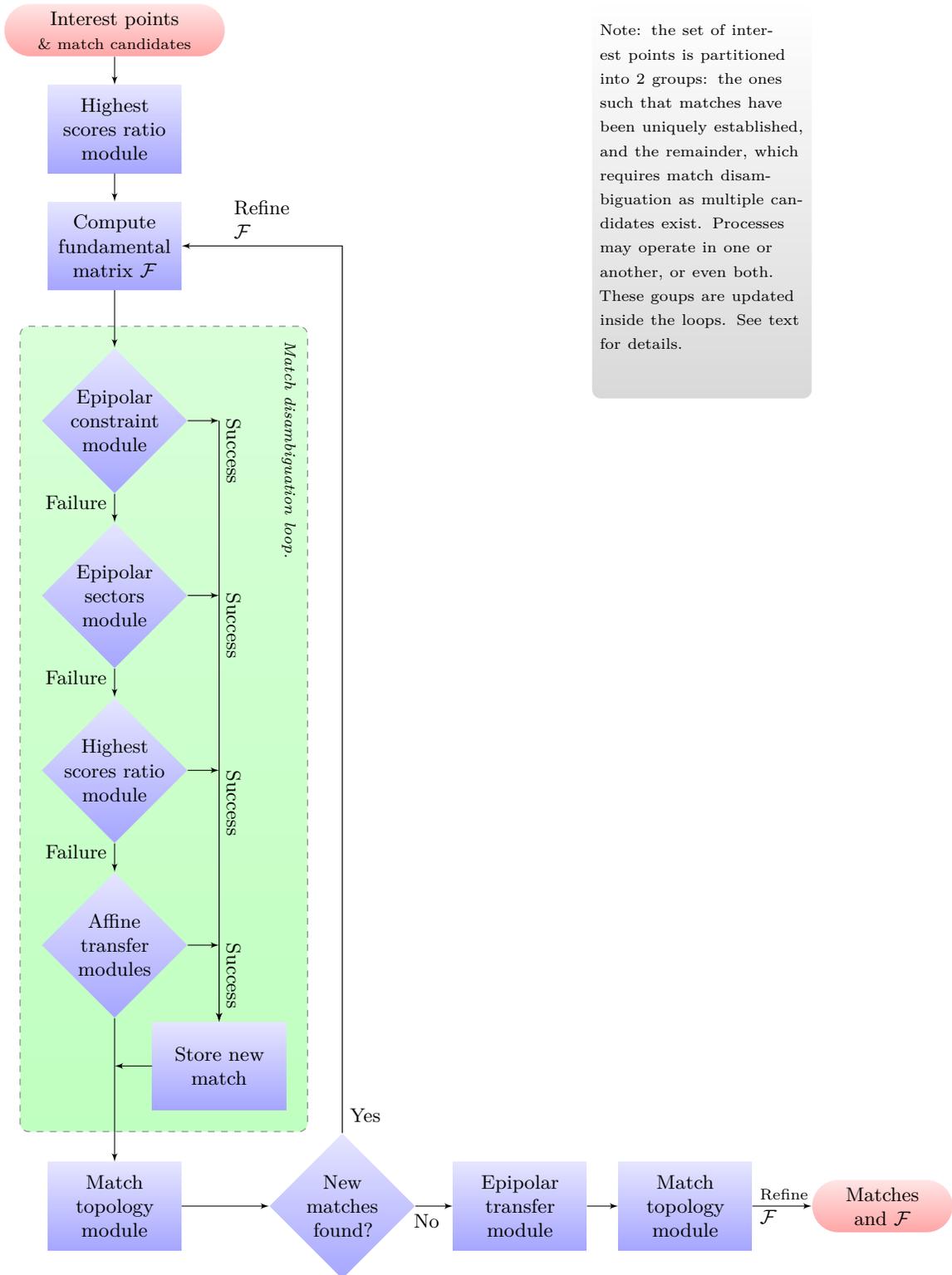


Figure 4.4: Disambiguation process block from figure 4.3.

Notation	Description
\mathcal{W}	Sliding window of adjacent frames.
I_R	A reference image, a source of interest points to be matched.
I_T	A target image, where a search for matching features is performed.
I_A	An auxiliary image, used to test match geometric consistency.
$S(x, y)$	A rectangular correlation search area in I_T .
$C(u, v)$	A rectangular array of correlation coefficients.
\mathbf{U}_m	The subset of interest points with unique estimated matches.
\mathbf{M}_m	The subset of interest points with multiple match candidates (\mathbf{U}_m^c).
\mathbf{x}_R	An interest point location in I_R .
\mathbf{x}_T	A putative matching location of \mathbf{x}_R in I_T .
\mathbf{x}_A	A putative matching location of \mathbf{x}_R in I_A .
$\mathbf{x}_R \leftrightarrow \mathbf{x}_T$	A correspondence between I_R and I_T .
$\mathbf{x}_R \leftrightarrow \mathbf{x}_T \leftrightarrow \mathbf{x}_A$	A correspondence triplet between I_R, I_T, I_A , respectively.

Table 4.1: Notation for proposed feature matching pipeline.

Parameter	Default value	Description
R_f	4	Number of frames in \mathcal{W} .
κ	0.04	Sensitivity parameter for Harris-Stephens corner detection.
Q_c	0.01	Quality level for corner detection.
t_a	5 pixels	Radius of square template patch (11×11 patch).
Q_m	0.85	Quality level for correlation-based matching.
C_h	21 pixels	Height of correlation search area on rectified images.
C_w	width(I_T)	Width of correlation search area on rectified images.
τ_c	5	Highest ratio of matching points corner scores.
τ_{hsr}	1.1	Lowest correlation ratio for highest scores ratio module.
τ_{ratio}	1.5	Lowest ratio of lowest distances for ratio-inequalities test.
τ_{global}	2.0	Distance threshold for ratio-inequalities test.
τ_f	0.2	Sampson distance threshold for fundamental matrix estimation.
c_h	7 pixels	Height of correlation window for epipolar transfer module.
c_w	31 pixels	Width of correlation window for epipolar transfer module.
n_m	1	Threshold for match topology module.

Table 4.2: List of proposed feature matching pipeline parameters.

markers allowing occasional ambiguities that are fairly easy to discern. See figure 4.5 for an example of a pairwise matching displayed with random markers and obtained with the proposed method of this chapter. Figure 4.6 shows details of figure 4.5.

4.2 Interest points

In order to estimate camera geometry for images in a dataset, a number of features must be tracked from frame to frame. This section describes the proposed method to find interesting features for further matching using normalized cross-correlation.

Corner detection. Interest points in an image are locations that have expressive texture and are easy to match on other images due to high distinctiveness, unlike textureless locations. It is desired that a matching descriptor of an interest point is invariant under changes in illumination and viewpoint. Common interest point detectors used in the literature are the ones from Harris and Stephens [52], Shi and Tomasi [104] and Lowe [74].

In this thesis, point tracking will be carried along frames where the inter-frame motion is smooth and not very large, *i.e.*, parallax due to viewpoint changes may be noticeable, but scale of features may be considered constant and camera motion is dominated by translation. The Shi-Tomasi algorithm is slightly slower than Harris-Stephens algorithm, but tends to detect faint corners better and proved adequate for region matching over longer time spans, which corresponds to wider baselines and more parallax. Thus, a variation of the Shi-Tomasi algorithm is chosen as corner detector for the proposed matching system since the target motion model expects some parallax. Lowe has a robust detector that finds keypoints over scale and space. SIFT keypoints and SIFT matching are used for evaluation and comparison in chapter 8.

Since the Shi-Tomasi detector is more sensitive than the Harris-Stephens detector, it tends to find more spurious corners that are in fact edges (unidirectional texture patterns). These responses are in fact outliers and are difficult to accurately match in practice. Unlike the Shi-Tomasi detector, the Harris-Stephens detector is a combined corner and edge detector that can differentiate between edges and corners in its signed metric. In order to eliminate the spurious edge responses from the

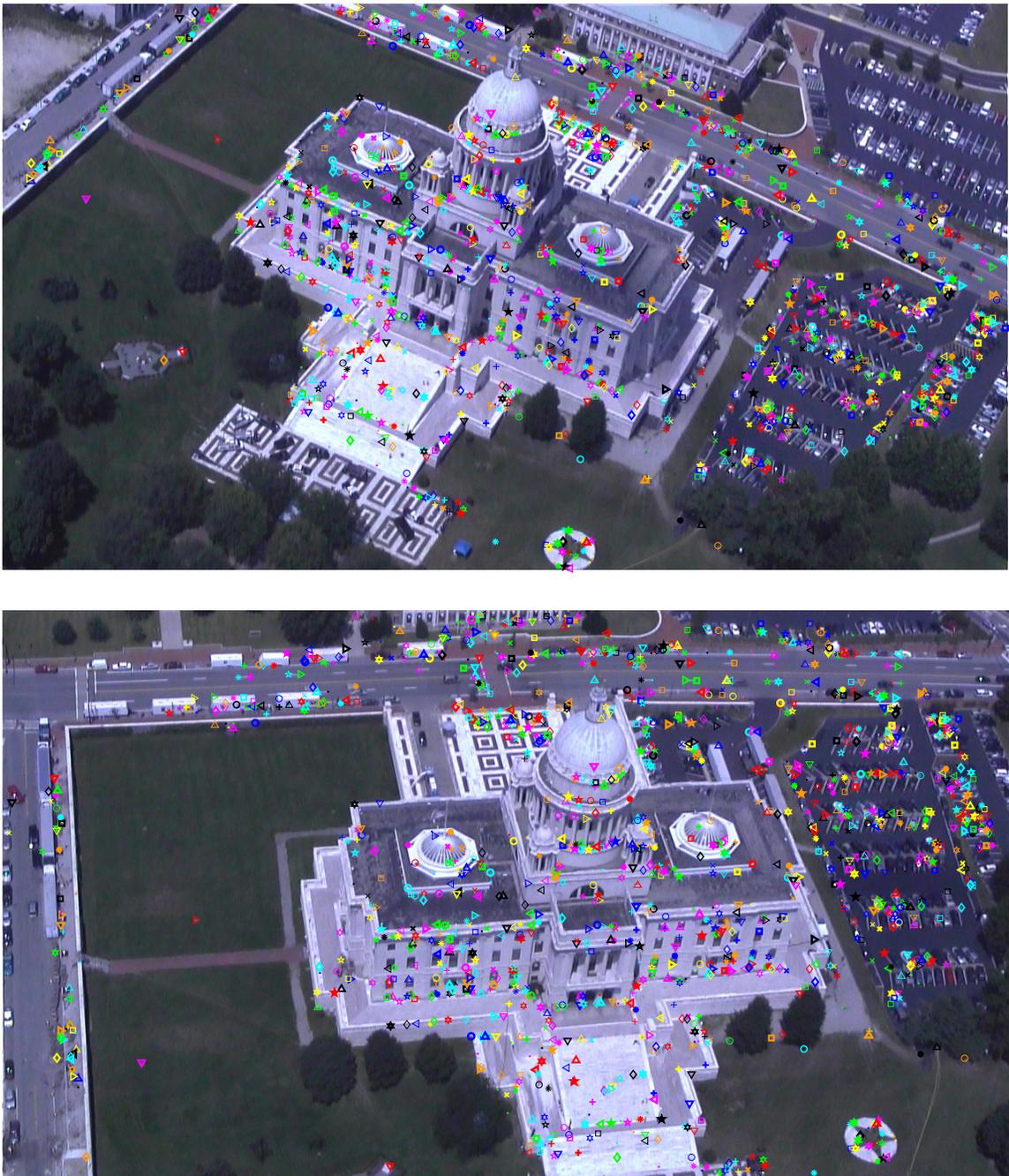


Figure 4.5: Typical example of the pairwise matching method proposed in this chapter. Matching points are shown with the same marker shape, width and color, which are picked randomly. Marker shapes can be dots, circles, stars, crosses, squares, triangles *etc.* Colors used are blue, red, green, yellow, black, among others. Occasionally, different matches may have an identical random marker due to small finite number of distinct possible markers. However, ambiguities are often not confusing.



Figure 4.6: Detail of the same example as figure 4.5 with estimated matches shown using only small dot markers, which provide better visualization of match accuracy.

Shi-Tomasi detector while keeping the interesting faint corners it detects, the two detectors are combined.

The two detectors and the proposed combined detector are described here for comparison. The Harris-Stephens detector is based on a *structure tensor* S_T (per pixel) in which elements are local averages of a function of image gradients:

$$S_t = \sum_u \sum_v w(u, v) \begin{bmatrix} (\frac{\partial I}{\partial x})^2 & (\frac{\partial I}{\partial x} \frac{\partial I}{\partial y})^2 \\ (\frac{\partial I}{\partial x} \frac{\partial I}{\partial y})^2 & (\frac{\partial I}{\partial y})^2 \end{bmatrix} \quad (4.1)$$

where $w(u, v)$ is a rotationally symmetric Gaussian averaging function. The size of the two eigenvalues of the tensor matrix λ_1, λ_2 (which can be ordered $\lambda_1 \geq \lambda_2 \geq 0$), indicate whether the intensity in the pixel is varying in more than one direction:

- Case 1:** no features are found if $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$,
- Case 2:** an edge is found if $\lambda_2 \approx 0$ and λ_1 is large and positive,
- Case 3:** a corner is found if both λ_1 and λ_2 are large and positive.

Given that the computation per pixel of the eigenvalues of the structure tensor S_T is expensive, *cornerness* functions were designed to avoid the eigenvalues computation directly:

$$M_{\text{Harris-Stephens}} = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 = \det(S_T) - \kappa \text{trace}^2(S_T), \quad (4.2)$$

$$M_{\text{Shi-Tomasi}} = \min(\lambda_1, \lambda_2) = \frac{\text{trace}(S_T) - \sqrt{\text{trace}^2(S_T) - 4 \det(S_T)}}{2}, \quad (4.3)$$

where κ is a tunable sensitivity parameter, here set to $\kappa = 0.04$. Negative scores for $M_{\text{Harris-Stephens}}$ indicate edges, small magnitudes suggest featureless regions, and larger positive values suggest corners. Large positive values also suggest corners for $M_{\text{Shi-Tomasi}}$, which otherwise display small nonnegative magnitudes at noncorner regions.

Harris-Stephens corners are defined as *local maxima* (using 8 nearest neighbors) of the metric in equation (4.2) computed over the image space that satisfy the minimum quality condition in equations (4.4) and (4.5). Analogously, Shi-Tomasi corners are defined in terms of the local maxima

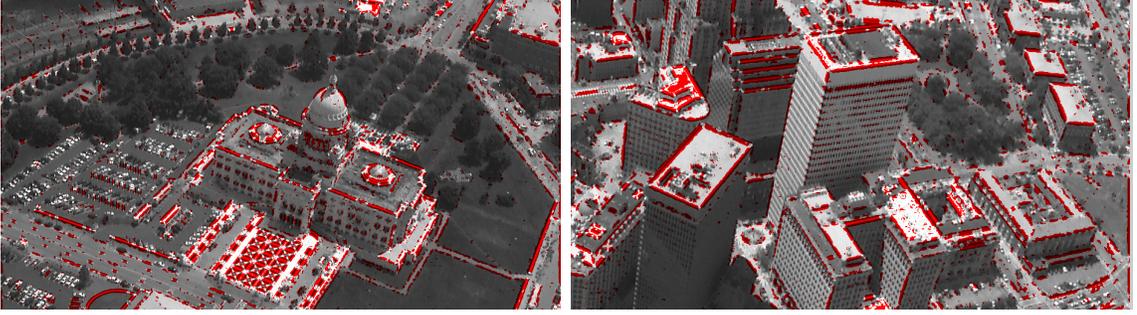


Figure 4.7: Edge responses on two aerial images of urban scenes. The red regions are the locations where the Harris-Stephens detector has negative cornerness value, indicating the likelihood of an edge. The proposed interest point detector rejects corners on red regions according to equation (4.8).

of equation (4.3) and conditions in equations (4.6) and (4.7):

$$M_{\text{Harris-Stephens}}(x, y) > \tau_{HS}, \quad (4.4)$$

$$\tau_{HS} = Q_c \cdot \max_{x,y} M_{\text{Harris-Stephens}}(x, y), \quad (4.5)$$

$$M_{\text{Shi-Tomasi}}(x, y) > \tau_{ST}, \quad (4.6)$$

$$\tau_{ST} = Q_c \cdot \max_{x,y} M_{\text{Shi-Tomasi}}(x, y). \quad (4.7)$$

where Q_c is scalar *quality level* parameter $0 < Q_c < 1$ specifying a minimum score for corners with default value $Q_c = 0.01$.

Regarding the combined detector, (x, y) is an interest point if it is a Shi-Tomasi corner and its Harris-Stephens cornerness measure satisfy the following edge rejection condition:

$$M_{\text{Harris-Stephens}}(x, y) > 0. \quad (4.8)$$

Therefore, the combined detector is in fact a Shi-Tomasi corner detector that uses the Harris-Stephens metric to eliminate edge responses, as illustrated in figure 4.7. The combined detector metric M_{STHS} is given by

$$M_{\text{STHS}}(x, y) = \begin{cases} M_{\text{Shi-Tomasi}}(x, y) & : M_{\text{Harris-Stephens}}(x, y) > 0 \\ 0 & : M_{\text{Harris-Stephens}}(x, y) \leq 0 \end{cases}, \quad (4.9)$$

and the interest points are every (x, y) such that

$$M_{\text{Shi-Tomasi}}(x, y) \text{ is a local maximum,} \quad (4.10)$$

$$M_{\text{Harris-Stephens}}(x, y) > 0. \quad (4.11)$$

Figure 4.8 provides a comparison of the three described corner detectors. There are a total of 1749 Harris-Stephens corners (green circles), and 5062 Shi-Tomasi corners, from which 179 are considered outliers (larger red dots) and the remaining 4883 are the result of the combined detector (blue dots). Some corners in figure 4.8 appear as edges at the displayed resolution, however they are true weak texture corners noticeable only in detail. This is a desirable result as such weak corners can be tracked, and when some are in fact edges, they are often ignored at the matching stage. Edges are ambiguous to track as they can match anywhere along themselves, causing the proposed matcher to often fail to estimate a single matching location for them.

4.3 Matching process

Given a set of interest points found in an image, normalized cross-correlation is the matching tool used to find their matches in a target view.

The matching paradigm used here is different from the one in SIFT matching [74]. In [74], a database of interest points is computed independently for each image and their descriptors are compared to find a match. In this chapter, an interest point is a corner and its match can be anywhere in a search region in I_T assigned to the corner. Normalized cross-correlation provides a similarity score for each interest point in I_R and all possible locations inside its own search region. No corners are estimated in I_T .

The normalized cross-correlation similarity measure is not salient enough to pinpoint a true match by simply locating a global maximum, but empirically the true match lies at a local maximum. Section 4.3.1 describes how to choose potential matches of a point as local peaks of $C(u, v)$.

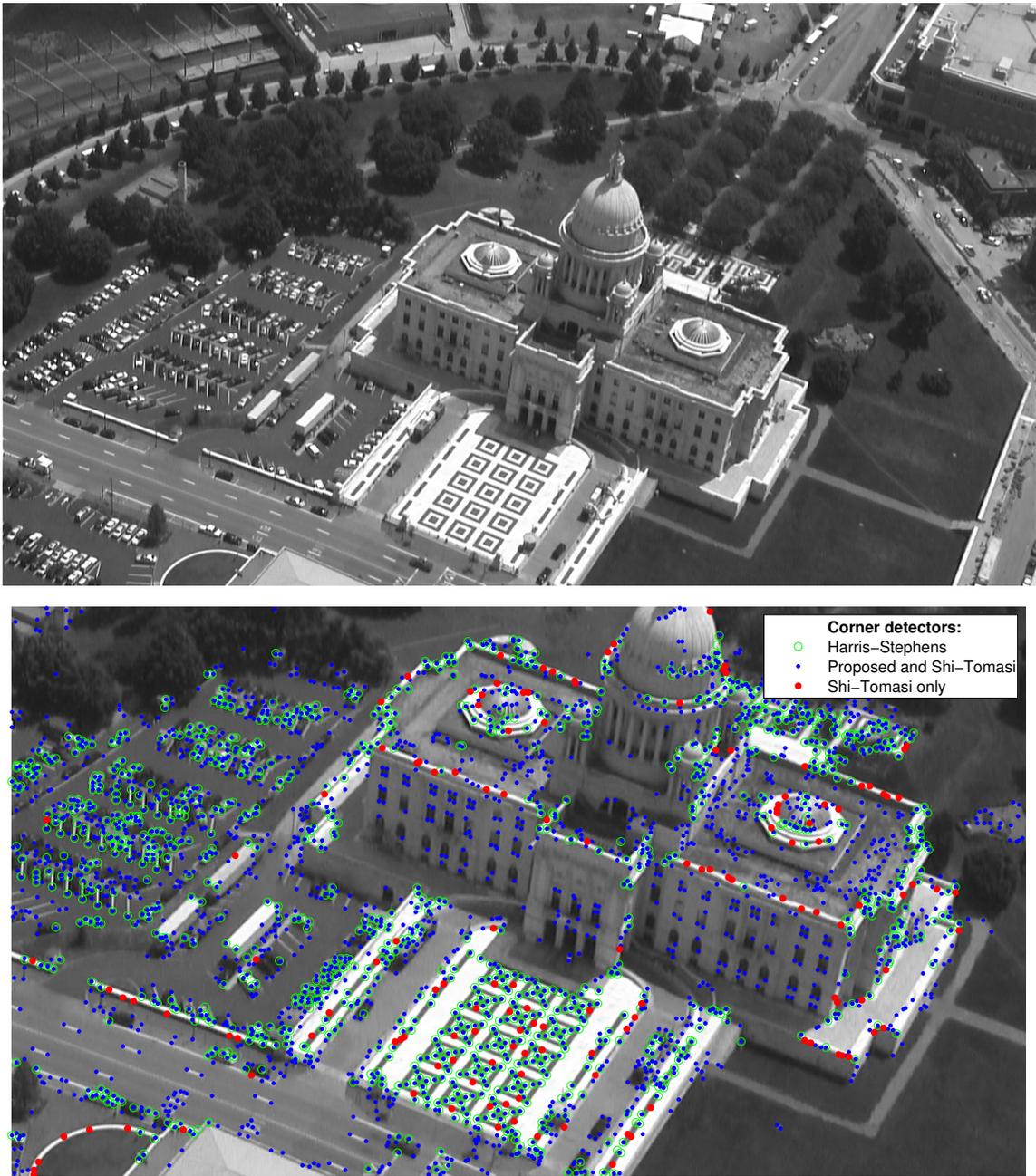


Figure 4.8: Interest point detection. *Top*: an aerial image. *Bottom*: detected interest points from corner detectors. Harris-Stephens corners are shown as circles and Shi-Tomasi corners are shown as “dots” (blue or red). Larger dots, displayed in red, are Shi-Tomasi corners that the proposed detector rejects as spurious edge responses using its combined metric from Harris-Stephens and Shi-Tomasi detectors. The remaining dots, in blue, are the combined detector responses and include legitimate corners the Harris-Stephens detector does not find, *e.g.*, the ones at the windows of the building and road surface markings.

4.3.1 Correlation peaks

This section is implemented on a GPU (see chapter 7 for details). For each interest point $\mathbf{x}_R = (x, y)$ in a reference image I_R , a search for a match in the target image I_T is carried out using normalized cross-correlation, as defined in equation (3.7) with uniform weights.

The template $t(x, y)$ is a square centered at \mathbf{x}_R with side $t_s = 2t_a + 1$, where $t_a > 0$ is the *template apothem*, an integer template radius. The default apothem of 5 pixels yields a template size of 11×11 , which empirically provides a good trade-off between the bias in large sizes and the variance in small ones, as discussed in section 3.2. The *search window* is a rectangular subimage $S(x, y)$ of I_T presumably containing the matching feature of \mathbf{x}_R . $S(x, y)$ could be at any location and be of any shape. Two rectangular shapes are considered:

- **Full image.** If no prior knowledge of the scene is known, $S(x, y)$ is the entire target image.
- **Epipolar strip.** If a fundamental matrix is known, according to epipolar geometry, $S(x, y)$ is reduced to a strip around epipolar lines (group of adjacent rows after rectification). The strip height is thin, but it is as wide as the rectified image I_T .

A correlation coefficient is computed for each location aligned with the pixel grid where the template fits inside $S(x, y)$. The result is a 2-d array of correlation coefficients $C(u, v)$ of size $C_u \times C_v$, denoted as correlation *scores*. Note, interest points lying too close to the reference image borders are discarded if a template patch centered on them does not fit inside the image. Moreover, the size of $S(x, y)$ is controlled by the desired size of the correlation window $C(u, v)$ and the size of the template $t(x, y)$ (see section 3.2.1).

In the typical image matching scenario, corresponding features are observations of a 3-d point from distinct viewpoints. Ideally, the observations would be identical and their correlation score would be $C = 1$. In practice, they are not, but the matching location is a local maximum of the correlation array with peak score $C \leq 1$. Moreover, often a peak in $C(u, v)$ with the highest score

$$\gamma_{max} = \max_{i,j} C(i, j) \tag{4.12}$$

is not the true match and any local maximum could be a potential match candidate. Furthermore, the highest peak (and the true match location) may sometimes present a relatively low correlation score

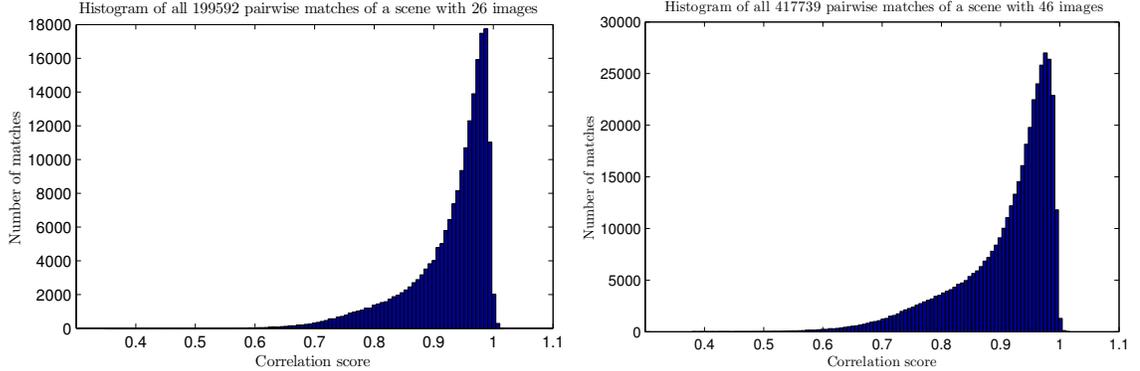


Figure 4.9: Distributions of the correlation scores of all estimated matches of an aerial urban scene.

due to viewpoint changes, quantization and noise. Figure 4.9 shows the distributions of correlation scores of uniquely estimated matches of two scenes, one with 26 images and the other with 46. The matches are estimated and disambiguated by the framework of this chapter. Both scenes are aerial views or urban areas and were matched using a window of $R_f = 4$ adjacent frames. The distributions suggest that the majority of matches present correlation scores from 0.7 to 1.0 in one scene, whereas the range for the other is 0.6 to 1.0. Note, true match scores tend to decrease with increasing baselines and are affected by partial occlusions.

In order to detect matches in all quality ranges, matches of an interest point are pruned according to the best score of all peaks. Analogously to the pruning of detected corners in equation (4.5), a scalar quality level parameter $0 \leq Q_m \leq 1$ specifies a minimum peak score. Peaks with correlation scores $C(\mathbf{u})$, $\mathbf{u} = (u, v)$, such that

$$C(\mathbf{u}) \geq \gamma_{max} Q_m \quad (4.13)$$

are preserved, others are discarded. Note, allowing low score peaks through equation (4.13) may include false matches that will be discarded at later stages of validation and disambiguation.

A peak location in $C(\mathbf{u})$ is associated with an image coordinate in $I_T(\mathbf{x}_T)$, $\mathbf{x}_T = (x', y')$, using the natural change of coordinates

$$\mathbf{x}_T = m(\mathbf{u}) \quad (4.14)$$

from the domain of C to the domain of I_T . $m(\cdot)$ is defined as in figure 4.10 in terms of the relative positioning of the correlation windows.

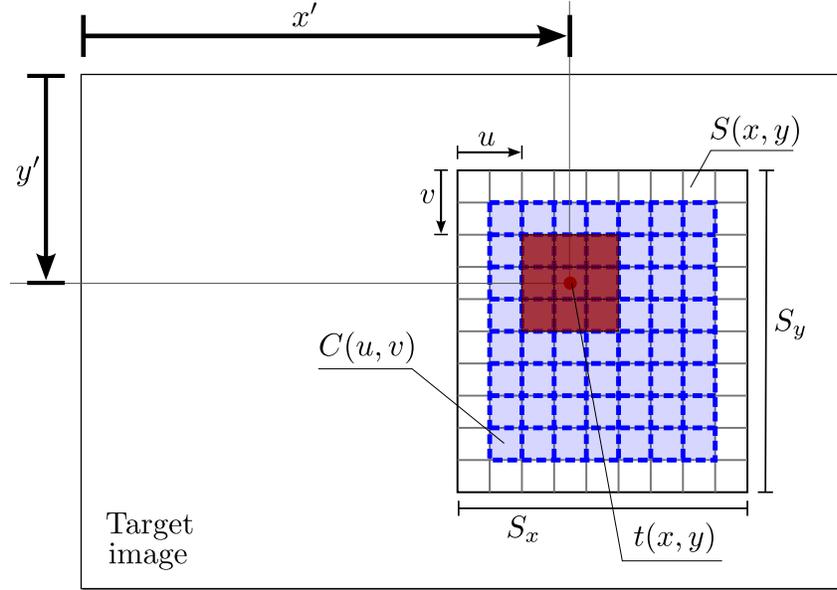


Figure 4.10: Domains of correlation windows. t is a small subimage of I_R , S is a subimage of I_T and C is the resulting correlation window. The location (u, v) in the domain of C is associated with the underlying location (x', y') in I_T if $C(u, v)$ was computed with t centered at (x', y') . This means the correlation coefficient between the patch $t(x, y)$ and the patch around $I_T(x', y')$ is $C(u, v)$, and, the natural change of coordinates $m(\cdot)$ between domains is $m(u, v) = (x', y')$.

For a given interest point \mathbf{x}_R in I_R and its correlation array $C(\mathbf{u})$, let $h_p(\mathbf{x}_R)$ be the set of all remaining correlation peaks of $C(\mathbf{u})$ that satisfy equation (4.13):

$$h_p(\mathbf{x}_R) = \{\mathbf{u} \mid C(\mathbf{u}) \geq \gamma_{max} Q_m \text{ is a peak of } C(\mathbf{u})\}. \quad (4.15)$$

Hence, $h_p(\mathbf{x}_R)$ includes the highest peak location and possibly other peaks with high confidence scores according to Q_m and γ_{max} . However, the peaks are sampled at the pixel grid. A refinement is required for subpixel accuracy and it is preformed using a bivariate quadratic function $b_{\hat{\mathbf{u}}}$ fit to the local 3×3 pixel neighborhood of each peak $\hat{\mathbf{u}} \in h_p(\mathbf{x}_R)$. The bivariate (two variable) quadratic function is a second-degree polynomial of the form

$$b(\mathbf{u}) = Au^2 + Bv^2 + Cu + Dv + Euv + F. \quad (4.16)$$

The subpixel refinement updates a peak location $\hat{\mathbf{u}}$ and its value $C(\hat{\mathbf{u}})$ to the ones from the peak of

$b_{\hat{\mathbf{u}}}$, as follows:

$$\hat{\mathbf{u}} \mapsto \hat{\mathbf{u}}' = \arg \max b_{\hat{\mathbf{u}}}, \quad (4.17)$$

$$C(\hat{\mathbf{u}}) \mapsto C(\hat{\mathbf{u}}') = \max b_{\hat{\mathbf{u}}}. \quad (4.18)$$

With this notation, let $r_p(\mathbf{x}_R)$ be the set of refined peaks of \mathbf{x}_R , where the crude peaks were taken from $h_p(\mathbf{x}_R)$:

$$r_p(\mathbf{x}_R) = \{\mathbf{x}_T = m(\mathbf{u}') \mid \exists \mathbf{u} \in h_p(\mathbf{x}_R) \text{ such that } \mathbf{u} \mapsto \mathbf{u}'\}. \quad (4.19)$$

Here, $\mathbf{x}_T = m(\mathbf{u}')$ is the change of coordinates presented in equation (4.14). Consequently, $r_p(\mathbf{x}_R)$ stores refined match locations in I_T for an interest point \mathbf{x}_R in I_R , while $h_p(\mathbf{x}_R)$ was storing integer peak locations from $C(u, v)$.

4.3.2 Weak corner removal

Intensity fluctuations in homogeneous patches are mainly dominated by noise and normalized cross-correlation enhances such noise of homogeneous regions by normalizing the patch signal subtracting its mean and dividing by a small standard deviation. As a result, it is possible a noticeable corner becomes highly correlated with homogeneous points by coincidental resemblance of the normalized intensities, as shown in figure 4.11. Such matching points have weak corner scores since they are in fact not even near corners. In order to avoid these false matches, every match candidate $\mathbf{x}_T \in r_p(\mathbf{x}_R)$ is required to have similar corner scores to \mathbf{x}_R . Match candidates \mathbf{x}_T are rejected if

$$R_M(\mathbf{x}_R, \mathbf{x}_T) = \frac{M_{\text{STHS}}(\mathbf{x}_R)}{M_{\text{STHS}}(\mathbf{x}_T)} > \tau_c \quad (4.20)$$

where M_{STHS} is the corner metric defined in equation (4.9). Note, no corners are detected in the target image. Detected corners are interest points in the reference image and normalized cross-correlation is used as the matching tool. The matched points are not necessarily corners, which would be located at metric peaks (see section 4.2). As the corner metric can be computed anywhere, its values at matched points are analyzed in equation (4.20) for pruning purposes only, rejecting matches that have corner metric value very low in comparison to its match. Let $p_m(\mathbf{x}_R)$ be a subset of $r_p(\mathbf{x}_R)$

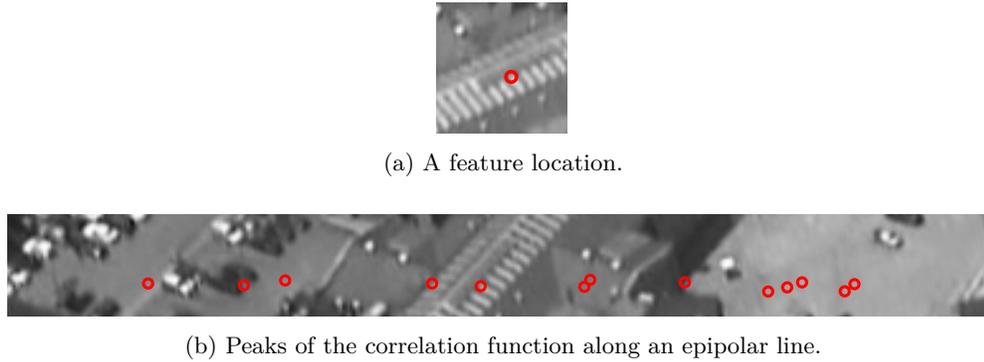


Figure 4.11: Illustration of normalized cross-correlation peaks at homogeneous locations. (a) A feature location that is a strong corner of the edge of a white stripe. (b) Correlation peaks associated with the feature are presented in a new viewpoint. While the true correspondence is at the center and other plausible matches with similar texture are found, many others located in homogeneous regions are clearly erroneous and are discarded based on relative corner scores.

that successfully pass equation (4.20) test:

$$p_m(\mathbf{x}_R) = \{\mathbf{x}_T \in r_p(\mathbf{x}_R) \mid R_M(\mathbf{x}_R, \mathbf{x}_T) \leq \tau_c\}. \quad (4.21)$$

Therefore, $p_m(\mathbf{x}_R)$ is the set of all putative matching feature locations established in I_T for an interest point \mathbf{x}_R from I_R .

4.3.3 Collection of sets of match candidates

Denote $M_c(I_R)$ the collection of all putative matches sets $p_m(\mathbf{x}_R)$ for every interest point \mathbf{x}_R detected in I_R :

$$M_c(I_R) = \{p_m(\mathbf{x}_R) \mid \mathbf{x}_R \text{ is an interest point of } I_R\}. \quad (4.22)$$

The sets have potentially multiple matches. Figure 4.12 illustrates the distribution of the cardinality of these sets for a given image pair and the two types of investigated search windows: *full image* and *epipolar strip* (see section 4.3.1). Figure 4.12 shows the benefits of searching for peaks around epipolar lines versus the entire image. The experiment is in native image resolution. Interest points are colored according to the number of putative matches (cardinality of elements of M_c). Interest points such that their true match is not visible in the target image are expected to have no peaks. A full image search presents high-level of match ambiguity given very few points are unambiguous

(single peak), and essentially every point has peaks, except the ones unprocessed due to proximity to an image border where the template patch does not fit. The epipolar strip search not only is faster, but it presents a higher quality result, *e.g.*, in the distribution on the bottom-right corner of figure 4.12, the majority of matches are unambiguous. In addition, most points that have no peaks do not actually have a true matching point visible in I_T .

In conclusion, very distinct points will have a single match candidate under a restricted window, but multiple potential matches may very often exist for others. This suggests that disambiguation processes are crucial. Section 4.4 describes the use of several geometric relations and confidence tests to unveil a reliable match estimate when there are multiple choices. In addition, a fundamental matrix \mathcal{F} must be available in order to search for matches around epipolar lines. \mathcal{F} can be computed either from estimated cameras or matches, however no information is known ahead of time. This problem is solved with an initial low-resolution matching step discussed in detail in section 4.3.4.

4.3.4 Initial low-resolution matching

According to discussion at the end of section 4.3.3, can the full image search be avoided? In the sense of finding the fundamental matrix without any prior knowledge of the scene, it is unavoidable, but it can be expedited by preprocessing downsampled images. Since no prior information is known, a low-resolution full image search is first performed to find match collections, followed by disambiguation (section 4.4). This search and disambiguation is actually the complete matching pipeline of this chapter running on downsampled images. The only matched points are low-resolution features, which are regularly not ambiguous as the ones in high spatial frequencies. Then, the unambiguous matches are used to estimate a fundamental matrix \mathcal{F} to drive the iterative disambiguation process, which finds more matches and refines \mathcal{F} . Since the images are smaller, the additional processing time is negligible compared to the time of a native resolution full image search (roughly 20-100X slower, depending on data and GPU architecture).

In summary, low-resolution matching is performed to estimate \mathcal{F} and avoid inefficient full image search at native resolution. The disambiguation (section 4.4) increases the number of matches, which augments the accuracy of \mathcal{F} to allow an efficient epipolar strip search with a thin strip.

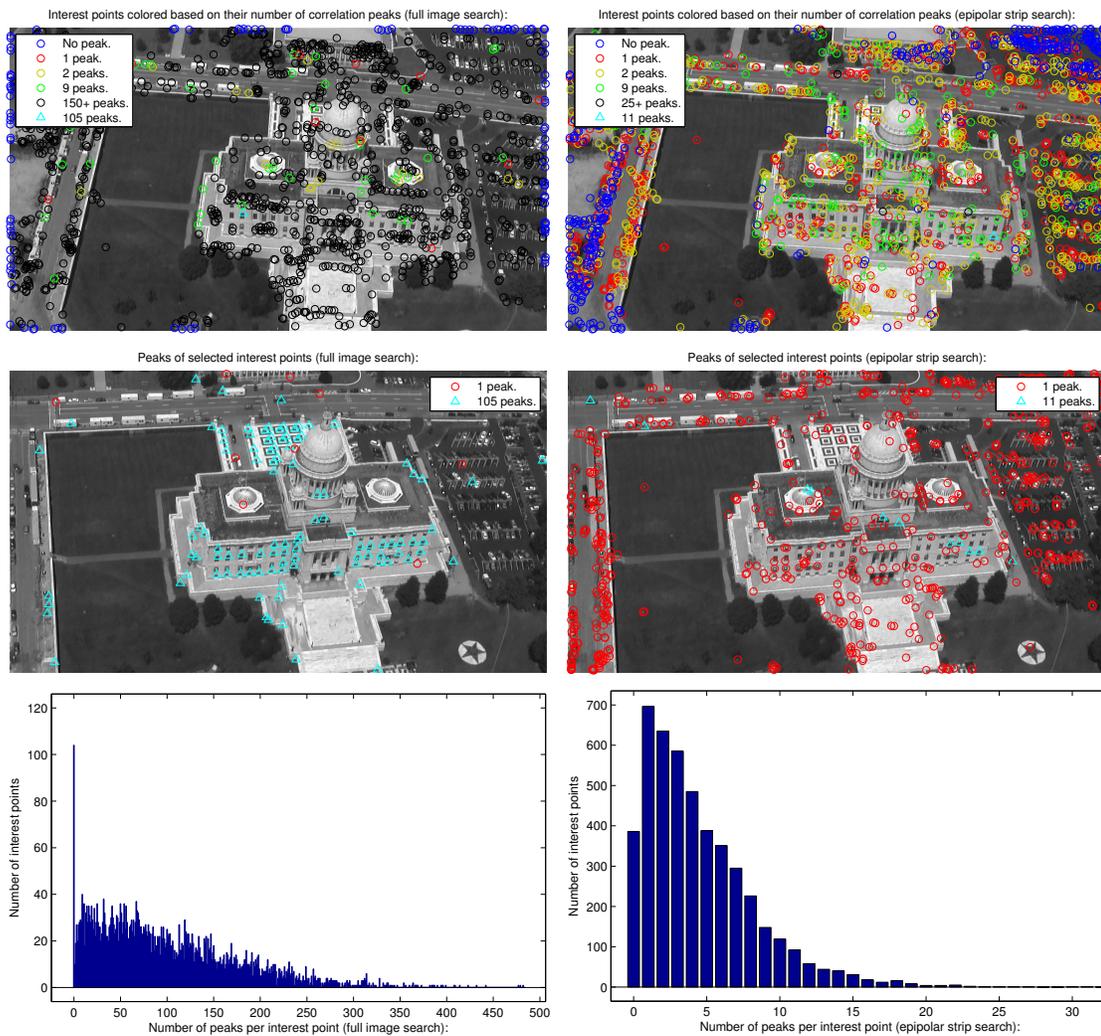


Figure 4.12: Comparison of spatial distribution and density of normalized cross-correlation function peaks. *Left images*: search window is the full image. *Right images*: search window is a strip around the epipolar line. *Top row*: reference image interest points colored according to the legend based on the number of peaks they are highly correlated with in the target image. *Center row*: associated target image peaks for selected interest points. The density of unambiguous points, which have a single match (red circle markers), increases drastically when performing an epipolar strip search. Unambiguous points are useful for initial estimates of fundamental matrices essential to the matching process. One point with relatively high match ambiguity (triangle marker) is shown for each search type. *Bottom row*: illustration of the distributions of number of peaks per interest point for the two search cases.

4.3.5 Native resolution matching

After low-resolution preprocessing, the matching returns to the original scale. The estimated \mathcal{F} is scaled back to native resolution to rectify and perform matching on the original images. This is illustrated in the flowchart in figure 4.3. The epipolar strip search is then used for normalized cross-correlation. The resulting collection of matches in native resolution is disambiguated through the processes described in section 4.4, identically as it was done for low-resolution matching.

4.4 Disambiguation process

4.4.1 Overview

Once a collection of sets of match candidates $M_c(I_R)$ is found, the disambiguation process analyzes each set and determines which matches are ambiguous. This process runs in both low and native resolution matching and is identical in both cases. When only one match candidate exists for an interest point, the single candidate is chosen as a correspondence, except in the very rare event its score is not positive. These matches represent *preliminary* matches. The interest points are then partitioned into two groups according to match ambiguity:

- **Group \mathbf{U}_m :** group of interest points \mathbf{x}_R such that the disambiguation process has found a unique match candidate $\mathbf{x}_T \in p_m(\mathbf{x}_R)$ to be feasible as an estimate of the true match of \mathbf{x}_R .
- **Group \mathbf{M}_m :** is the complement of \mathbf{U}_m . \mathbf{M}_m stores the interest points such that multiple matches are a feasible correspondence.

Each element of \mathbf{U}_m represents a unique correspondence estimate $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$. Denote these correspondences $\tilde{\mathbf{U}}_m$. Preliminary matches are the initial correspondences in $\tilde{\mathbf{U}}_m$ and are mainly unambiguous matches. In low-resolution matching, preliminary matches represent the very first estimated matches of an image pair, providing the first estimate of \mathcal{F} between downsampled I_R and I_T . \mathcal{F} is required for the disambiguation process, which iteratively refines \mathcal{F} (see figure 4.4). In native resolution matching, preliminary matches are the unambiguous epipolar strip matches and the fundamental matrix estimated from them replaces the one computed at low resolution, used solely

for the epipolar strip search and rectification of native images. The new \mathcal{F} drives the disambiguation at native resolution. The estimation of fundamental matrices uses RANSAC and is described in detail in section 4.4.2. The inliers from the RANSAC estimation are kept in \mathbf{U}_m , and outliers are assigned to \mathbf{M}_m , even though they have a single match in p_m . The matching pipeline is designed to only allow matches that tightly satisfy the epipolar constraint.

The disambiguation process launches by running an iterative series of *disambiguation modules* and *outlier rejection modules*, as shown in figure 4.4. Disambiguation modules succeed when specifying a match $\mathbf{x}_T \in p_m(\mathbf{x}_R)$ for $\mathbf{x}_R \in \mathbf{M}_m$, in which case, \mathbf{x}_R moves to \mathbf{U}_m , $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ is included in $\tilde{\mathbf{U}}_m$ and \mathbf{x}_T is removed from $p_m(\mathbf{x}_R)$. There are two outlier rejection modules: one analyzes $p_m(\mathbf{x}_R)$ and removes elements that do not satisfy a minimum quality condition for the epipolar constraint, while the other module moves some points from \mathbf{U}_m back into \mathbf{M}_m according to its inconsistency criterion. Interest points \mathbf{x}_R that rejoin \mathbf{M}_m return to the disambiguation process with one least match candidate in $p_m(\mathbf{x}_R)$, due to the removal that happens when \mathbf{x}_R moves to \mathbf{U}_m . The removal guarantees the iterative process converges even though points moved to \mathbf{U}_m may return to \mathbf{M}_m . It is possible that some points may end up with a single putative match in p_m . This special case is handled only by a modified version of the epipolar constraint module, described in section 4.4.5. As a second special case, $p_m(\mathbf{x}_R)$ may become empty for some \mathbf{x}_R , which is then promptly removed from the matching process and marked as having no match in I_T .

As more matches are found, the fundamental matrix \mathcal{F} is refined and the process repeats, as in figure 4.4, with predominantly increasing efficiency. At any given time, the unique correspondences associated to \mathbf{U}_m indicate the current set of estimated matches from which \mathcal{F} is recomputed. The iteration ends when the disambiguation process fails to assign a new match for all \mathbf{x}_R in \mathbf{M}_m . At this time, a non-iterative final stage of the disambiguation process employs a more accurate multi-view module and a final pruning. The resulting $\tilde{\mathbf{U}}_m$ is the final set of correspondences for the proposed matching method and the fundamental matrix \mathcal{F} estimated from $\tilde{\mathbf{U}}_m$ is used in subsequent image pairs matching where the current I_R is used as an auxiliary view.

The remainder of this section is structured as follows. Two methods that are commonly used throughout the disambiguation process are first presented in detail, namely the fundamental matrix

estimation in section 4.4.2 and the ratio-inequalities test in section 4.4.3. Other procedures are organized in the order of appearance in figure 4.4. The two *basic disambiguation modules* are discussed in sections 4.4.4 and 4.4.5. Sections 4.4.6 and 4.4.10 present the two outlier rejection methods. Section 4.4.7 gives an overview of the two *multi-view disambiguation modules* (described in detail in sections 4.4.9 and 4.4.11), which adopt a third view and use a method denoted *indirect matches* (section 4.4.8) to extend pairs of correspondences into triplets.

4.4.2 Fundamental matrix estimation

A fundamental matrix \mathcal{F} between two views is computed using RANSAC [38] and a set of correspondences $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ expected to satisfy the epipolar constraint

$$\mathbf{x}_T^\top \mathcal{F} \mathbf{x}_R = 0. \quad (4.23)$$

At each iteration, a sample of 8 randomly selected correspondences are chosen to compute \mathcal{F} using the normalized eight-point algorithm and the number of trials necessary to ensure a sample is free from outliers (algorithm termination) is computed adaptively by probing the data [53].

Sampson distance. In general, the correspondence pair $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ does not satisfy equation (4.23) and the estimation depends on the choice of an error model measuring the gap between corresponding points and epipolar lines. The cost function chosen to enforce the epipolar constraint is the *Sampson distance*, which complexity lies between the algebraic and geometric distances. The geometric distance is the gold standard error model and it is defined and used at the *epipolar transfer module* (see equation (4.33) and section 4.4.11), a final one-time disambiguation step. However, for RANSAC estimation of \mathcal{F} , the geometric distance is relatively expensive to compute several times. The Sampson distance is preferred for providing a close first-order approximation of the geometric distance yielding excellent estimation results with a much simpler implementation [53]. The Sampson squared distance for a correspondence pair $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ is given by

$$\frac{(\mathbf{x}_T^\top \mathcal{F} \mathbf{x}_R)^2}{(\mathcal{F} \mathbf{x}_R)_1^2 + (\mathcal{F} \mathbf{x}_R)_2^2 + (\mathcal{F}^\top \mathbf{x}_T)_1^2 + (\mathcal{F}^\top \mathbf{x}_T)_2^2}, \quad (4.24)$$

where $(\mathcal{F} \mathbf{x}_R)_j^2$ is the square of the j -th component of the vector $\mathcal{F} \mathbf{x}_R$. The distance is zero for true matches and expected to be small for estimated ones. The default threshold for finding RANSAC

inliers is the Sampson distance upper bound $\tau_f = 0.2$.

4.4.3 Ratio-inequalities test

Many matching confidence tests comprise of the ratio of a given quality measure involving a feature and its best two matching features from a set of possible match candidates. A ratio test increase the confidence on the choice of the best candidate by ensuring it is much better than any other alternative. For instance, in attempting to match 128-dimensional SIFT feature descriptors in [74], the author notes that a global threshold in the distance to nearest descriptor did not perform well for establishing a match, and a more effective measure is to compare the ratio of distances of the two nearest neighbors. In another example, ratio tests were used to match adjacent jigsaw puzzle pieces [44].

In the context of feature matching for this thesis, ratio tests are widely used. In addition, in the low-dimensional 2-d image space of feature coordinates used here, a global threshold is also used to enforce matching errors are within a few pixels or less.

The ratio-inequalities test requires defining a cost function for pairs of feature locations, *e.g.* the Euclidian distance or the Sampson distance. The costs between a point in question and a set of other possibly corresponding points is computed. Three operations on the lowest cost, d_1 , and the second lowest cost, d_2 , are performed:

$$\frac{d_2}{d_1} > \tau_{\text{ratio}}, \quad (4.25)$$

$$d_1 < \tau_{\text{global}}, \quad (4.26)$$

$$d_2 > \tau_{\text{global}}. \quad (4.27)$$

If all three inequalities are satisfied, the candidate with lowest cost, d_1 , is chosen as the matching point. The ratio constraint in equation (4.25) guarantees the best candidate is much better than any other. The other inequalities are important to ensure the effectiveness of the ratio constraint in two cases: when the lowest distances are either both too small or both too large. When both are too small, the ratio constraint is unreliable as both costs indicate a possible true match regardless of the ratio. When both are too large, the true match was probably not detected and choosing any

candidate would likely be a mistake. The value τ_{global} represents the threshold between suitable and ineligible matches.

4.4.4 Highest scores ratio module

Let \mathbf{x}_R be a feature with multiple matching candidates. The candidate with the highest correlation score is accepted as the correspondence of \mathbf{x}_R if the ratio of its score and the second highest score is larger than a threshold τ_{hsr} .

4.4.5 Epipolar constraint module

This test analyzes the proximity of the match candidates to the epipolar line associated to their interest point. The candidate with the lowest Sampson distance is accepted as the true match if its distance passes a ratio-inequalities test with threshold $\tau_{\text{ratio}} = 10$ and $\tau_{\text{global}} = 0.2$. If there is only one candidate left, a complete ratio-inequalities test is not feasible given at least two candidates are required. Instead, only equation (4.26) is used in this case.

4.4.6 Epipolar sectors module

This section proposes an outlier rejection approach that does not use a threshold. The epipolar sectors module rejects potentially false matching candidates by enforcing that they must lie very near their associated epipolar lines, which are computed from the fundamental matrix \mathcal{F} . As discussed in section 4.4.1, \mathcal{F} is estimated from \mathbf{U}_m , the set of estimated matches at some given time.

A Delaunay triangulation \mathcal{D}_t (see section 3.3) is constructed on the image plane of I_R from the point locations in \mathbf{U}_m and its triangles are used to define the concept of proximity to the epipolar lines. Note, \mathcal{F} and \mathcal{D}_t are computed from the same set of points \mathbf{U}_m leading to a connection between higher spatial accuracy of \mathcal{F} where \mathcal{D}_t has smaller triangles, discussed later in detail.

This module complements the imposition of the epipolar constraint by a search for correlation matches exclusively on a strip around epipolar lines, since the strip is relatively thick (21 pixels) and has constant size. Note, correlation matches are strictly defined as local peaks in a 3×3 neighborhood, then border correlation pixels are never considered peaks, and the effective strip thickness becomes

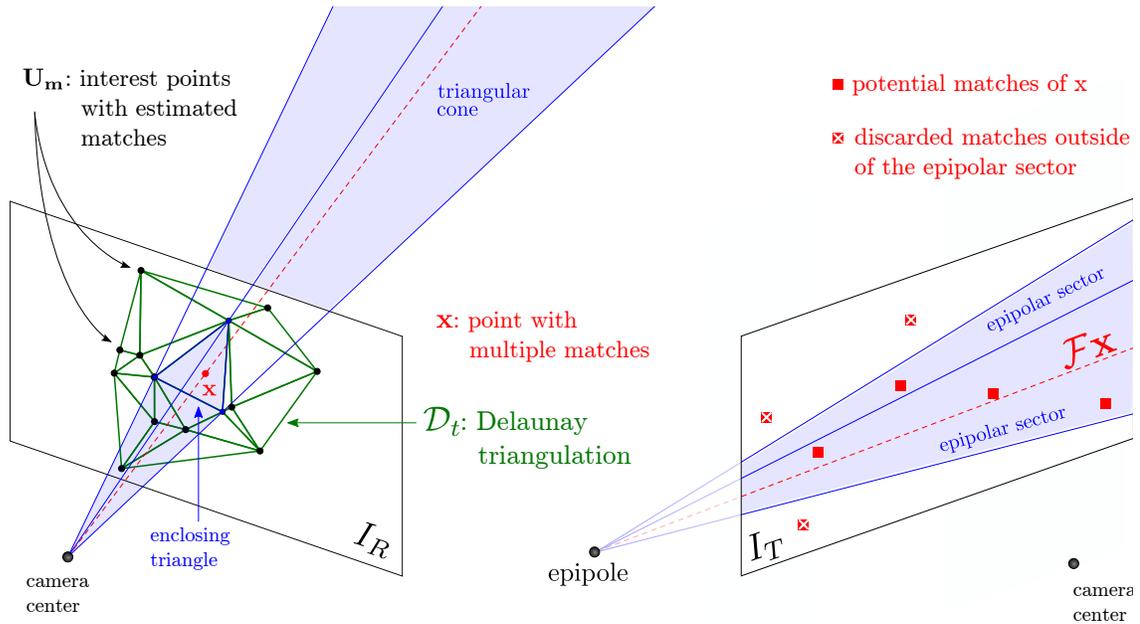


Figure 4.13: Elimination of outlier matches from the epipolar sectors module, which is based on the epipolar constraint. Matches not within a certain region around their associated epipolar lines are discarded, reducing match ambiguity. The bounds are determined by geometric relations involving a given point in the match disambiguation process and its surrounding points that have matches already established. The surrounding points are vertices of a Delaunay triangle.

19 pixels. Thus, refined peak locations are at most 9.5 pixels away from the epipolar lines.

The rejection method is shown in figure 4.13 and defined as follows. Extract from \mathcal{D}_t the enclosing triangle of an interest point $\mathbf{x}_R \in \mathbf{M}_m$ that has multiple match candidates. The reference image camera center and the enclosing triangle define a triangular cone \mathcal{C}_t in the 3-d scene. Let the image of \mathcal{C}_t in the target image be the *epipolar sector*, a circle sector centered at the epipole. Note, the edges of the solid \mathcal{C}_t are three optical rays emanating from the camera center and passing through the enclosing triangle vertices and the epipolar sector is the silhouette of \mathcal{C}_t seen from I_T . Compute the epipolar lines for each one of the three corners of the enclosing triangle. Since the lines all meet at the epipole in I_T , one of the lines will be in between the other two, which are outer lines defining the boundary of the epipolar sector. Geometrically, since \mathbf{x}_R is inside the triangle, its associated optical ray (in 3-d) lies inside \mathcal{C}_t . Consequently, the epipolar line from \mathbf{x}_R (and likely its true match) must lie within the sector. Therefore, all candidates outside the sector are discarded. The strength of this test is three-fold:

- No threshold needs to be defined.
- The rejection intensity is proportional to the spatial accuracy of the fundamental matrix \mathcal{F} . For instance, if matches are denser in a region, the spatial accuracy of epipolar lines derived from \mathcal{F} is higher on the associated part of the scene. This yields smaller enclosing triangles, narrower epipolar sectors, and tighter bounds, since the sectors define the tolerance for violating the epipolar constraint. Analogously, on regions of sparse matches, the bounds are relaxed.
- Finally, the Delaunay triangulation depends on the coordinates of the matched interest points in the reference image, but the correctness of the correspondences is basically unimportant, since the geometric relations described do not depend on the match locations. False matches can only affect the choice of enclosing triangles in their local neighborhood and their negative effect on the estimation of \mathcal{F} is negligible since the epipolar constraint is enforced on them by the matching pipeline.

4.4.7 Multi-view disambiguation modules

Unlike the basic modules described above, multi-view disambiguation modules use an additional third view, the *auxiliary* viewpoint or I_A , to check for the geometric compatibility of matches using *point transfer* methods. Given a correspondence in two views, which is the image of an unknown 3-d point \mathbf{X} , determining the image location of \mathbf{X} in a third view with no use of image content is a point transfer problem. This may be possible if enough geometric information is available regarding the three views, *e.g.* the placement of the cameras or the fundamental matrices.

When an interest point \mathbf{x}_R on the reference view, I_R , has multiple possible correspondences in a target view, I_T , that no other simpler tests were able to disambiguate, multi-view disambiguation modules attempt to choose the correct match, if any, in one of two different ways:

Method 1: if a match has been estimated for \mathbf{x}_R on the auxiliary view, the winning candidate must be the closest one to the corresponding transferred point and pass a ratio-inequalities test (section 4.4.3). This method is used by the *affine transfer modules* (section 4.4.9).

Method 2: if a match on the auxiliary view is unknown, the point transferred to it from the winning match must be the only one to share some common appearance properties with the interest point \mathbf{x}_R . This method relies on the low probability that a transferred point from a wrong match would also have the common appearance, and uses the intensity content from image I_A to validate the transferred point. This method is used by the *epipolar transfer module* (section 4.4.11).

The two multi-view disambiguation modules accomplish similar tasks. The main difference is that the affine transfer modules use an approximation and are faster, whereas the epipolar transfer module can operate without estimated correspondence triplets and uses image content. As seen in figure 4.4, the affine transfer modules are used inside a loop, as they are faster, and the approximation used does not disrupt the quality of disambiguated matches. The epipolar transfer module is used once at the end of the disambiguation process when the quality of estimated fundamental matrices is the highest, which is essential for a module designed to address matches all others fail to disambiguate.

4.4.8 Indirect matches

As seen in figure 4.14, if a point \mathbf{x}_R , in one image, matches to a point \mathbf{x}_T , in a second image, and \mathbf{x}_T matches to a point \mathbf{x}_A , in a third image, then \mathbf{x}_R matches to \mathbf{x}_A through the pair of known matches without actually performing a match search, defining an *indirect match*. Indirect matches are a means to quickly extend two pairwise correspondences into a match triplet:

$$\begin{cases} \mathbf{x}_R \leftrightarrow \mathbf{x}_T \\ \mathbf{x}_T \leftrightarrow \mathbf{x}_A \end{cases} \Rightarrow \mathbf{x}_R \leftrightarrow \mathbf{x}_A \Rightarrow \mathbf{x}_R \leftrightarrow \mathbf{x}_T \leftrightarrow \mathbf{x}_A. \quad (4.28)$$

In practice, one may have instead $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ and $\hat{\mathbf{x}}_T \leftrightarrow \mathbf{x}_A$, with $\mathbf{x}_T \neq \hat{\mathbf{x}}_T$. One may extend these matches into $\mathbf{x}_R \leftrightarrow \mathbf{x}_T \leftrightarrow \mathbf{x}_A$ (or optionally $\mathbf{x}_R \leftrightarrow \hat{\mathbf{x}}_T \leftrightarrow \mathbf{x}_A$) if $\mathbf{x}_T \approx \hat{\mathbf{x}}_T$:

$$\begin{cases} \mathbf{x}_R \leftrightarrow \mathbf{x}_T \\ \hat{\mathbf{x}}_T \leftrightarrow \mathbf{x}_A \\ \mathbf{x}_T \approx \hat{\mathbf{x}}_T \end{cases} \Rightarrow \mathbf{x}_R \leftrightarrow \mathbf{x}_A \Rightarrow \begin{cases} \mathbf{x}_R \leftrightarrow \mathbf{x}_T \leftrightarrow \mathbf{x}_A \\ \mathbf{x}_R \leftrightarrow \hat{\mathbf{x}}_T \leftrightarrow \mathbf{x}_A \end{cases}. \quad (4.29)$$

The criterion to determine if there is a unique $\hat{\mathbf{x}}_T$ satisfying $\mathbf{x}_T \approx \hat{\mathbf{x}}_T$ among all estimated matches from I_T to I_A is via a ratio-inequalities test using the Euclidian distances from \mathbf{x}_T to all points

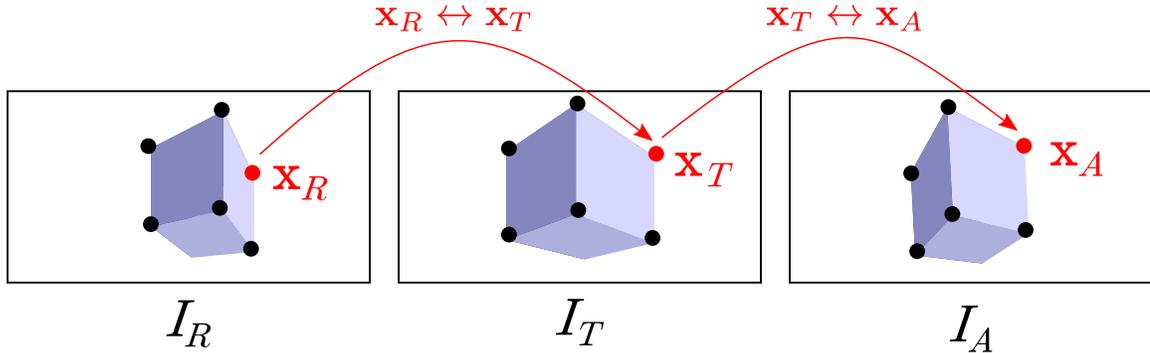


Figure 4.14: Illustration of matches $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ from I_R to I_T and $\mathbf{x}_T \leftrightarrow \mathbf{x}_A$ from I_T to I_A . $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ and $\mathbf{x}_T \leftrightarrow \mathbf{x}_A$ are established through a search using the matching framework (*direct matches*), which together imply an *indirect match* $\mathbf{x}_R \overset{I_T}{\leftrightarrow} \mathbf{x}_A$ from I_R to I_A established through a common matching point in I_T without a matching search.

$\hat{\mathbf{x}}_T \in I_T$ from the given correspondences. Cases where $\mathbf{x}_T \neq \hat{\mathbf{x}}_T$, but $\mathbf{x}_T \approx \hat{\mathbf{x}}_T$, arise when \mathbf{x}_T is a subpixel location matching some \mathbf{x}_R via correlation and $\hat{\mathbf{x}}_T$ is an interest point of I_T detected as a corner (section 4.2).

The indirect matches are used by disambiguation modules to acquire rough estimates, *e.g.* a fundamental matrix between I_R and I_A without matching them directly.

4.4.9 Affine transfer modules

Given three views and their cameras, rays back-projected from corresponding points in two views intersect in a 3-d point. The forward projection of the 3-d point onto the image plane of the third view results in the corresponding point in that view. This type of transfer can rule out many false correlation matches found in a view that are far from the transferred point.

The affine transfer modules are multi-view tests that analyze the geometric consistency of triplets of correspondences in 3 views via estimated affine cameras, which have $(0, 0, 0, 1)$ as the last row of the projection matrix. The cameras are estimated using the factorization method [117, 53]. Only correspondence triplets are used. The solution is not unique and up to a full continuum of affine transformations, yet any solution can transfer points and attempt to disambiguate matches. Correspondences between the auxiliary and reference views must have been estimated prior to this test. This disambiguation test is fast but approximate and is complemented by a similar method,

the epipolar transfer module in section 4.4.11, which trades speed for accuracy.

Finding match triplets. In order to proceed, one must extend the pairwise matches into triplets for the three views in question. This extension depends on the arrangement of the views (see figure 4.1):

Arrangement 1: auxiliary is the prior view, followed by the reference and then the target views.

The matching order is auxiliary into the reference followed by reference into the target.

Arrangement 2: auxiliary is a posterior view, so that the reference image interest points are matched into it, and also into the target.

In arrangement 2, interest points from I_R are matched into both I_A and I_T . Since the set of interest points is exactly the same in both cases, triplets are trivially defined by the interest points matched twice. In arrangement 1, there are in I_R two sets of points: a set of correlation matches from interest points of I_A and a set of interest points of I_R itself. Note, these points may not be identical and the pairs of correspondences are extended to triplets using *indirect matches* (section 4.4.8).

Affine camera estimation via RANSAC. Given match triplets extended from pairwise matches, a measurement matrix W is constructed and affine cameras are estimated using the factorization method to decompose W into a motion matrix and a structure matrix, as described in [53] in more detail. The decomposition is very efficient since it only uses the SVD of $W = UDV^\top$ truncated to rank 3, *i.e.* $\hat{W} = U_{2m \times 3} D_{3 \times 3} V_{3 \times n}^\top$, where m denotes the number of views and n denotes the number of triplets. Since three views are used, $m = 3$. The affine cameras are taken from $U_{2m \times 3}$.

Given the estimated cameras for the 3 views, affine transfers are carried out for all estimated triplets by triangulating corresponding rays (in pairs) into 3-d points and projecting these back into the views to compute reprojection errors. Since the rays do not meet in space, the intersection is taken as the point of minimum distance from both rays, *i.e.*, the midpoint of the line segment that is perpendicular to both rays and also join them. Given a reprojection error threshold τ_{affine} , outliers can be determined. This process is repeated a few times to estimate a final set of inliers and cameras via RANSAC using a minimum of 4 random triplets at each iteration. Since the affine model is an

approximation of the actual camera model, average inliers errors are expected to be higher than a pixel. A threshold of $\tau_{\text{affine}} = 10$ pixels is chosen empirically.

Disambiguation. The reprojection error of correct matches are expected to be within the range of the inliers errors. Multiple matches are then disambiguated based on the highest inlier error τ_{MAX} . Given an interest point \mathbf{x}_R in the reference view that has a match \mathbf{x}_A on the auxiliary view, the correspondence is transferred to the target as \mathbf{x}_T . Note, a match \mathbf{x}_A exists when it can be established as in the preceding via *indirect matches* or intersection, respectively for arrangements 1 or 2. Given that \mathbf{x}_R has multiple matches in the target, a winning match is chosen as the one which passes a ratio-inequalities test on its distance from transferred point \mathbf{x}_T using threshold $\tau_{\text{global}} = \tau_{\text{MAX}}$ and default τ_{ratio} , otherwise a match remains unknown.

4.4.10 Match topology module

Even under a careful disambiguation process, some wrong estimated matches exist due to small inaccuracies throughout the matching procedure. This module rejects inconsistent matches based on other neighboring matches. Given a set of corresponding points distributed in two images, the detection insight is based on nearby points $\mathbf{N}_{\mathbf{x}_R}$ (defined below) of a given point \mathbf{x}_R and the nearby points $\mathbf{N}_{\mathbf{x}_T}$ of its estimated match \mathbf{x}_T . The neighbor set $\mathbf{N}_{\mathbf{x}_T}$ tends to correspond to the set $\mathbf{N}_{\mathbf{x}_R}$, when \mathbf{x}_R indeed corresponds to \mathbf{x}_T . This relationship is expressed by the notation

$$\mathbf{N}_{\mathbf{x}_R} \longleftrightarrow \mathbf{N}_{\mathbf{x}_T} = \{r \leftrightarrow l \mid r \in \mathbf{N}_{\mathbf{x}_R} \text{ and } l \in \mathbf{N}_{\mathbf{x}_T} \text{ are estimated matches}\} \quad (4.30)$$

representing all estimated correspondences found in the neighboring sets. If \mathbf{x}_T is a false match away from the true match of \mathbf{x}_R , it is probable that $\mathbf{N}_{\mathbf{x}_R} \longleftrightarrow \mathbf{N}_{\mathbf{x}_T} = \emptyset$ and many of these false positives can be easily detected assuming they are isolated within a reasonably denser set of true correct matches. Under occlusion and in the neighborhood of false matches the cardinality of $\mathbf{N}_{\mathbf{x}_R} \longleftrightarrow \mathbf{N}_{\mathbf{x}_T}$ is reduced even if $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ is a true match, yet a large fraction of points in the neighboring sets still regularly corresponds. The match topology module rejects inconsistent matches that show little evidence of correctness via small cardinality

$$|\mathbf{N}_{\mathbf{x}_R} \longleftrightarrow \mathbf{N}_{\mathbf{x}_T}| \leq n_m, \quad (4.31)$$

for $n_m = 1$. The threshold n_m could be a percentage of neighbors instead, however empirical observation suggests that enforcing at least 2 matches being the same regardless of $|\mathbf{N}_{\mathbf{x}_R}|$ and $|\mathbf{N}_{\mathbf{x}_T}|$ produces better detection.

If equation (4.31) is satisfied, \mathbf{x}_T is discarded as a match candidate and the disambiguation process for \mathbf{x}_R starts over. In this case, \mathbf{x}_R moves back from \mathbf{U}_m into \mathbf{M}_m , but \mathbf{x}_T is no longer in $p_m(\mathbf{x}_R)$.

Choosing the neighborhood structure. The k -nearest neighbors algorithm could be used to express neighborhood, but the choice of the good parameter k is not obvious as match density varies. The use of a 2-d Delaunay triangulation of the matched points in \mathbf{U}_m is preferred as there is no parameter to choose and also it provides nearby matches in all directions (when available) ensuring a surrounding neighborhood for both $\mathbf{x}_R \in \mathbf{U}_m$ in I_R and its match \mathbf{x}_T in I_T . There is one triangulation for the reference image and other for the target. In average, $|\mathbf{N}_{\mathbf{x}_R}| = |\mathbf{N}_{\mathbf{x}_T}| = 6$ since the sets are computed from a Delaunay triangulation (see section 3.3), so the threshold $n_m = 1$ corresponds in average to 16.7% of the nearby points. Figure 4.15 exemplifies a 2-d Delaunay triangulation used by this outlier rejection method in a pair of images.

Note, this method is a simplification of the spatial consistency method from [12], which also takes into consideration the ratios of distances of the corresponding features \mathbf{x}_R and \mathbf{x}_T to their corresponding neighbor matches. The proposed simplification is more suitable for real-time goals. Moreover, the input data used here has already been pruned by other disambiguation modules and has lower ambiguity than the data used in [12].

4.4.11 Epipolar transfer module

Given 3 views, their pairwise fundamental matrices and a pair of corresponding points in two views, epipolar geometry dictates the matching point in the third view must lie at the intersection of the associated epipolar lines. If no occlusion happens, the transferred point must have similar appearance when seen in either view. Based on this evidence, the epipolar transfer module is a photometric and geometric multi-view test that attempts to resolve ambiguous matches from an image pair using the image content of a third view. It is performed as a last resort at the final stage of the matching

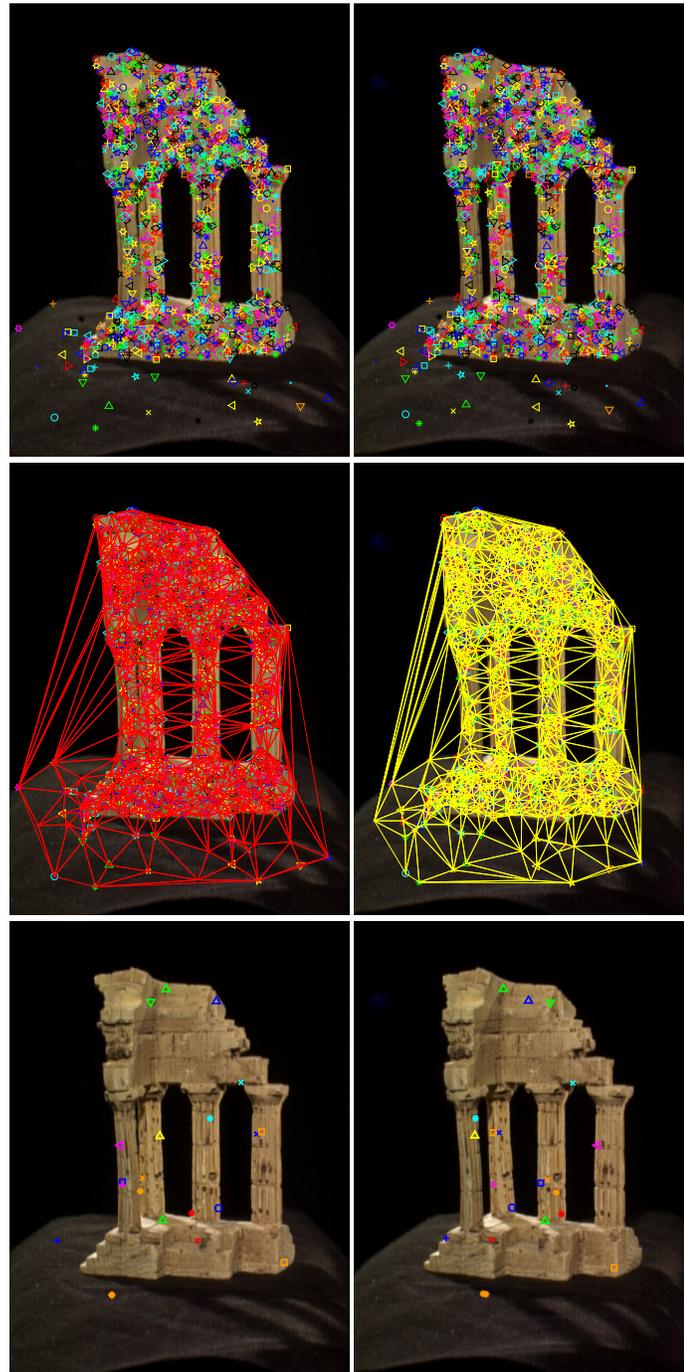


Figure 4.15: Illustration of match topology module disambiguation steps operating in two views of the “templeRing” dataset. The top row shows estimated matches for the image pair. Matching points are shown with the same marker shape and color, which are picked randomly (see section 4.1.1). The center row displays the Delaunay triangulations computed on matching points of each image. The triangulations are used to define the neighboring matches for each point in each view. The bottom row presents correspondences rejected as false matches using the match topology module.

process, applied to the most challenging points that all other attempts failed to disambiguate. Thus, it must be more effective than the affine transfer modules, and in fact, it is theoretically more accurate since the highest number of matches are available to encode the scene geometry via estimated fundamental matrices. Furthermore, match location refinements are performed and there is no use of approximations such as affine cameras.

The disambiguation is based on the similarity of the appearances of corresponding points, measured by normalized cross-correlation. A point \mathbf{x}_A in a given view, transferred from a match candidate $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ in a pair of other views, must lie in the intersection of the epipolar lines associated to \mathbf{x}_R and \mathbf{x}_T . Note, this point transfer method is called *epipolar transfer* [53] and it becomes increasingly ill-conditioned as the 3-d point associated to the matches lies closer to the *trifocal plane* defined by the three camera centers. In this case, the epipolar lines are coincident. In the special case the camera centers are collinear, the trifocal plane is not uniquely defined and epipolar transfer fails for all points. According to the proposed problem assumptions, camera geometry is unknown and the proposed disambiguation method does not attempt to detect these degenerate conditions, it is simply designed to provide no match under such circumstances.

Another feasible way to obtain the transfers is to compute projective cameras and project 3-d points. The camera parameters would have to be optimized via bundle adjustment [118] to achieve similar results as this module, but it is possible to be trapped in a bad local maximum due to small baselines. Moreover, the optimization would be driven by the already estimated match triplets, whereas the epipolar transfer module performs an optimization in transferring the *ambiguous* matches, which are the ones requiring attention. Hence, the epipolar transfer module method is preferred based on previous insights. The desambiguation is carried out as follows.

Estimating fundamental matrices. Let I_R be a reference view, I_T be a target view and I_A be an auxiliary view. Let the matching process between I_A and I_R be successfully finished and the one between I_R and I_T be the one in progress and at the final stage. Therefore, matches between I_A and I_R are already estimated, as well as many between I_R and I_T . If images I_A and I_T were never matched, correspondences among them are quickly computed using *indirect matches* (section 4.4.8) through I_R . Given the three sets of pairwise matches between the views, let \mathcal{F}_{10} , \mathcal{F}_{20} and \mathcal{F}_{21} be the

associated fundamental matrices, obtained independently as described in section 4.4.2.

Point transfer and refinement. Given an interest point in I_R with several match candidates in I_T , take any of the possible putative matches $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$. In practice, this match does not satisfy the epipolar constraint relation

$$\mathbf{x}_T^\top \mathcal{F}_{10} \mathbf{x}_R = 0. \quad (4.32)$$

A refinement is required to increase the disambiguation success rates under near degenerate conditions. An optimized match would be one such that the refined points are as close as possible to the original match but satisfy equation (4.32) relation exactly. This is accomplished in terms of the *geometric distance* [53]. The geometric distance $G_d(\mathbf{x}_R, \mathbf{x}_T)$ minimizes the Euclidian distance of the correspondence $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ w.r.t. all matches $\hat{\mathbf{x}}_R \leftrightarrow \hat{\mathbf{x}}_T$ that perfectly satisfy the epipolar constraint $\hat{\mathbf{x}}_T^\top \mathcal{F}_{10} \hat{\mathbf{x}}_R = 0$ and is given by

$$G_d(\mathbf{x}_R, \mathbf{x}_T)^2 = \min_{\hat{\mathbf{x}}_T^\top \mathcal{F}_{10} \hat{\mathbf{x}}_R = 0} [d(\mathbf{x}_R, \hat{\mathbf{x}}_R)^2 + d(\mathbf{x}_T, \hat{\mathbf{x}}_T)^2], \quad (4.33)$$

where $d(\cdot, \cdot)$ is the Euclidian distance. Minimizing the geometric distance is equivalent to minimizing reprojection error since, given projection matrices compatible with \mathcal{F}_{10} , the points $\hat{\mathbf{x}}_R \leftrightarrow \hat{\mathbf{x}}_T$ can be perfectly triangulated into a 3-d point \mathbf{X} , *i.e.*, the associated rays will meet precisely in 3-d space. Note, the test does not need to triangulate the match into \mathbf{X} . The minimization can be solved either by a numerical minimization, or it can be reduced, using elementary calculus, to a more efficient non-iterative computation of the real roots of a sixth-degree polynomial [53]. The refined match $\hat{\mathbf{x}}_R \leftrightarrow \hat{\mathbf{x}}_T$ is computed via the more efficient method involving the six-degree polynomial and it is transferred to I_A as the intersection of the epipolar lines $\mathcal{F}_{20} \hat{\mathbf{x}}_R$ and $\mathcal{F}_{21} \hat{\mathbf{x}}_T$:

$$\mathbf{x}_A = (\mathcal{F}_{20} \hat{\mathbf{x}}_R) \times (\mathcal{F}_{21} \hat{\mathbf{x}}_T). \quad (4.34)$$

Image content analysis. The image content is now analyzed to check if $\hat{\mathbf{x}}_R$ and \mathbf{x}_A have similar appearance. If not, the candidate \mathbf{x}_T is rejected as a match for \mathbf{x}_R . The image content is analyzed via normalized cross-correlation on the rectified views I_R and I_A . It is sufficient to know if there is a supporting feature on \mathbf{x}_A or its surroundings. Hence, the normalized cross-correlation operation

is restricted to a small rectangular search window around \mathbf{x}_A with the quality parameter from equation (4.13) set to $Q_m = 1$ to return only a global peak, if exists. The window size is chosen to return a correlation image of size $c_w \times c_h$ (see table 4.2) correlation pixels centered in \mathbf{x}_A . c_w is chosen larger to accommodate larger errors along the direction of the epipolar lines, the x -axis after rectification. This type of error typically increases as baselines become small or camera centers become collinear (ill-conditioned trifocal plane), both leading to shallow angles between the epipolar lines. Small baselines and approximately linear motion are typical in computer vision applications, as well as in the image datasets used in this thesis experiments. Nevertheless, this disambiguation module accurately produces a decent number of matches.

Pruning and disambiguation. The above process is repeated for all putative matches $\mathbf{x}_R \leftrightarrow \mathbf{x}_T$ of \mathbf{x}_R . Some pruning is necessary. Transferred points with weak corner scores are discarded using the weak corner removal method in section 4.3.2. Moreover, similarly to equation (4.13), low-quality peaks are also eliminated if their correlation score is not within the quality level Q_m of the highest score among of all putative matches of \mathbf{x}_R and all transferred points \mathbf{x}_A . A match \mathbf{x}_T is established among all possible candidates if \mathbf{x}_T is the only one that has a valid supporting match around its transferred point \mathbf{x}_A , otherwise the disambiguation fails.

4.5 Conclusion

This chapter presents a novel framework for feature matching using a GPU implementation of NCC (presented in chapter 7), and includes a disambiguation step that runs on CPU and performs a broad analysis of geometric relations to establish more matches. The pipeline uses no prior information about scene geometry. The following contributions are provided:

- Combines Harris-Stephens and Shi-Tomasi corner metrics to get interest points avoiding edge responses.
- Uses multiple ratio tests to disambiguate matches (ratios of correlations, ratios of corner scores, ratios of distances to epipolar lines).
- Rejects outliers based on epipolar circle sectors and without using a threshold.

- Typically does not find matches to points that are out of the target image field of view.
- Uses 3-view geometry to disambiguate matches.
- Enforces spatial consistency of matches.
- Works on environments with limited acquisition control such as aerial urban scenes.
- Achieves higher accuracy than SIFT matching (see chapter 8).

Chapter 5

Planar grid matching

Duplicated structures are ubiquitous in urban scenes and handling their inherent match ambiguity has been an important topic in computer vision. Scene duplicates usually consist of man-made planar objects and are very common in architectural scenes. In this chapter, planar models are introduced to improve ambiguous matching of interest points exhibiting dense repetitive similar texture and enable wide-baseline normalized cross-correlation matching when applicable. The planar model targets man-made planar surfaces having a high density of nearly identical features arranged in periodic grid patterns forming lattices, such as the windows in large tall building facades seen in figure 5.1. Such features are difficult to disambiguate due to their density and periodicity. The algorithm deals with significant viewpoint changes between images, multiple repeating patterns and features of a pattern exhibiting almost identical appearances in any given image. Most existing appearance based algorithms cannot handle densely and highly repetitive textures due to the match location ambiguity. The target structure provides a rich set of repeating features to be matched and tracked across multiple views, improving camera estimation. Wide-baseline tracking is possible using normalized cross-correlation by compensating perspective viewpoint changes from frame to frame for each planar surface. The compensation is possible via image warping through estimated planar homographies induced by the planes of each set of tracked features. Such tracked planar objects provide a first step in the geometric interpretation of a 3-d scene at a stage where cameras and scene geometry are still unknown.



Figure 5.1: Examples of facades of large buildings. All buildings present planar grids of densely repeating features. In the top-right image, a small portion of the visible facades bends and does not qualify as planar. In the bottom-right image, the grid features are only piecewise planar.

The main ideas of this chapter have been published in [4] and it is organized as follows. An outline of the matching approach is given in section 5.1. A method for detecting planar grid points is discussed in section 5.2. Section 5.3 describes the estimation of a lattice that fits the grid points. The estimation of a corresponding lattice in a different image is discussed in section 5.4. Finally, a method for tracking a lattice through multiple images for achieving wide-baseline lattice matching is discussed in section 5.5.

Grid features are notably seen in urban scenes often as windows of large buildings. It is assumed the depth of the plane from the camera is much larger than the variation in the depth of the planar grid such that its image projection is approximately affine. As a consequence, it is assumed that nearby parallel lines in the grid remain parallel in the images and spacing between adjacent features in a given direction is constant.

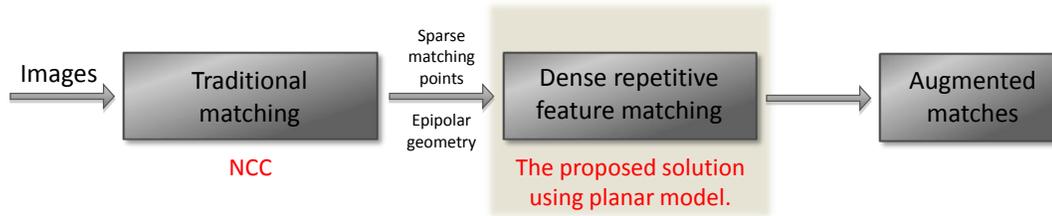


Figure 5.2: Matching pipeline for dense repetitive features, which assumes given epipolar geometry estimated from matches of non-densely repeating features found via method of chapter 4, or via other traditional matching method.

5.1 Overview

Appearance based matching algorithms typically fail to establish a unique correspondence between two views of a highly repetitive texture due to the fact that the appearance is duplicated at multiple locations in both images. Highly repetitive textures mostly occur on planar man-made objects, especially in urban scenes with repeating structure, *i.e.* a homogeneous facade of windows. However, by introducing a planar geometric constraint, the algorithm proposed in this chapter establishes a correspondence between two views of the same repetitive structure by exploiting the global planar geometry to disambiguate features with identical appearance.

The pipeline is illustrated in figure 5.2 and the proposed solution block is expanded in more detail in figure 5.5. In a reference image, potential grid points are detected as features that have many surrounding replicas in their local neighborhood found via normalized cross-correlation, and are defined as a grid seeds. After selecting one grid seed point (figure 5.5a) a larger neighborhood is searched for image locations with similar appearance (figure 5.5b). A Delaunay triangulation provides edge connectivity among these similar points (figure 5.5c) which may include outliers, points that are similar to the seed in appearance, but do not lie on the same planar object. After removing outliers from the original Delaunay triangulation (figure 5.5d), a regular lattice is inferred from the refined triangulation based on common edge lengths and orientations. A regular lattice is a repeating arrangement of points in space defined by a point and two generator vectors. A planar lattice can be seen a regular tiling of the plane by a parallelogram primitive cell (figure 5.3). Regularly repeating features in building facades define planar lattices in 3-d space and their approximately affine

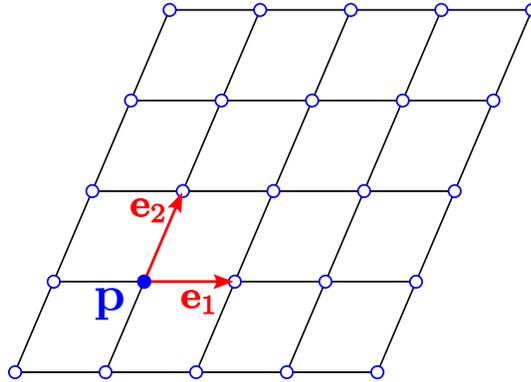


Figure 5.3: Definition of planar lattice: a repeating arrangement of points in space defined by a point \mathbf{p} and two generator vectors, \mathbf{e}_1 and \mathbf{e}_2 , resulting in a regular tiling of the plane by parallelograms.

projections define planar lattices in 2-d images. The aforementioned triangulation is in image space. The estimated lattice grid points are the vertices of the refined triangulation and the two lattice edge directions are inferred from the refined triangulation common edge orientations (figure 5.5e). Unlike the triangulation, the lattice provides a genuinely regular topological structure to the grid features (compare connectivities of figures 5.5d and 5.5e).

Grid points are matched individually in a target image via normalized cross-correlation while imposing the epipolar constraint. Epipolar geometry is estimated from unambiguous matches found elsewhere. Essentially all the grid matches are still highly ambiguous when only considering appearance, due to the repetitive nature and density of the grid. Then, no planar homography mapping can be computed to correspond the underlying planar object in two views. A homography may be estimated using all possible matching candidates and RANSAC [38], but the search for inlier samples is prohibitive when the ambiguity is high due to large outlier percentage. The lattice is computed solely on the reference image and matched on the target image. One alternative approach to solve the matching problem would be to estimate a lattice in the target view as well and match the two lattices. In this solution, the grid points may change due to phenomena such as occlusion or specular reflection and edge connectivities may be completely different as a lattice that fits the grid points is not unique. Therefore, this alternative solution may be complex since the learned lattice structure is not preserved under viewpoint change and will be considered in future work. However, other spatial

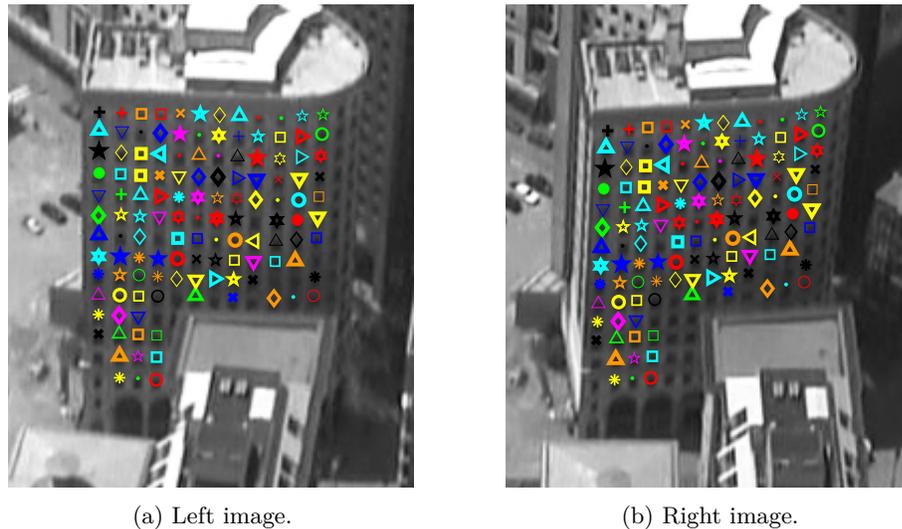


Figure 5.4: Typical example of the model-based pairwise matching method proposed in this chapter. Corresponding points have identical markers (see section 4.1.1).

regularity constraints of lattices are preserved. Then, such constraints are used to disambiguate the point matches and are applied in two stages. First, let a *lattice line* consist of a set of collinear points along one of the generator vectors of a lattice, as the lines in figure 5.3. Constraints are applied to lattice lines such that the sequence of corresponding matches of a sequence of lattice line points must preserve collinearity, ordering and spacings. These constraints drastically reduce point correspondence ambiguity simplifying the problem to disambiguating line matches (shown in figure 5.5f). Finally, the intersection of two lattice lines must have a common corresponding point defined individually from their line match candidates. This property of lattice lines, denoted intersection consistency, is an important concept ensuring every grid point has a common correspondence from the two lines it belongs to. This concludes the model-based disambiguation pipeline, resulting in disambiguated matches (figure 5.5g). The way matches are displayed is discussed in section 4.1.1 and other examples are given in figures 4.5 and 5.4.

After processing a lattice estimated from a seed point, a new seed is selected to process other lattices of the same image and this new seed does not belong to a previously learned or analyzed grid, as shown in figure 5.6. The system only returns a corresponding lattice when both the learning and matching procedures succeed in estimating a structure conforming with a proper planar lattice. Thus,

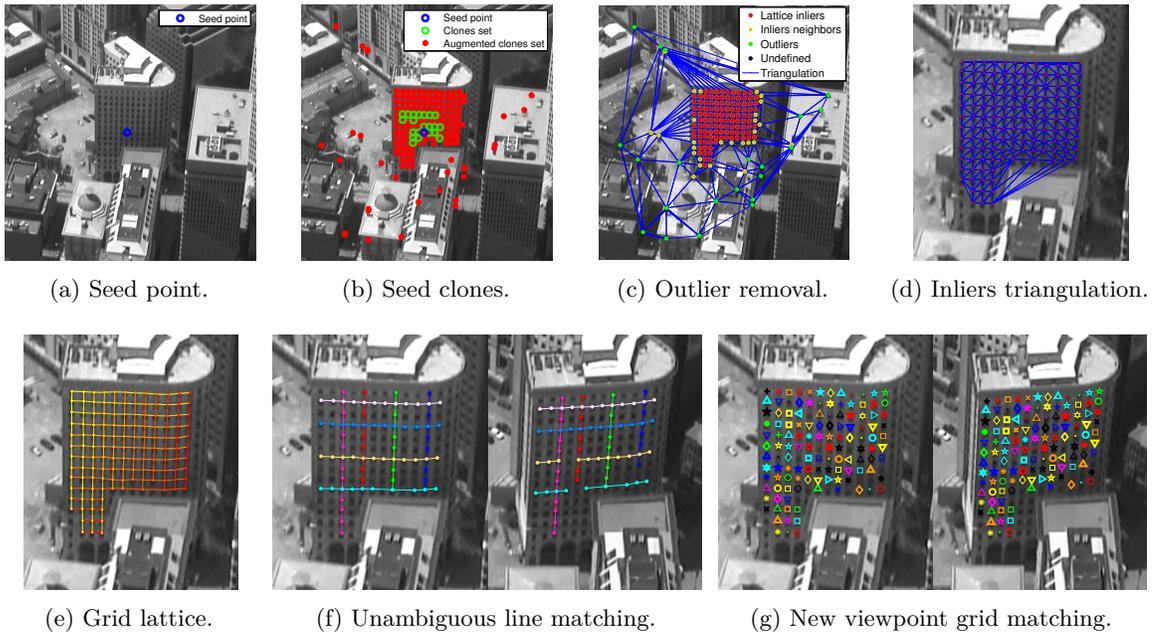


Figure 5.5: Overview of the planar grid matching pipeline. In a reference image, a grid seed point is found (a), similar features are detected as potential grid points (b), outliers are removed (c), inliers are triangulated (d) and a lattice is estimated from the triangulation structure (e). Global lattice spatial constraints are imposed on point matches and the problem of matching ambiguous grid points from the reference to the target image is reduced to a line matching problem (f). The lattice points are matched one line at a time in images taken from different viewpoints establishing correspondences (g).

estimated corresponding lattices must present simple canonical properties derived from definition in figure 5.3, such as line parallelism, no lines intersect more than once, every node has no more than four neighbors, the grid points are not all collinear and the correspondences satisfy a planar homography. These sanity checks prevent erroneous matches arising from false seed points or from degenerate lattices. Multiple lattice correspondences are presented in figure 5.7.

Limitations The proposed method does not provide a correspondence for lattices of very skewed facades where the grid corners are starting to merge due to narrow viewing angles (see figure 5.23), and also for camera configurations where epipolar lines and lattice lines coincide leading to multiple possible solutions. The method works well for large lattices and the smallest ones it handles in experiments have in the order of six to eight points (see figure 5.6). Note, matching features in very oblique views is difficult in general and is not a particular problem of the proposed algorithm. The

limitation of alignment of grid lines and epipolar lines is addressed in section 5.5 through three-view geometry.

5.1.1 Motivation for modeling planar surfaces

Planar geometry is very common in urban scenes and, under viewpoint changes, it preserves

Property 1: the collinearity of features,

Property 2: the relative ordering of a set of collinear features, and,

Property 3: the local neighborhood.

The local neighborhood represents the surrounding features of points on a plane, which are invariant to viewpoint changes, in ways unlike on surfaces with 3-d relief. These preserved properties are exploited to match a rich set of strong essentially identical features that are too ambiguous to match individually without a region model.

The model-free proposed matching pipeline of chapter 4 reliably matches repeating features only if they are not too close together as building windows are in figure 5.1. The topological constraints of the match topology module in section 4.4.10 is not enough to handle such cases, as illustrated in figure 5.8a. Although there are several correct matches in the ambiguous grid structure of figure 5.8a, many are also incorrect or missing, especially for grid points in the interior of the grid, where matching is more ambiguous and therefore more sparse than at the borders. Estimated matches of a subset of features can be incorrectly shifted together on the grid and still preserve neighborhood topology, as seen in figure 5.8a. The match topology module is not designed for such sparsity of correct matches, and, as it is highlighted with a dashed red ellipse, some matches moved together to a wrong location while preserving neighborhood topology. An incorrect shift of a match is possible due to match sparsity causing poor neighborhood estimation, in addition to an isolated wrong match that attracts others when enforcing match topology.

The challenge of matching a grid of dense similar features is the location uncertainty that can only be resolved via global constraints. Note, matching similar features can be disambiguated using the method of chapter 4 if the repetitive features are not very dense, *i.e.*, nearest features are not within a few pixels away from each other. The matching problem can be even more challenging

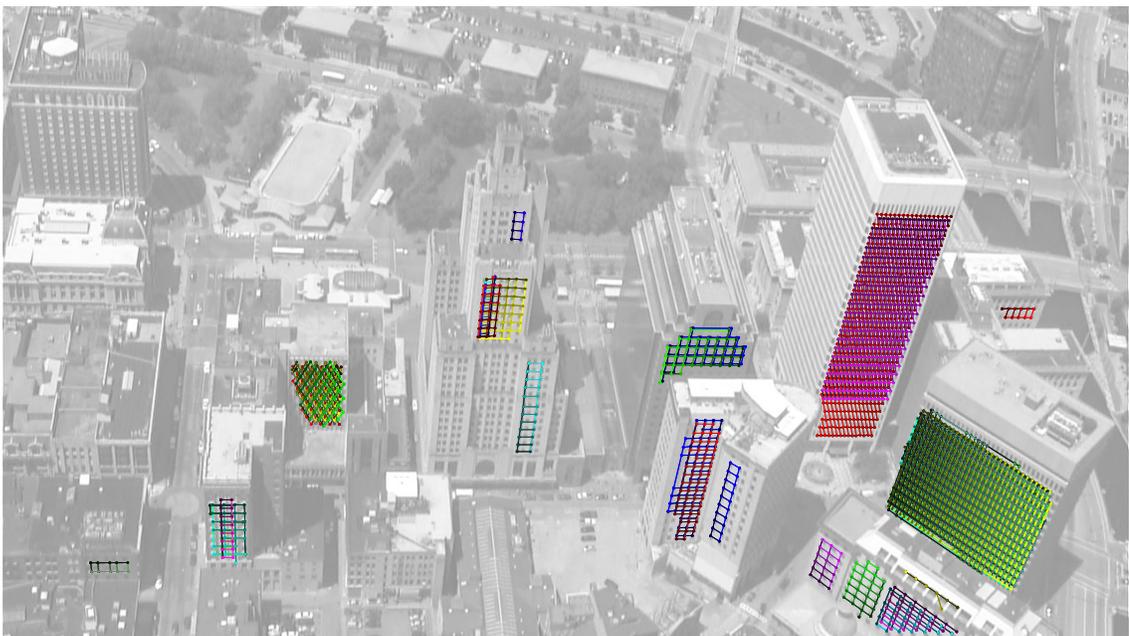


Figure 5.6: Multiple estimated lattices of a reference image that produce correspondences in a target image. The two processed images have equal sizes of 1280×720 pixels and the matching results are given in figure 5.7. Other detected lattices that have no visible correspondence in the target image are omitted, as the one from the tall building in the top-left corner, whose estimated lattice is shown in figure 5.19. A building facade may have multiple lattices, one for each distinct repeating corner from windows because windows are seen as small blobs with possibly multiple corners.

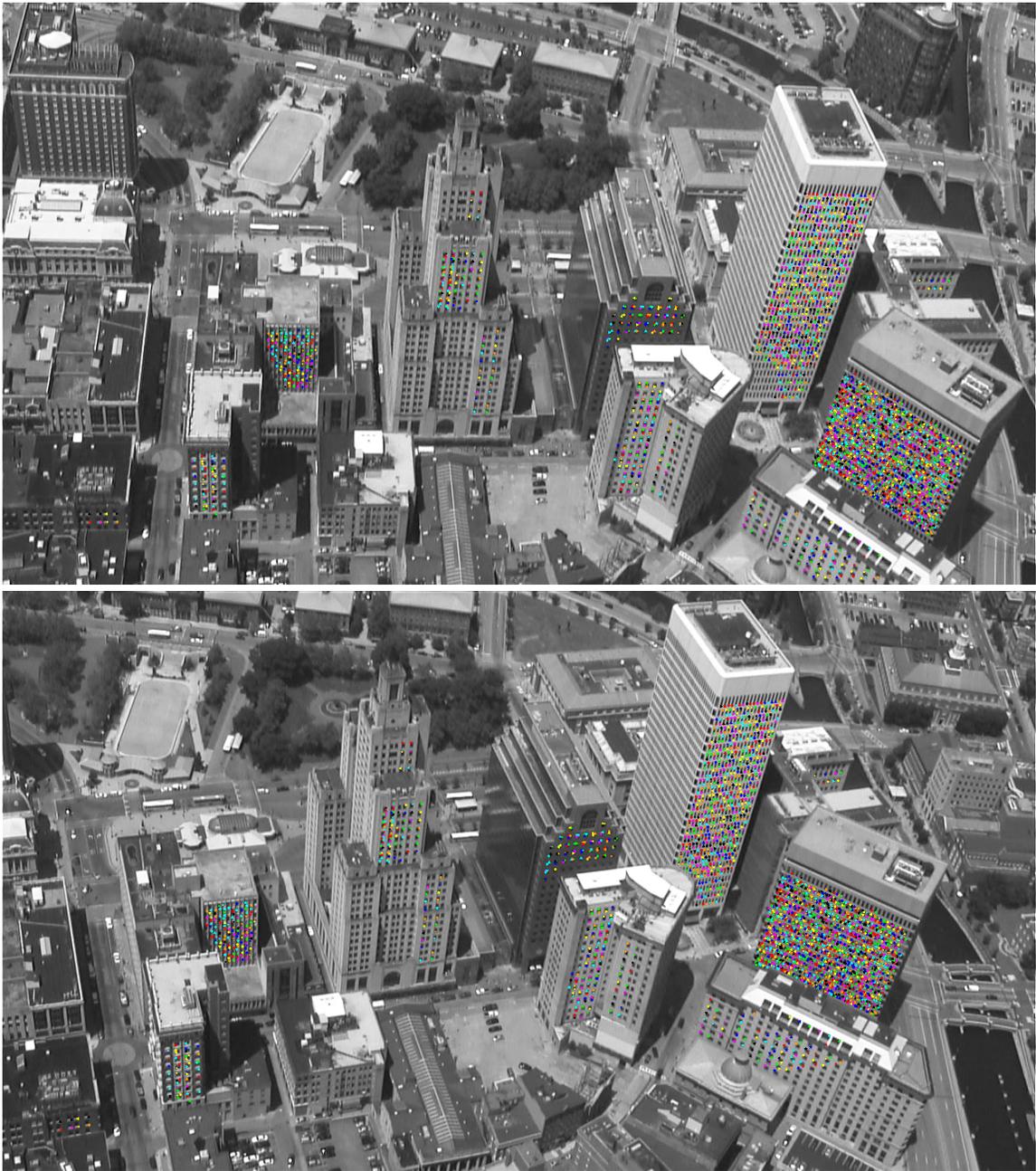


Figure 5.7: Multiple lattice correspondences found in a single image pair. The smallest matched lattice dimensions are 2×4 . The largest ones comprise hundreds of matches.

under major occlusion since shifted solutions become possible even under global constraints. For instance, a dense grid of building windows can be erroneously matched to a location shifted by two windows, as illustrated in figure 5.8b. Note, the building facade has more points than the ones shown in figure 5.8b because some were deliberately removed to simulate the effect of a partial building occlusion and illustrate the possibility of incorrectly shifted grid matches. The true match of the grid is shown in figure 5.8c. Both solutions approximately satisfy the epipolar geometry. It is difficult to disambiguate a point independently, but is easier to detect a global biased mean error on the shifted grid. Detecting the bias then requires the grid to be estimated and matched.

The proposed matching of dense repetitive features that uses a planar model, illustrated in figure 5.8c, succeeds for an entire grid of repeating points under simulated occlusion and justifies the use of a distinct plane-based matching method tailored for this type repetition.

5.2 Grid points detection

A *grid* denotes a collection of nearly identical feature points called *grid points* or *clones* originating from a planar lattice in a 3-d scene. The clones in the planar grid lattice are evenly distributed displaying evident collinear subsets. Any clone can be a *seed* to a procedure that detects the grid for which it is a point. A result of the proposed grid detection method is shown in figure 5.5e.

Note, notation from tables 4.1 and 4.2 are used in the following sections.

5.2.1 Finding grid seeds

Interest points are detected in an image as in chapter 4. In order to find an interest point that is a potential grid seed, correlate every interest point \mathbf{x}_R in the reference image with its own local neighborhood (in the same image) using normalized cross-correlation as in section 4.3.1 with the quality level parameter set to a high value $Q_m = 0.9$. Q_m is presented in equation (4.13). The returned matching peaks are nearby features resembling the interest point, *i.e.*, *clones* of \mathbf{x}_R . The search areas are small squares around each interest point. The size of the square S_{sq} is chosen as

$$S_{sq} = 0.05 \cdot \min(\text{width}(I_R), \text{height}(I_R)), \quad (5.1)$$

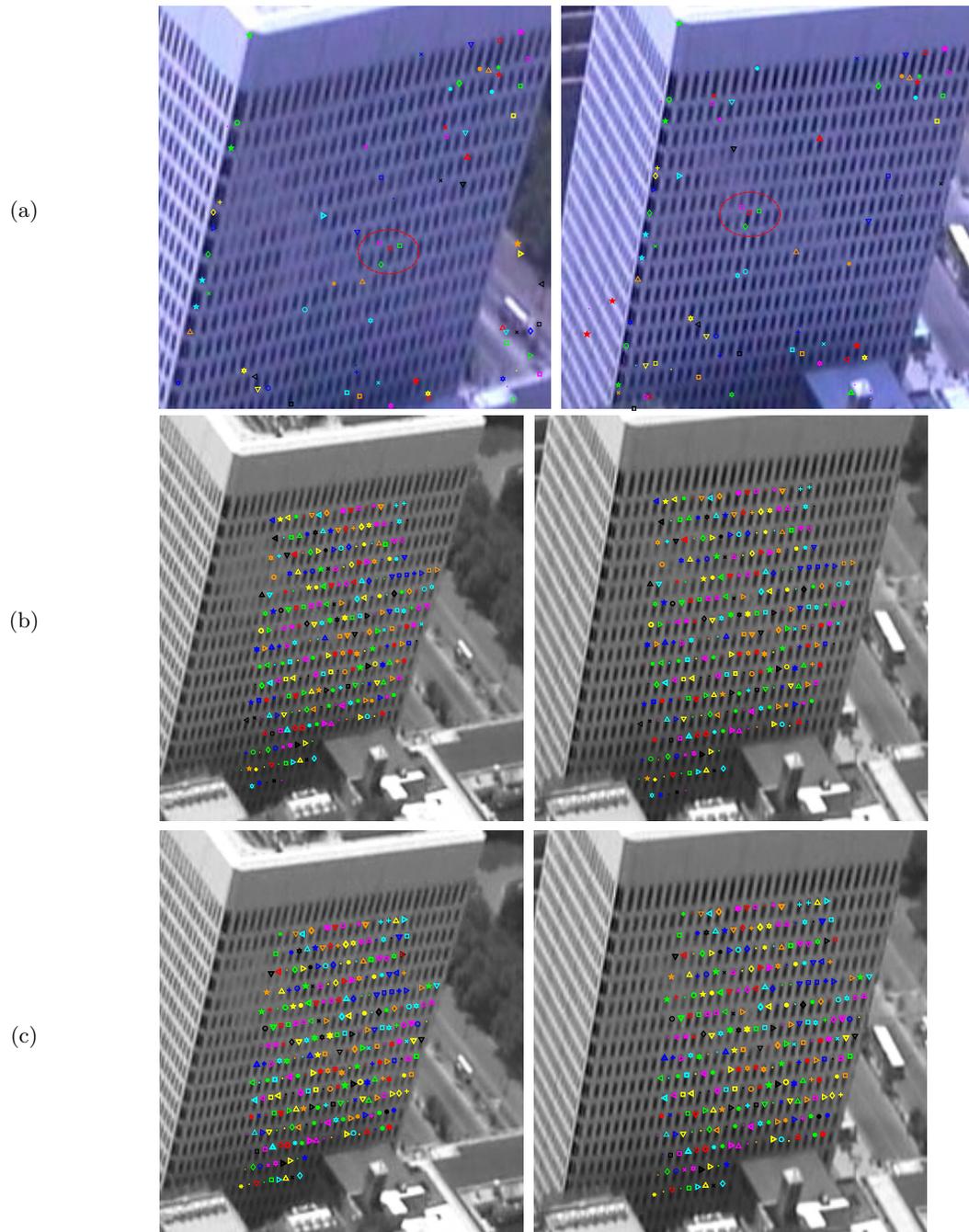


Figure 5.8: Matching results for highly repetitive urban scene features. (a) Proposed model-free matching using epipolar geometry, multiple view disambiguation and match topology module of chapter 4. Matching is sparse everywhere on the dense grid region and is reliable only at the borders. (b) A possible incorrectly shifted match of an entire patch of features. Some interest points were deliberately removed to allow illustrating shifted matches. (c) Matching using proposed planar model of this chapter. Matching is correct even when a shifted solution is possible.

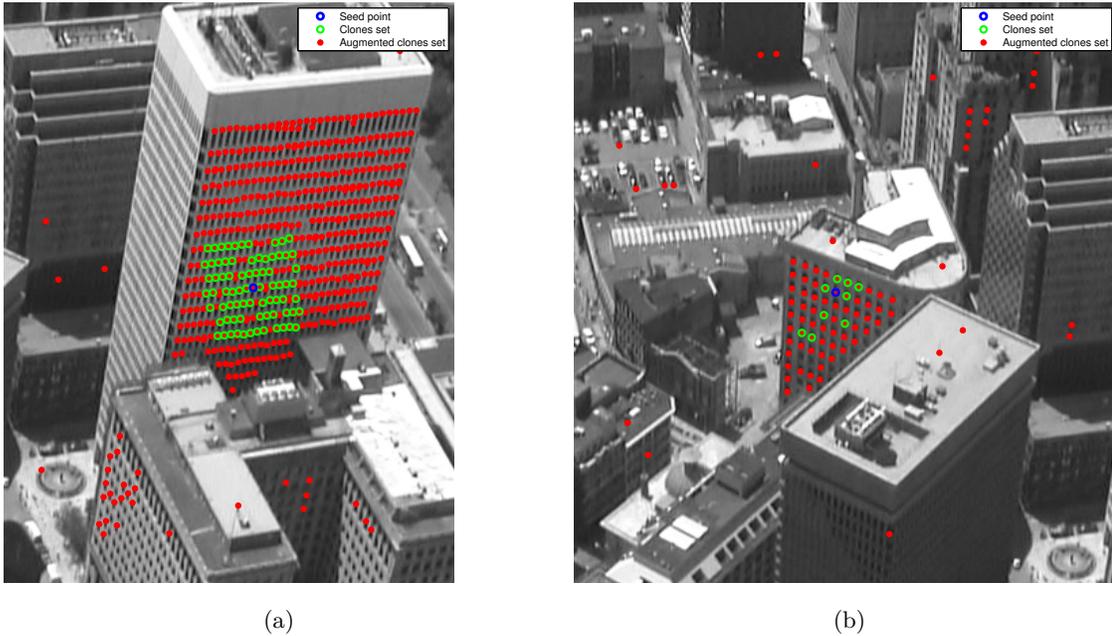


Figure 5.9: Illustration of two grids with grid seeds and clone sets. The grid seed points are in blue, their clones, composed of nearly identical features, are in green, and the augmented clones sets are in red. Not every true clone is detected and outliers are present both within and outside the grid area. On the left, clones are detected despite specular reflections disrupting their appearance.

which is a small fraction of the image size. Points with a high number of clones (at least 9 within the square) are putative repeating grid features and potential *seeds* for an underlying grid structure. The small size of S_{sq} is effective in finding potential seeds and at very low computational costs.

5.2.2 Augmented clones set

A grid seed is a point that presents a large number of clones in its local neighborhood. In order to find the entire grid, a new correlation search is performed for the seed only, this time in a larger neighborhood. The new search area is a square 5 times larger than the one in equation (5.1), a large portion of the image. The quality level parameter is relaxed to

$$Q_m = 0.75 \tag{5.2}$$

to find clones under illumination changes such as specular reflectance and other variabilities. The matching peaks are then points in the reference image that compose a planar grid and some outliers.

The new set is denoted the *augmented clones set* and includes potentially more clones than the original clones set, as shown in figure 5.9. Not every clone is necessarily detected and outliers exist, some also forming a secondary and less evident grid on nearby buildings. Outliers exist within and outside the grid region.

In figure 5.9a, bright surfaces of the scene reflect off the glass windows of the taller building where clones are detected. A different viewpoint is shown on the top-left image of figure 5.1 illustrating the reflection motion. Nevertheless, most clones are found even when specular illumination is reducing feature contrast, possibly due to the normalization of the correlation matching.

The augmented clones set has outliers, is unordered and unorganized. The method for detecting true grid points and grid structure is explained next and must be robust to outliers and some missing features.

5.2.3 Lattice model

A Delaunay triangulation \mathcal{D}_t of the augmented clones set is computed and there is likely outliers present in the set, as in figure 5.10. Delaunay triangulation is discussed in section 3.3.

On inlier regions, the edges of the triangulation are expected to be periodic as the lattice and therefore predictable, however this is not guaranteed since a Delaunay triangulation is sensitive to some point configurations and present multiple periodic patterns, as shown in the left image of figure 5.11 (one diagonal edge has different orientation than the others despite point regularity). Nevertheless, the proposed grid detection method is robust to a few distinct regular patterns. Near outliers, the triangulation edges are often in an irregular spatial configuration (see figure 5.11). Outliers are filtered out by learning the periodic patterns of the triangulation edges and removing points that do not fit these patterns.

Lattice feature. The learning of the lattice structure is based on the feature vector

$$\mathbf{E} = \begin{pmatrix} \theta \\ l \end{pmatrix} \quad (5.3)$$

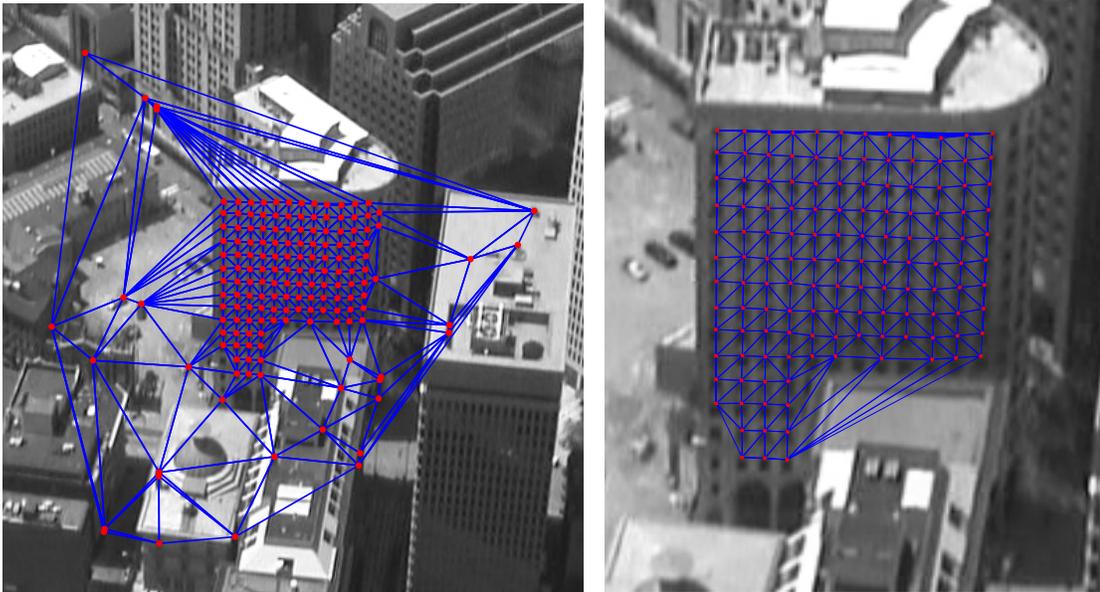
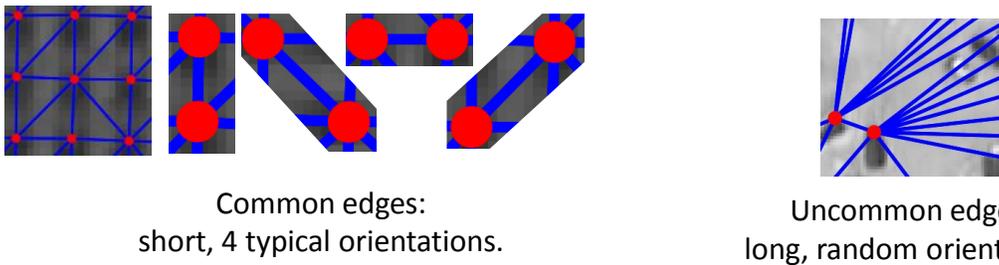


Figure 5.10: Example of Delaunay triangulation of grid features. *Left*: a triangulation of the augmented clone set of a grid seed showing potential grid points and some outliers. *Right*: triangulation of estimated inliers.



Common edges:
short, 4 typical orientations.

Uncommon edges:
long, random orientation.

Figure 5.11: Example of common triangulation edges due to actual repetitive and periodic features, and example of uncommon edges due to outliers, where all edges were taken from the left image in figure 5.10.

composed of the orientation θ and the length l of an edge of \mathcal{D}_t . The orientation angle and the length of an edge $\mathbf{e} = (e_x, e_y)^\top$, are illustrated in figure 5.12 and defined as

$$\theta = \arctan \frac{e_y}{e_x}, \quad (5.4)$$

$$l = \sqrt{e_x^2 + e_y^2}. \quad (5.5)$$

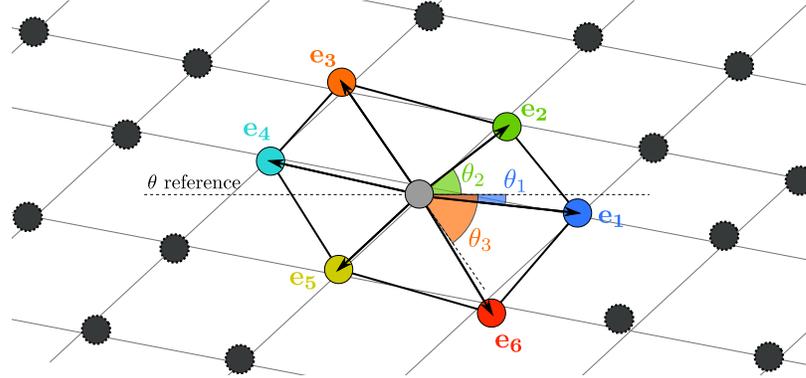


Figure 5.12: Illustration of feature vectors of Delaunay triangulation edges of a grid lattice point shown in gray at the center. Six edge vectors are shown: \mathbf{e}_i for $i = \{1, \dots, 6\}$. The orientation components of the edge feature vector \mathbf{E} are displayed for the first three edges as θ_1 , θ_2 and θ_3 . The length component is represented by the arrow lengths. By definition, the orientation feature for \mathbf{e}_1 and \mathbf{e}_4 are approximately the same small negative angle θ_1 . Similarly for \mathbf{e}_2 and \mathbf{e}_5 orientations as θ_2 , and for \mathbf{e}_3 and \mathbf{e}_6 as θ_3 .

Therefore, $-\pi/2 \leq \theta \leq \pi/2$. Thus, edge vectors for neighbors that are in approximately opposing directions, *e.g.*, Θ_e and $\pi + \Theta_e$, are represented by a single orientation, which is desirable, since opposing directions are interchangeable for representing a single lattice edge direction.

Lattice distribution. In order to derive the lattice structure from the triangulation \mathcal{D}_t , assume the distribution of the edge feature vectors \mathbf{E} follows a 2-d mixture of Gaussians density function, $f(X|\Theta)$. The vector Θ represents the mixture parameters: Gaussian weights w_k , mean vectors $\boldsymbol{\mu}_k$ and full covariance matrices $\boldsymbol{\Sigma}_k$ (k indexes Gaussian mixture components). The mixture $f(X|\Theta)$ is given by

$$f(X|\Theta) = \sum_{k=1}^K w_k g(X|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad (5.6)$$

$$g(X|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{|2\pi\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(X-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(X-\boldsymbol{\mu})}, \quad (5.7)$$

$$\sum_{k=1}^K w_k = 1. \quad (5.8)$$

The mixture parameter vector is

$$\Theta = \{\theta_1, \dots, \theta_K, K\}, \quad (5.9)$$

where $\theta_k = \{w_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$.

The mixture component parameters and the unknown number of components K are learned incrementally according to online update equations based on [68, 96]. The update equations given an observation \mathbf{E}_t are as follows:

$$w_{k,t+1} = (1 - \alpha_t)w_{k,t} + \alpha_t q_{k,t}, \quad (5.10)$$

$$\boldsymbol{\mu}_{k,t+1} = (1 - \rho_{k,t})\boldsymbol{\mu}_k + \rho_{k,t}\mathbf{E}_t, \quad (5.11)$$

$$\boldsymbol{\Sigma}_{k,t+1} = (1 - \rho_{k,t})\boldsymbol{\Sigma}_k + \rho_{k,t} [(\mathbf{E}_t - \boldsymbol{\mu}_{k,t+1})(\mathbf{E}_t - \boldsymbol{\mu}_{k,t+1})^\top]. \quad (5.12)$$

where the time-varying gain α_t , the posteriori probability $q_{k,t}$ of the k -th mixture component and the learning parameter $\rho_{k,t}$ are

$$\alpha_t = 1/t, \quad (5.13)$$

$$q_{k,t} = p_{k,t} / \sum_{j=1}^K p_{j,t}, \quad (5.14)$$

$$\rho_{k,t} = \frac{\alpha_t q_{k,t}}{w_{k,t+1}}. \quad (5.15)$$

where the likelihood $p_{k,t}$ is

$$p_{k,t} = w_{k,t} g(\mathbf{E}_t | \boldsymbol{\mu}_{k,t}, \boldsymbol{\Sigma}_{k,t}) \quad \forall k=1, \dots, K. \quad (5.16)$$

Note, $\alpha_t = 1/t$ is chosen for an even contribution from each sample, integrating them over time through the update equations. If α_t was constant, it would emphasize more recent samples of \mathbf{E} over older ones, which would be undesirable since all edges are equally important and their ordering is not meaningful. The number of mixture components K does need to be provided because it is adaptively learned and updated throughout the online learning, *i.e.*, new components are generated when current ones cannot explain a new observation. The online update equations update all Gaussian components simultaneously based on their posterior probability w.r.t. to each sample [68], hence use no thresholds to determine which Gaussians are updated at every iteration.

Learning the lattice. The truly periodic features must be very common and produce strong Gaussian components, while the outliers are normally random and generate none. If there are multiple periodic patterns on \mathcal{D}_t , they are also learned.

An initial pruning is performed prior to the learning. Triangulation edges define triangulation neighbors. The number of edges connected to a vertex \mathbf{x}_R in \mathcal{D}_t defines its number of neighboring

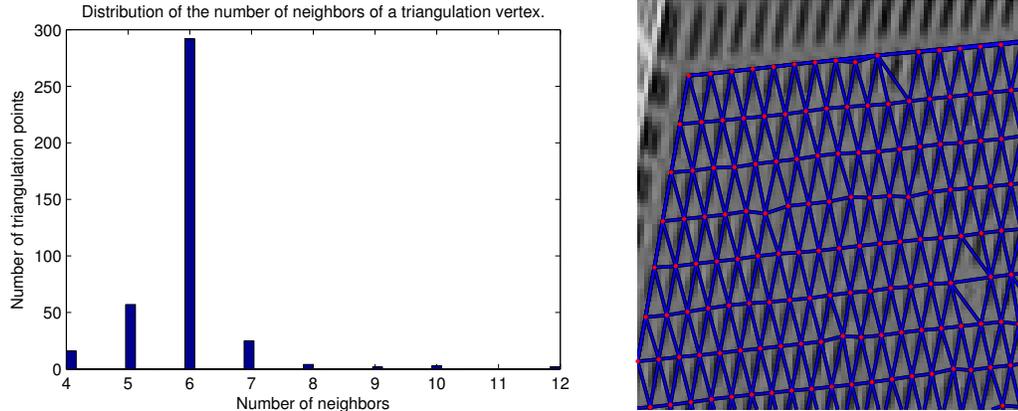


Figure 5.13: Distribution of the number of neighbors of a vertex of a Delaunay triangulation of a typical facade. Border vertices tend to have an unusually high number of neighbors (zoom in for details).

vertices, which is often 6 for a Delaunay triangulation (see section 3.3 and figure 5.13). To reduce the amount of outliers on the training samples, only the edges connected to vertices that have the most common number of neighbors N_n (computed as the mode) are used and N_n is often 6. The number of Gaussians components learned is up to $K = N_n$.

Only the Gaussian components with highest weights that jointly have at least 80% of the weight are preserved, others are discarded and weights renormalized. Hence, when multiple periodic patterns arise in \mathcal{D}_t , all the common contributions are retained and spurious ones are discarded. Experiments show that N_n neighbor edge orientations profiles are frequently learned, but essentially only three are genuine and preserved with enough joint weight, representing one periodic pattern with an emblematic hexagonal shape as in figure 5.12.

5.2.4 Iterative outlier removal

Given the learned mixture of 2-d Gaussians, each vertex \mathbf{x}_R in \mathcal{D}_t is tested to check if all its edges are in accordance with the mixture. The most common variety of nongrid outliers are detected to clean the triangulation from abnormal vertices that do not conform with the learned lattice.

A discriminant value Z_e is computed for each edge e of \mathbf{x}_R as the minimum of the Mahalanobis

distances of the edge feature to each mixture component:

$$Z_{\mathbf{e}} = \min_k \sqrt{(\mathbf{e} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{e} - \boldsymbol{\mu}_k)}. \quad (5.17)$$

The discriminants are used in an iterative pruning procedure that finds and removes highly inconsistent outlier vertices a few at a time, update \mathcal{D}_t and repeat. The $Z_{\mathbf{e}}$ quantity expresses standard deviations away from the mean. In theory, $Z_{\mathbf{e}} < 2.5$ represents approximately 95.6% of the volume under the 2-d normal density function used here.

An outlier on the triangulation interior affects the triangulation on its surroundings potentially causing any of its neighbors to also appear as an outlier. The triangulation must be updated to reflect the removal of outliers. The simple removal of outliers is not a final solution since they may still leave holes in the lattice causing new abnormal edges due to missing grid points. The proposed procedure copes with this problem to prevent an outlier from iteratively spreading its outlier condition outward as \mathcal{D}_t is updated, an issue denoted *outlier proliferation*. The solution is provided below.

Finding nongrid outliers. At each iteration, vertices such that a minimum number of their edges agree with the Gaussians mixture by having a discriminant less than 2.5, *i.e.*,

$$Z_{\mathbf{e}} < 2.5, \quad (5.18)$$

are considered inliers. The number of agreeing edges must be at least $N_n/2$, which often means 3 or more as the average number of edges N_n is often 6. Vertices at the borders and corners of the lattice need special treatment as some of their edges differ from the edges of interior vertices. To preserve border vertices and avoid outlier proliferation, some neighbors of the inliers are also considered inliers and protected from removal at each iteration. The protected inlier neighbors are the ones associated with edges such that their discriminant is less than 2.5. Remaining vertices are *nongrid outlier candidates* and are not promptly discarded, but must satisfy an inconsistency test to be considered an outlier.

A candidate is an outlier if 60% of its edges discriminants (Mahalanobis distances) are greater than 5, *i.e.*,

$$Z_{\mathbf{e}} > 5. \quad (5.19)$$

This condition is based on the insight that outliers usually have many abnormal edges that are statistically away from the learned model. However, some true lattice points may also have a few abnormal edges in case of neighboring outliers or missing neighboring lattice points, as seen in figure 5.10. In addition to nongrid outliers, another type of outlier is also detected and removed, the redundant *immediacy outliers* described next.

Immediacy outliers. Small image perturbations may induce the correlation matcher to detect a repeating feature point twice in a location only a few pixels apart, giving rise to another type of outlier, the *immediacy* outlier. These outliers are noticeable on the left image in figure 5.9 and are detected by a proximity test: if the nearest vertex is closer than a minimum distance, the point (or maybe the neighbor) may be an immediacy outlier. The minimum distance threshold l_{min} is extracted from the learned mixture of Gaussians as the lowest second component of all mixture means. The component is the typical length component l in equation (5.3), then l_{min} is the minimum average distance among lattice points. In order to detect immediacy outliers, the bound l_{min} must be reduced further to handle statistical variations, given that l_{min} is just a mean. Let τ_{imo} , such that

$$\tau_{imo} = l_{min} - 5\sigma(l_{min}), \quad (5.20)$$

be the decision boundary, where $\sigma(l_{min})$ is the standard deviation associated with l_{min} . $\sigma(l_{min})$ is taken from the mixture as the square root of the second diagonal component of the covariance matrix associated with l_{min} . Then, it is reasonable to assume that points within τ_{imo} pixels from each other are immediacy outlier candidates. When points are deemed too close together according to τ_{imo} , they must be removed except for one. Keeping all points around would be problematic for subsequent processing on the grid lattice. Furthermore, the correctness of the preserved point in representing the most accurate lattice node is unimportant, as the accuracy of matching it in another image is what actually matters and the lattice estimation is robust.

Among any cluster of immediacy outlier candidates, nearby points according to τ_{imo} , a vertex is unconditionally considered an immediacy outlier (regardless of being considered an inlier by other criteria) if its average discriminant is larger than that of its nearest neighbor.

Figure 5.14 displays the iterative outlier removal method, showing inliers, outliers and the

triangulations. Outlier candidates that are not deemed to be estimated outliers are labeled “undefined”. Many immediacy outliers are shown in green in the top-left image of figure 5.14 in the interior of the grid next to inlier points.

Termination condition. After each outlier removal iteration, the structure of \mathcal{D}_t is locally rearranged with the outliers removed and the procedure repeats until none is left, as shown in figure 5.14, resulting in estimated inlier triangulations as in figure 5.15. Note that nearby identical facades may exist causing multiple grids to arise, as in the bottom of figure 5.14. In order to remove similar nearby grids, consider ignoring all edges not in accordance with the learned model as in equation (5.19), and keeping only nodes connected to the seed via remaining valid edges.

5.2.5 Defining grid main directions

The main directions are any two directions in the grid where a lattice can be defined by connecting sets of collinear points parallel to each other. The directions are taken from the Gaussian mixture learned means μ_k , discussed in section 5.2.3, since μ_k has two components and the first represents a triangulation edge direction. There are multiple possibilities for the choice of directions, as can be seen in figure 5.15. The main directions are chosen as the two orientations from μ_k such that their relative angle is closer to 90 degrees (in the image). Note, the main directions follow from the structure of a Delaunay triangulation in image space and need not be identical to the world horizontal and vertical directions. In fact, relating scene and image directions is irrelevant for the feature matching purposes of this chapter.

5.3 Grid lattice estimation

Given estimated grid points, they must be connected in a consistent way in order to form a 2-d lattice, which consists basically of finding straight line connections along two main directions. Lattice lines must be estimated since the grid points alone are not useful to ease image matching disambiguation. As discussed in section 5.1.1, matching becomes more constrained and easier to disambiguate when the neighborhood topology is estimated for a planar surface. The topology includes the grid lattice

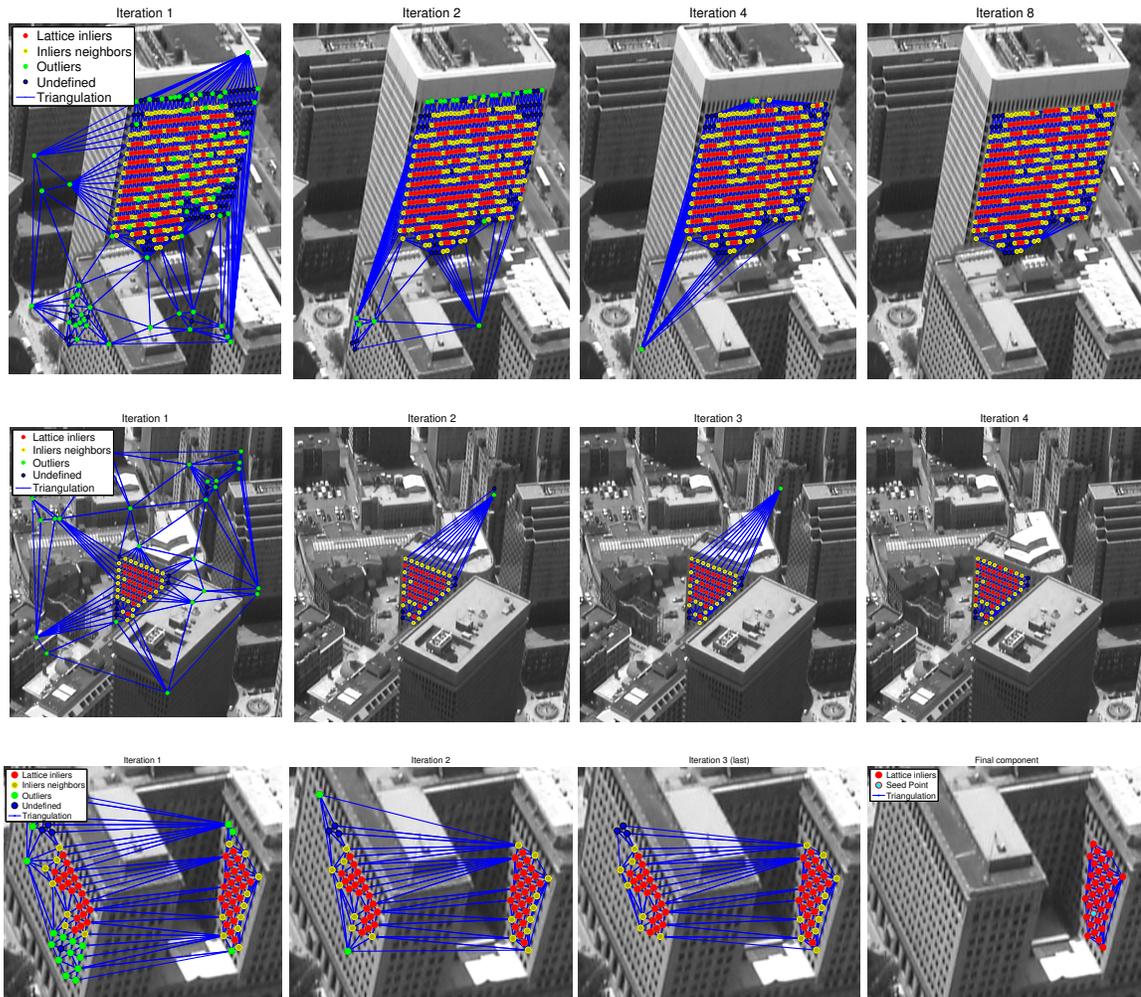


Figure 5.14: Iterative detection and removal of non-lattice points based on learned lattice spacings and angles. The process starts with the augmented clones set of a grid seed point. The removal process for the two buildings from figure 5.9 is illustrated in the top and middle rows, where the last iteration is shown on the right. Note, on the top row, the feature windows on the top floor are removed from the grid since those windows are larger and their spacing does not agree with the rest of the grid. The bottom row displays the three outlier removal iterations of a building that has a nearby identical facade and only the grid near the seed is preserved.

lines in two predominant directions, their ordering and the ordering of points within a line. This regular spatial topology provides grid point neighborhoods similar to that of image pixels: up, down, left and right neighbors.

To estimate lines from grid points, adjacent vertices are first connected in the two main grid directions, forming line segments (section 5.3.1). A single grid line can be broken into multiple

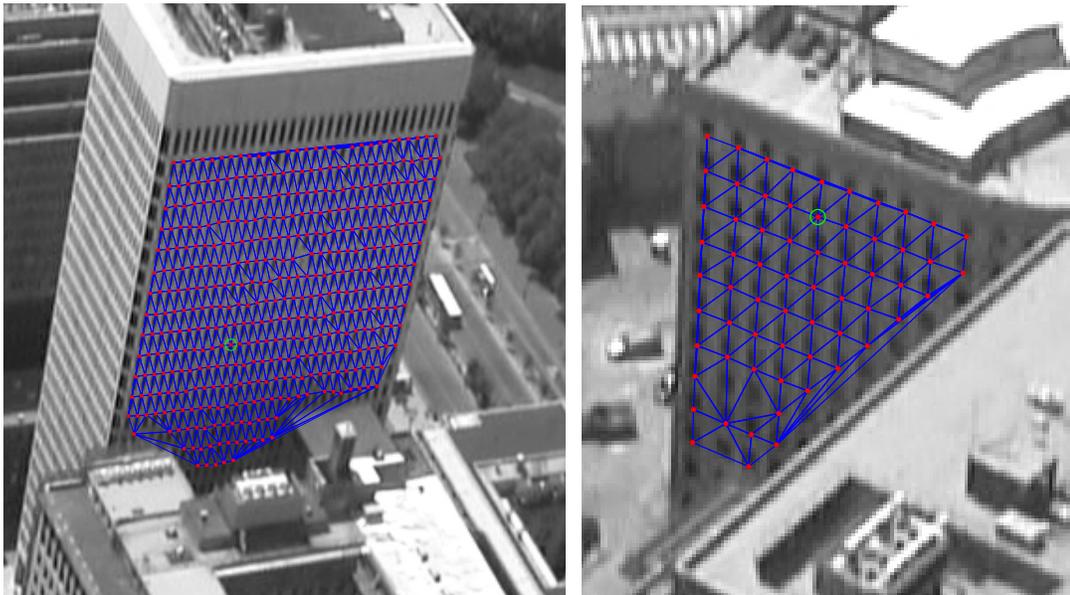


Figure 5.15: Examples of Delaunay triangulations computed from pruned estimated grid points, shown as red dots, with grid seeds shown as green circles. The triangulations, displayed as blue edges, depict the structure of the grid lattice.

segments if there are missing vertices or local statistical variability. Second, the segments are iteratively merged into complete grid lines (section 5.3.3).

5.3.1 Connecting vertices into line segments

In order to connect points into line segments in one of the main directions v , as in figure 5.16, every point is visited once. Given the first visited point p_1 , find its surrounding points according to \mathcal{D}_t and assign the neighbors along the direction v to be the two which are roughly in such orientation. One neighbor will be in one side of p_1 and the other on the opposite side, *i.e.*, towards v and $-v$, respectively. These neighbors are found via computation of the discriminants $Z_{\mathbf{n}}(v)$ of their edges \mathbf{e}_n according to the mixture component associated to v , denoted $\boldsymbol{\mu}_v$ and $\boldsymbol{\Sigma}_v$:

$$Z_{\mathbf{n}}(v) = \sqrt{(\mathbf{e}_n - \boldsymbol{\mu}_v)^\top \boldsymbol{\Sigma}_v^{-1} (\mathbf{e}_n - \boldsymbol{\mu}_v)}. \quad (5.21)$$

In order to be considered a neighbor point along v , the associated edge must satisfy

$$Z_{\mathbf{n}}(v) < 2.5. \quad (5.22)$$

Note, this decision process is very similar to outlier removal in equations (5.17) and (5.18). If there is more than one neighbor in each side, the ones with lowest discriminants are selected. This process iteratively repeats for the neighbors of p_1 , for their neighbors and so on, growing into a line segment of roughly collinear points. The growth ends at the borders of the triangulation or once no valid neighbor along v is detected.

Once a line segment is found, each one of its points, must not belong to any other line parallel to v . Thus, points assigned to a line, need not to be visited again. The segment estimation continues at other vertices that remain unassigned, until all have been visited and assigned a line segment. Isolated points are considered a line segment on their own (see figure 5.16a for examples) and may be merged to other segments in the merging stage (section 5.3.3).

The entire segment detection is repeated for the second main direction of the grid. Thus, every vertex is visited twice, once per main direction. The result of line segments detection is shown in figure 5.16. Note, the segment estimation is robust to small distances between the grid lines as shown in figure 5.16d.

5.3.2 Defining line segments neighborhood

The line segments provide a rough structure of the grid that is enough to find line-to-line neighborhoods. Given a line segment s_l in one direction, the two neighbor parallel segments, s_{l-1} and s_{l+1} , can be found by traversing through the other main direction, as shown in figure 5.17a. For each point in s_l , take which segment it belongs to on the other direction of the grid and traverse to its immediate neighbors. The majority of points in s_l will traverse to the same two parallel lines, which are immediate parallel neighbors of s_l . If more than two lines exist, the ones that occur more often are chosen as neighbors of s_l .

5.3.3 Merging line segments

Once all segments and their neighbors are established, they are merged to form longer and complete grid lines. The merging is based on distances from points to lines. Let s be line segment. From the points in s , a line is fit using SVD, resulting in a line l_s . The distances from the points in s to l_s are

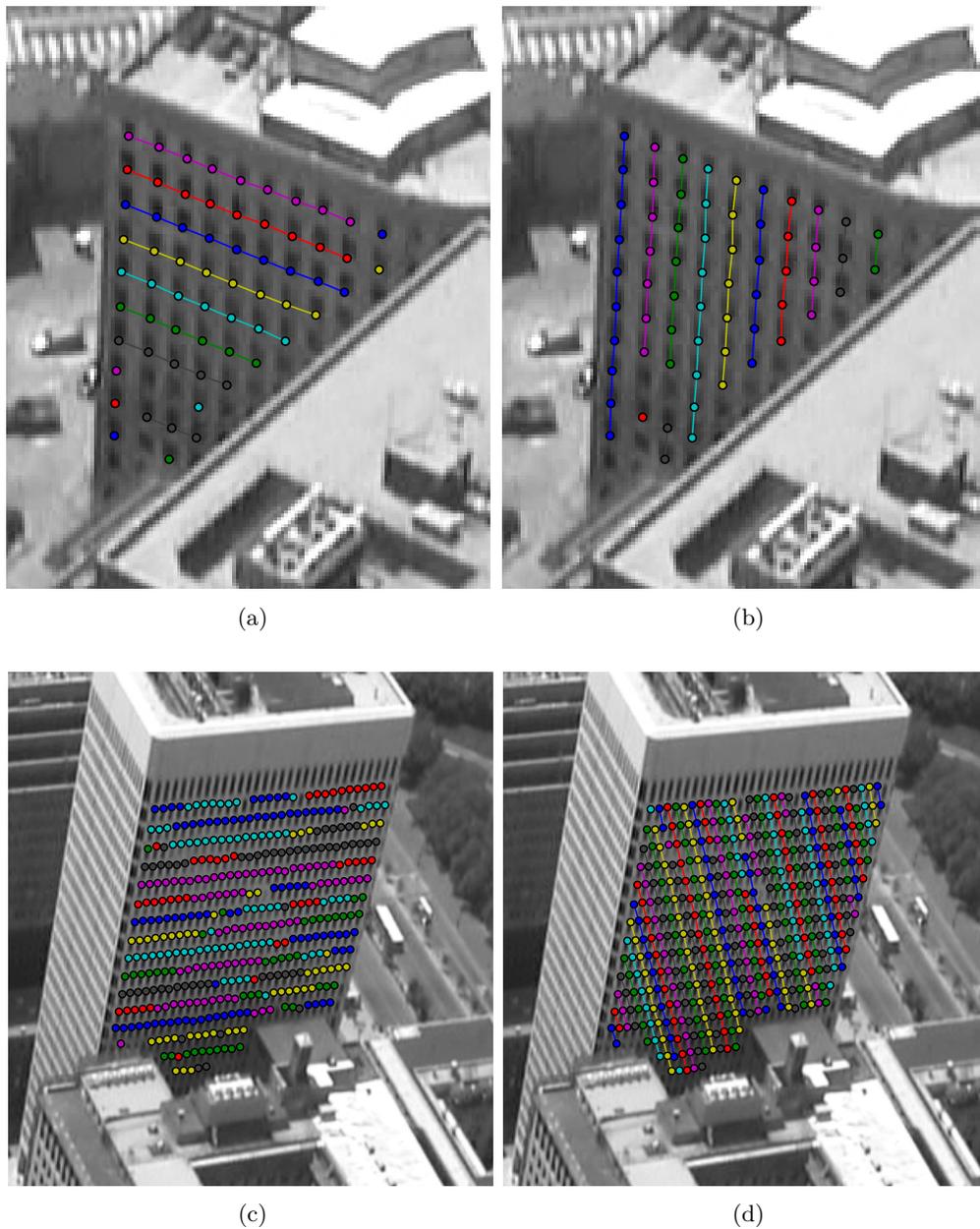


Figure 5.16: Line segments estimated for distinct grids of windows of two buildings. Segments are presented with random colors and in two learned grid directions that are roughly perpendicular. (a,c) Line segments in one main direction, and, (b,d) in the other main direction. In (d), the lines are much denser and not following the world vertical direction. A segment is estimated connecting neighboring grid points through edges of their Delaunay triangulation in a given direction. A single lattice line may split into multiple segments and their breaking points are due to missing features, missing triangulation edges connecting them or small statistical deviations from their associated learned straight line model.

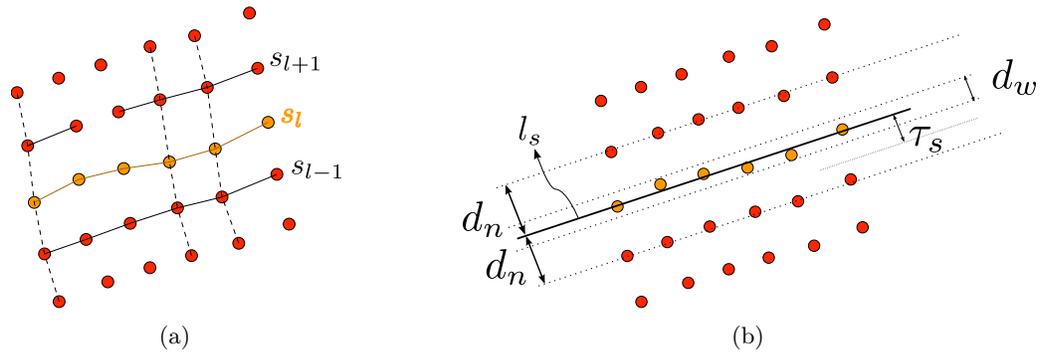


Figure 5.17: (a) Illustration of parallel neighbors of a line segment s_l and three approximately perpendicular segments (dashed). (b) Illustration of the decision threshold τ_s for points to belong to a given fit line l_s , and related variables, namely the within-line and neighbor-lines distances.

computed and their median is the *within-line* distance, $d_w(s)$. The distances between neighbors of s and l_s are also computed and their median is the *neighbor-line* distance, $d_n(s)$. These distances are illustrated in figure 5.17b. The mean of the two aforementioned median distances is a threshold τ_s ,

$$\tau_s = \frac{d_w(s) + d_n(s)}{2}, \quad (5.23)$$

representing the half-distance between a line and its neighbors, and τ_s is used to determine whether a point belongs or not to l_s . The threshold τ_s is also used for matching lines in section 5.4.3.

The distance from a segment to l_s is defined as the highest of all distances to l_s from the points composing the segment. Computing the distances from all segments to l_s , the ones such that the distance is at most τ_s are then merged to segment s . This process runs from the largest segments to the smallest ones incrementally merging them into larger sets of collinear points. Once a first pass through all segments finishes, the merging repeats to combine merged segments into even bigger ones until nothing can be combined anymore.

Singleton-point segments. Segments with a single point can be naturally assigned to merge with other larger segments of the same underlying grid line via the proposed merging method, but no line can fit to them. If all segments of an underlying grid line are of singleton points, merging would fail for lack of a fitting line. However, this is resolved by borrowing line slopes from neighbor segments and fitting a line with such slope to the singleton-point segments. This is illustrated applied on two

grid lines near the lower left corner of figure 5.16a that have only two singleton-point segments, but are successfully connected as seen in figure 5.18c.

5.3.4 Organizing lines into a 2-d lattice

The iterative segment merging process of section 5.3.3 is performed once in each main direction of the grid providing complete lines. The lines can be straightforwardly ordered given the estimated parallel line neighborhoods (section 5.3.2). The result is an estimated 2-d lattice, as shown in figure 5.18, where the ordering of lines is color coded: each line has a position-dependent color taken from a gradient changing smoothly from red to yellow. In addition, the ordering of points in each line is illustrated by connecting adjacent points with a straight line segment. Note, the merging is robust to a sequence of missing grid points as shown near the bottom-left corner of figure 5.18a. More results are shown in figure 5.19.

5.4 Grid matching

After a grid lattice has been established providing grid neighborhoods, the next step is to match the grid on a target image enforcing properties of planar surface projection to disambiguate the matching, as motivated in section 5.1.1. It is possible to either design a Markov Random Field or heuristics to do the matching. In this section, a grid matching method using heuristics is proposed to enforce neighborhood topology. A number of geometric constraints of planar lattices are enforced to disambiguate the matching and essentially eliminate match ambiguity.

5.4.1 Selection of grid points to match

Section 4.2 proposes to use corners as interest points to be matched via normalized cross-correlation in chapter 4. The reference image grid points detected in section 5.2 are different than the corners found via method in section 4.2, yet some of these interest points can be very close to each other, originating from the same world features. These corners at grid points are known to have poor match quality due to high ambiguity, as shown in figure 5.8a and are then discarded together with their existing unreliable matches from method in chapter 4. More precisely, any corner within a two pixel

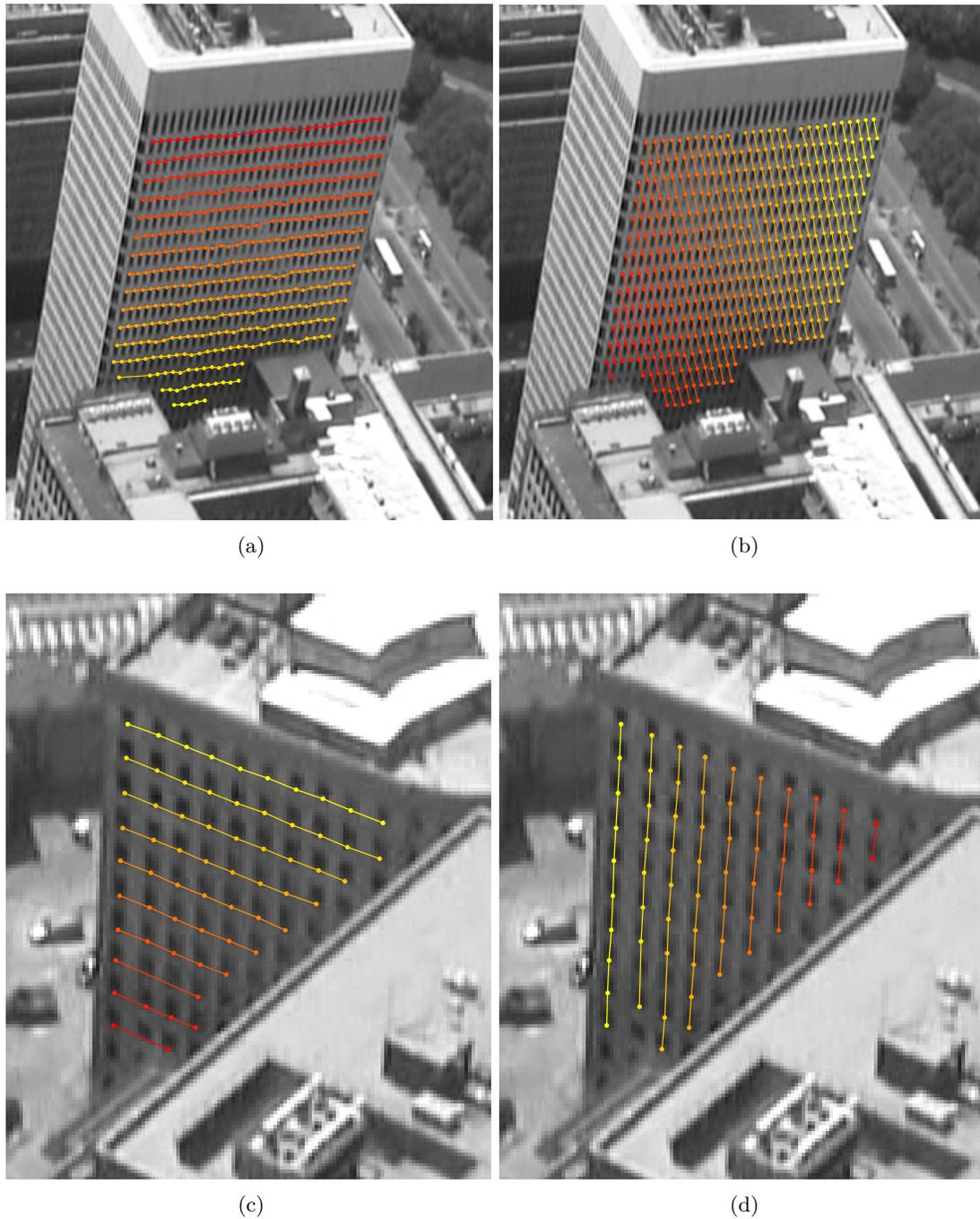


Figure 5.18: Grid lines detection from merging line segments. (a) Building 1 and lines in first main direction. (b) Building 1 and lines in second main direction. (c) Building 2 and lines in first main direction. (d) Building 2 and lines in second main direction. The sequence of dots within each line is shown through connected adjacent segments. The sequence of lines within the grid is shown through a colormap gradient changing in adjacent lines from red to yellow.

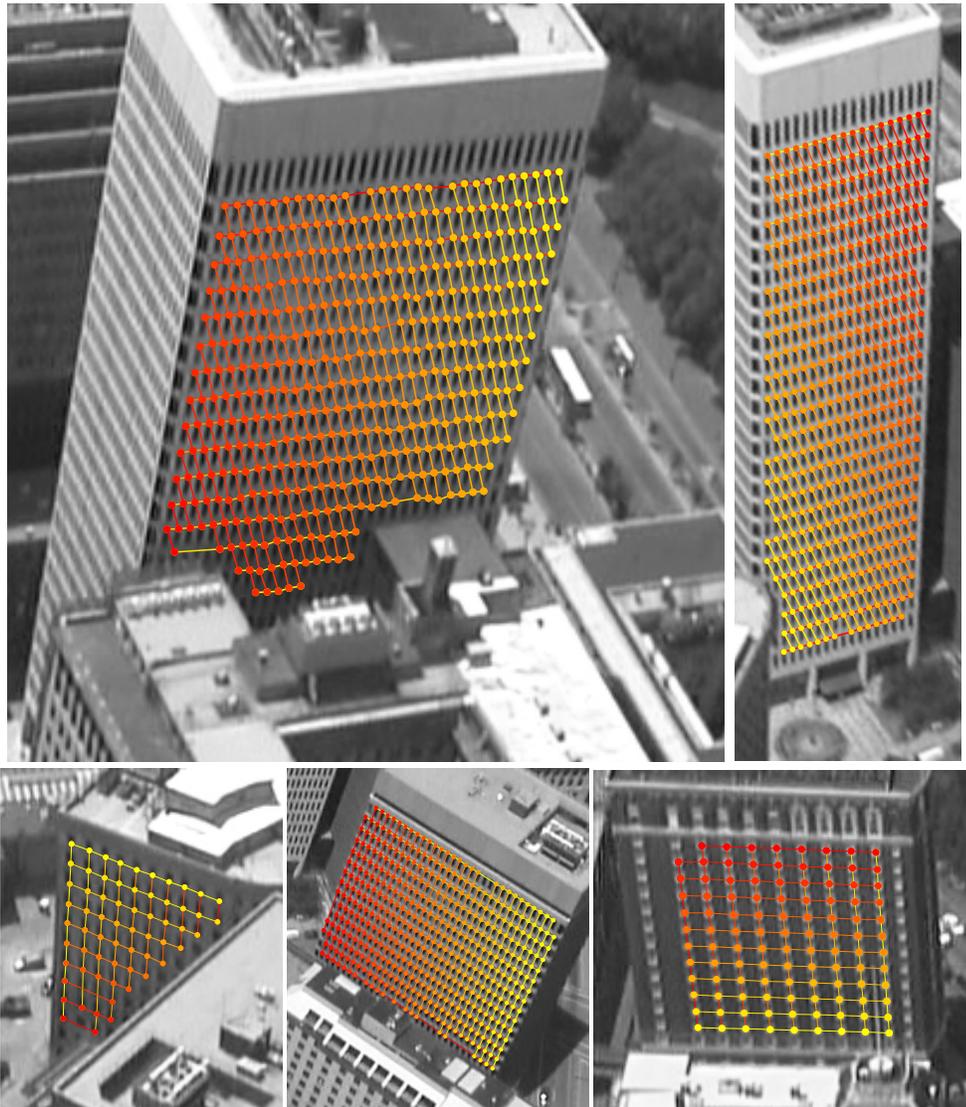


Figure 5.19: Examples of estimated 2-d planar grid lattices found in images from figure 5.1 using method proposed in section 5.3. Grid lines from two main grid directions, as the ones displayed in figure 5.18, are shown in a single image to depict the 2-d grid structure.

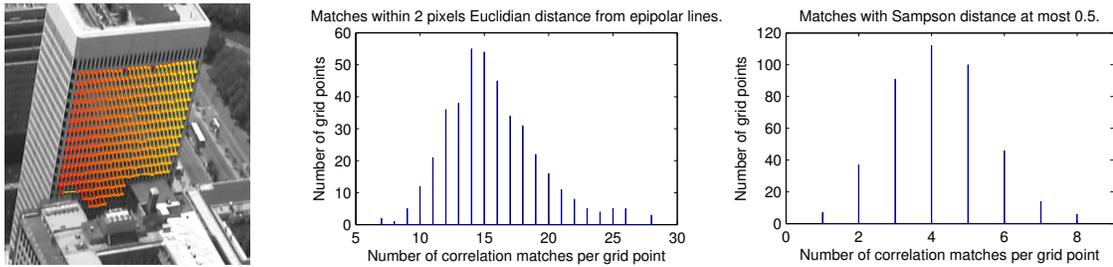


Figure 5.20: Histograms (*center and right*) of the number of correlation matches of repeating grid points (displayed in the reference image on the *left*). The correspondences are in a target image (not shown). The average number of matches within a 2 pixels Euclidean distance from associated epipolar lines is 15, and the average within 0.5 Sampson distance is 4 (see equation (4.24)).

distance to any grid point is discarded. Removed corners are replaced by the estimated grid points that can be matched reliably using the global constraints of a lattice.

5.4.2 Match grid points independently

Individual grid points are first matched from a reference to a target image using the same normalized cross-correlation procedure described before in section 4.3.1, retaining all high correlation peaks along epipolar lines. The same quality level used in equation (5.2) for the augmented clones set is used here, and the search area in rectified images is a band around the epipolar lines with 7 pixels of thickness. The epipolar geometry is estimated from other non-grid matched points from the image pair, estimated from method in chapter 4. Note, correlation matches are strictly defined as local peaks in a 3×3 neighborhood, then border correlation pixels are never considered peaks, and the effective strip thickness becomes 5 pixels. Thus, refined peak locations are at most 2.5 pixels away from the epipolar lines.

The average number of matches per grid point is high due to periodicity of the grid (see figure 5.20). The points are pruned, by keeping only the ones with maximum Sampson distance of 0.5. Sampson distance is defined in equation (4.24). Illustration of some point matches is provided in figure 5.21.

5.4.3 Reduce ambiguities by matching grid lines

Given that collinearity, ordering and neighborhood properties are preserved under viewpoint change in the case of planar surfaces (see section 5.1.1), these constraints can be used to rule out wrong

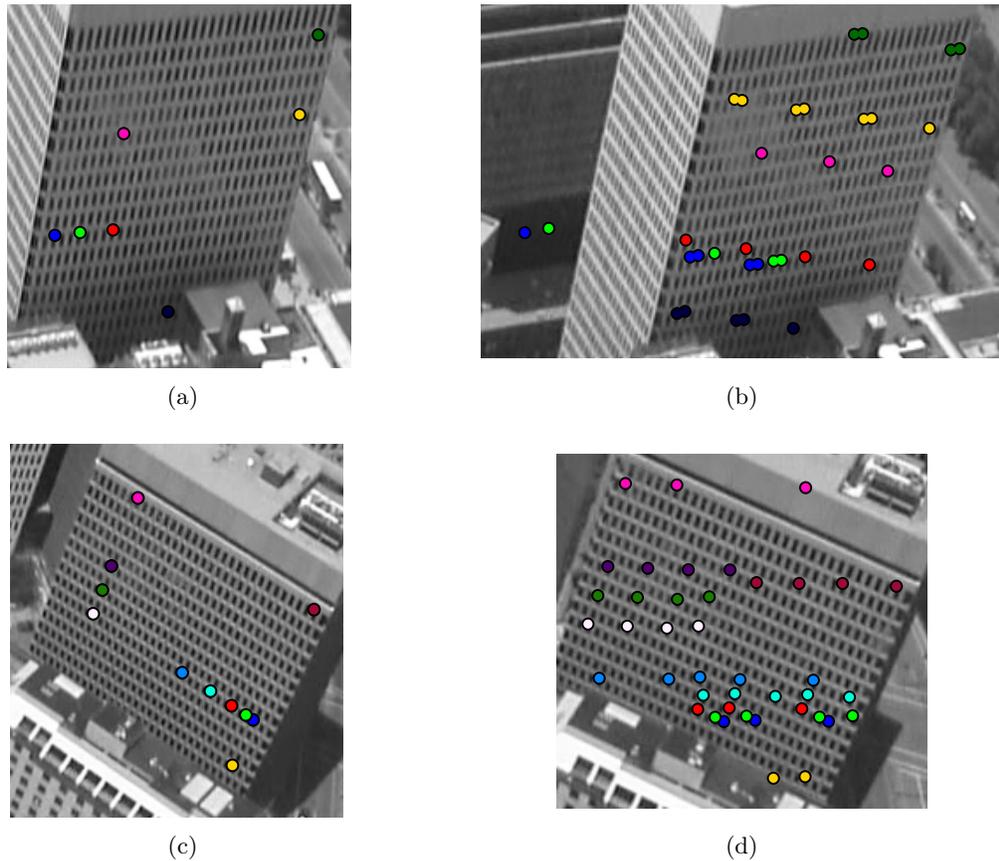


Figure 5.21: Examples of grid point matches computed individually. (a) Arbitrary feature points in a building, where the red, green and blue are collinear. (b) Matches of previous points in a different viewpoint. Note, the collinearity of points may be preserved even on incorrect shifted matches due to high periodicity of the grid and narrow angle between epipolar lines. (c) Arbitrary grid points on another building, where two subsets of points are collinear. (d) Matches of previous points in another viewpoint showing that collinearity alone does not completely disambiguate the line matching.

matches of grid points. Given all matches of all grid points, it is possible that there is only one matching configuration among all possible combinations that preserves the topology of the estimated grid. However, trying every possible combination is prohibitive. In figure 5.20 example, the grid has approximately 400 points with an average of 4 matches per point, *i.e.*, there are 4^{400} distinct match combinations for the grid. In practice, only one or a few configurations will be valid under the discussed planar constraints, but its infeasible to test them all. This search is reduced by assuming that the collinearity, ordering and neighborhood of grid points, which are already established in a reference image lattice, are preserved in the corresponding lattice. However, at this stage no information about

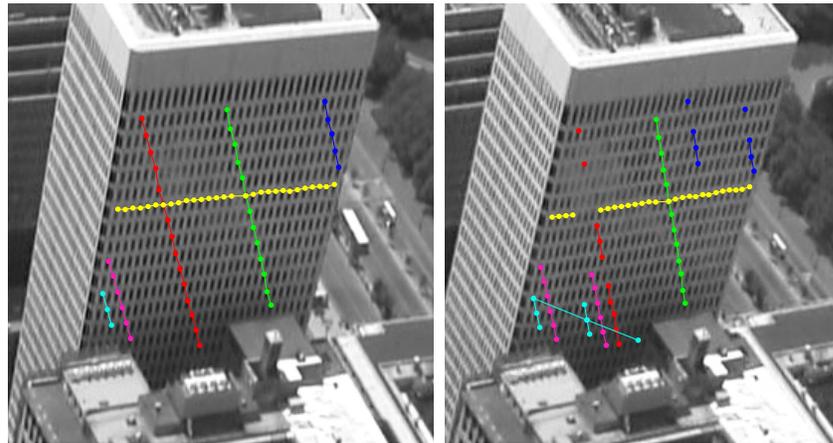
lattice structure in a target image is known. A solution for the lattice correspondence problem is found by attempting to reconstruct the reference lattice properties in the target image. For instance, match ambiguity is reduced by several orders of magnitude by first matching straight lines (collinear points) taken from the grid lattice.

Outline of line matching constraints. It is tractable to find valid matching lines by simply incrementally analyzing the matches of every estimated grid line point in the sequence in which they appear. Three constraints are implemented to reduce ambiguities and each require at least three matching points: *collinearity*, *ordering* and *spacing*. Let an estimated grid line have one endpoint with N_1 possible matches. Note, the grid line may have missing points due to occlusion or other phenomena, and, the endpoints of the estimated line are not necessarily the endpoints of the true complete line. Any of the N_1 matches of the given endpoint are feasible, according to the constraints. Any of the N_2 matches of the next point along the line are also feasible given the previous point, forming $N_1 N_2$ possible line segments. Then, the only valid matches of a third point are the few that jointly satisfy all three constraints with some of the $N_1 N_2$ segments, resulting in just a few viable match triplets. Repeating this process up to the other endpoint, the constraints may reduce the number of possibilities to essentially one, for a lattice line with a large number of points.

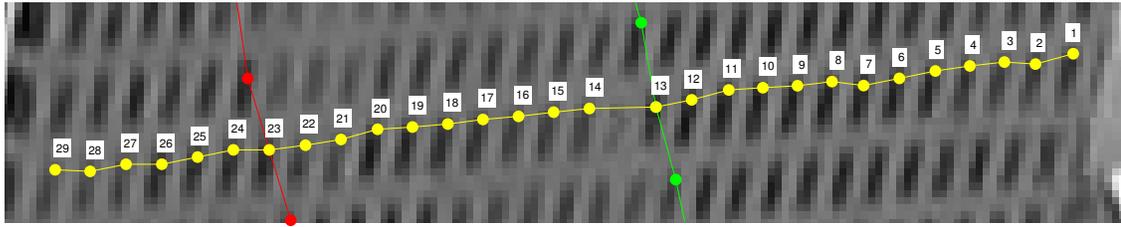
The proposed matching constraints are discussed in detail next and line matching results are shown in figure 5.22, illustrating the robustness of the matching to missing lattice points and to missing matches. Multiple line matches may exist, yet affecting mainly short grid lines. Disambiguation of multiple line matches is discussed in section 5.4.4.

Overview of collinearity and ordering constraints. Points from a planar grid lattice line in a reference image must appear in the same relative order in the target image, in addition to being collinear. It is possible that some may not appear due to occlusion or failure to match, but the *ordering* and *collinearity* constraints are still preserved for the visible ones.

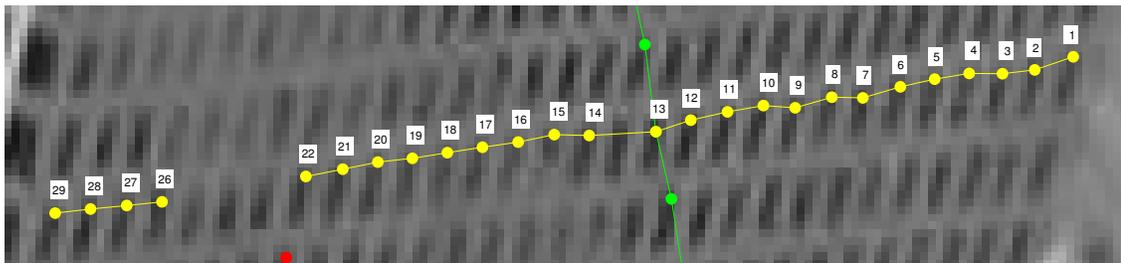
Overview of spacing ratio constraint. In order to enforce stronger geometric constraints besides collinearity and ordering, the spacing of the matches must also agree with the spacing of the grid points. The use of the spacing constraint may be redundant for long lines when collinearity and



(a)



(b)



(c)

Figure 5.22: Illustration of grid line matching. (a) Views of an aerial stereo pair showing grid lines in different colors on the left and their associated line match candidates on the right. The color of multiple matching candidates is the same color as that of the associated reference grid line. (b) Detail of the yellow horizontal grid line on the left image and (c) detail of its matching line on the right image. The numbers uniquely represent a line point and its match showing the match is pointwise correct. The line detector and matcher are both robust to local straightness perturbations on both images, robust to missing grid points as in going from point 13 to 14 and robust to missing matches (in sequence) as points 23, 24 and 25 exist in the grid line, but have no line matches. The absence of matches for line points is represented by disconnecting the line at such locations. In this example, missing matches are due to specular reflections from glass windows and are correctly skipped based on the spacing constraint.

ordering are already enforced. However, constraining the spacing is very useful for reducing match ambiguity during early stages of incremental line matching when lines are still short, reducing processing and increasing speed. Note, matching of short grid lines (up to 5 grid points) may fail to disambiguate without the spacing constraint.

Spacing denotes the distances between adjacent grid line points. In general, under perspective projection, spacings and their ratios are not preserved, except for the cross-ratio, an invariant number associated with an ordered set of four collinear points. Nevertheless, as neighbor grid features are very close together, locally their projection is approximately affine, so one may assume adjacent spacing *ratios* are invariant and neighbor grid lines are parallel.

Defining the geometric constraints. The tolerance level for the *collinearity constraint* of a corresponding lattice line is borrowed from the estimated lattice in the reference image and modeled using τ_s from equation (5.23), a decision threshold portraying the half distance of a line to its neighbor lines. Let $t = 1, 2, 3, \dots$, be a parameter indexing ordered points of a grid line, $\mathbf{x}_R(t)$ be a point of the line and $\mathbf{x}_T(t)$ be a matching candidate. Hence, $\mathbf{x}_R(1)$ is the first point of the line. Let $l(t)$ be a line constructed by simply connecting $\mathbf{x}_T(t-1)$ to $\mathbf{x}_T(1)$ and $d_s(t)$ be the signed distance of $\mathbf{x}_T(t)$ to $l(t)$ computed via the dot product of $\mathbf{x}_T(t)$ with a unit normal of $l(t)$ in some consistent direction. Then, as new points are incrementally matched, the inclusion of every new point $\mathbf{x}_T(t)$ satisfies the collinearity constraint if

$$|d_s(t) + d_s(t-1)| < \tau_s. \quad (5.24)$$

The expression in equation (5.24) provides satisfying results. Other collinearity constraint expressions were tested, such as the cumulative sum $|\sum_{u=1}^t d_s(u)|$ or, similarly, the cumulative sum of angle displacements, or even simply $|d_s(t)| < \tau_s$. However, these allowed some drifting of the estimated matches into curved lines, in part due to the fact that $l(t)$ is not constant.

The *ordering constraint* simply checks if the vectors $\mathbf{v}_1 = \mathbf{x}_T(t) - \mathbf{x}_T(t-1)$ and $\mathbf{v}_2 = \mathbf{x}_T(t-1) - \mathbf{x}_T(1)$ are pointing in the same direction via a dot product,

$$\mathbf{v}_1 \cdot \mathbf{v}_2 > 0, \quad (5.25)$$

which means the points are arranged in the expected order: $\mathbf{x}_T(1)$, $\mathbf{x}_T(t-1)$, $\mathbf{x}_T(t)$.

The *spacing constraint* is performed by comparing ratios of distances among neighbor points along a line. Let $\mathbf{x}_T(t)$ be a match candidate for $\mathbf{x}_R(t)$ and

$$s_r(t) = \frac{\|\mathbf{x}_R(t) - \mathbf{x}_R(t-1)\|}{\|\mathbf{x}_R(t-1) - \mathbf{x}_R(t-2)\|} \quad (5.26)$$

be the ratio of neighboring segments that have $\mathbf{x}_R(t-1)$ as a common point. Analogously, for the matching line denote $s'_r(t)$ as

$$s'_r(t) = \frac{\|\mathbf{x}_T(t) - \mathbf{x}_T(t-1)\|}{\|\mathbf{x}_T(t-1) - \mathbf{x}_T(t-2)\|}. \quad (5.27)$$

Then, $\mathbf{x}_T(t)$ satisfy the spacing constraint if $s_r(t)$ and $s'_r(t)$ are approximately equal values. A slack of $\delta = 1.1$ pixels chosen empirically is allowed in each segment and, in worst case, the deviations are in opposite directions, designating a range of values for $s'_r(t)$, namely $s_{min} < s'_r(t) < s_{max}$ defined by changing the numerator and denominator of equation (5.26) as follows:

$$s_{min} = \frac{\|\mathbf{x}_R(t) - \mathbf{x}_R(t-1)\| - \delta}{\|\mathbf{x}_R(t-1) - \mathbf{x}_R(t-2)\| + \delta}, \quad (5.28)$$

$$s_{max} = \frac{\|\mathbf{x}_R(t) - \mathbf{x}_R(t-1)\| + \delta}{\|\mathbf{x}_R(t-1) - \mathbf{x}_R(t-2)\| - \delta}. \quad (5.29)$$

Note, when $s_{max} < 0$, it is replaced with the more meaningful value $s_{max} = \infty$, but this is essentially unimportant as the chosen slack threshold is $\delta = 1.1$ pixels, in which case a replacement happens only when the grid resolution is higher than of the image and a grid would not even be easily detectable or maybe detectable but not easy to match (see figure 5.23). In addition to relative spacing ratios, the absolute distances of matching neighbor points is only allowed to change by a factor of 1.5 between adjacent images. This constrain in scale is in accordance with the limitations of normalized cross-correlation matching, which does not support large scale changes. Thus, lattice matches that violate this constrain are likely to be erroneous, as the one shown in figure 5.24.

Handling oclusions. One must deal with oclusions and missing true matches, which indistinctly demand a procedure that estimates when it is necessary to skip one or multiple matches. When all match candidate of a grid point \mathbf{x}_R geometrically disagree with the grid topology, then \mathbf{x}_R is allowed to have no match and the analysis continue with the next grid point, taking into account the absence of \mathbf{x}_R (and its match) when measuring new spacings and collinearity.

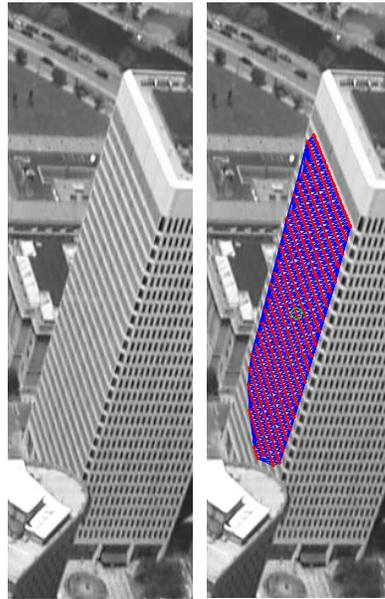


Figure 5.23: A grid seen from an oblique viewing angle. The resolution of the grid points is too high for the image resolution and adjacent corner features are merging into edges. Reasonable grid structure is still detected, however a regular lattice cannot be estimated.

The proposed geometric constraints require at least three points to be computed. It is then only possible to detect missing matches when a line is partially matched. Then, it becomes a problem to have the first two (or more) line points occluded in I_T as there would be no true match among candidates and there would be not enough information to detect it. The line matcher will likely still match those points at some arbitrary location. Shifted solutions can then exist, typically of small size due to probable inconsistency with the geometric constraints once the number of points increase.

In order to find the correct line matches under these aforementioned cases, the line matching is always repeated starting at every line point (as if it was the starting endpoint) and moving towards the other endpoint. When starting from a point that actually has a true match, the line is likely to be matched correctly and be longer than other line matching candidates. The constrained line matching robustly handles occlusion by tolerating missing grid points and skipping unmatched ones, as illustrated in figure 5.22. Figures 5.26c, 5.26g and 5.26h provide matching results under occlusion of the reference and/or target images.

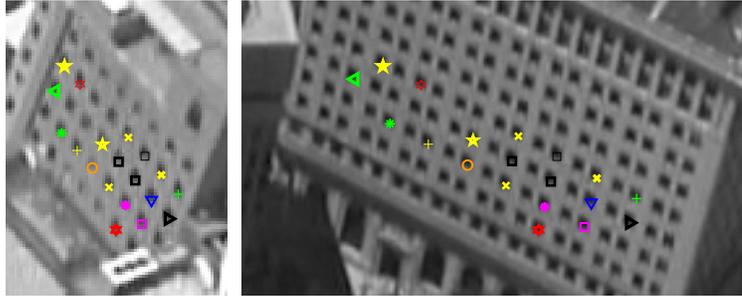


Figure 5.24: Erroneous corresponding lattices that would arise if neighbor point spacing scale was allowed to change drastically. *Left*: a lattice detected in a reference image that is not visible in the target image. *Right*: a target image erroneous lattice match correspondence that agrees with all epipolar and lattice constraints of the proposed model-based matching except that the spacing along horizontal rows of windows changes by a factor of roughly 2.

Handling the multiple grid line matches. For each starting endpoint, there is a line match solution. Among all solutions, assume the line with highest number of matched points is the correct one and discard other lines. If multiple lines have the highest number of matches, both are kept as possible solutions and are disambiguated based on grid line topology validations discussed in section 5.4.4.

5.4.4 Topology-preserving incremental line matching

The matching procedure discussed in this section solves a similar problem as the disambiguation steps in chapter 4, which enforces some spatial and topological global relations into the feature matching process. In chapter 4, features were matched individually, whereas in this section features are disambiguated in groups, *i.e.*, grid lines composed of a set of collinear points. Figure 5.22a has examples of line matches, some of which require disambiguation. Note, the ambiguity of line matches seen in figure 5.22a is smaller than is the ambiguity of matching individual points (*c.f.* figures 4.12 and 5.20).

Grid topology is enforced using estimated grid line neighborhoods and line intersections. The matching process starts by attempting to match a grid line using procedure in section 5.4.3.

Matching the first line. The initial line match must be unambiguous and/or highly reliable. The longer the line, the more reliable its matching line is given all points must satisfy geometric constraints. Therefore, the grid lines with the largest number of matched points are matched first using method in section 5.4.3. The top line may have multiple matches, which are disambiguated if the top line has only one line match that satisfy an additional validation geometric constraint, the *epipolar validation* discussed next. If the disambiguation of the top line fails, the search continues attempting to match the second highest ranked line and so on, until one match is established.

Epipolar validation of matching lines. The epipolar validation used for individual points is different than the one used for lines. The bounds used to enforce epipolar constraint on individual point correspondences, given in section 5.4.2, are relaxed w.r.t. line matching since the estimated fundamental matrix may have isolated spatial inaccuracies that would sporadically rule out true point matches on tight bounds. A line consists of a group of points and true corresponding lines must in average satisfy the epipolar constraint better than individual points. Therefore, when line matches are wrong and shifted, some bias is expected in the mean Sampson distance of their group of points. A putative line match is considered valid if most of its composing points satisfy the epipolar constraint with the tighter bounds used in chapter 4 for fundamental matrix estimation, *i.e.*, a maximum Sampson distance of

$$\tau_f = 0.2. \tag{5.30}$$

Thus, the percentage of points of a valid line match such that their Sampson distance d_i is lower than τ_f is greater than a minimum probability value $p_{min} = 0.6$ and no Sampson distances are higher than 1.5. In probabilistic terms:

$$P(d_i < \tau_f) > p_{min}, \tag{5.31}$$

$$P(d_i > 1.5) = 0. \tag{5.32}$$

Match intersection consistency. An additional spatial property used to match grid lines is *intersection consistency*, which is a very important concept. When two grid lines intersect at a grid point, their matching lines must also intersect at a common point, which is presumably the match

of the intersection. Two intersecting grid lines are consistent if the intersection grid point has, on both lines, a common match location that satisfies all collinearity, ordering and spacing geometric constraints. The lines are inconsistent if the intersection matches in different locations. If one of the two intersection matches is missing and undefined, consistency is undefined.

Matching a second line. After the first line match has been established, a second line in the other main direction is matched in the same way. The two matches are independent of each other. At least one of the matches is wrong if the two lines do not satisfy the intersection consistency, and the algorithm keeps searching for other lines in the same manner until a pair of consistent intersecting lines is found. Since these corresponding lines were matched independently and are consistent, the match is significantly reliable.

Matching more lines. More line matches are established in the same way as the first two (section 5.4.3) except that, in addition to other validation constraints, the intersection consistency is strictly enforced. In fact, the intersection constraint is the main spatial property driving the line matching disambiguation. If intersection consistency is preserved, the topology of the grid is preserved on the target image. New grid line matches are incrementally incorporated to corresponding grid in the target image. The incremental method expanding the matching grid is described next in more detail.

Incremental match expansion. The matching expansion tries to add one line at a time to the matched grid, alternating between lines of the two main grid directions. The method attempts to incorporate larger lines first. The expansion relies heavily on the consistency of intersecting points between matching grid lines. As new line matches are being incorporated, they must conform, in the context of intersections, with all lines matched prior to them. Thus, the larger the matching grid, the more constrained the matching of subsequent lines given previously matched lines, reducing match ambiguity and processing time. The matching using intersection constraint is basically unaffected by match ambiguities since there is essentially one coherent matching line or none because if two satisfy intersection consistency at a point, it is likely that one is abnormal and does not satisfy consistency on the rest of the grid. Abnormal lines are usually the small ones, so they are matched last. Note,

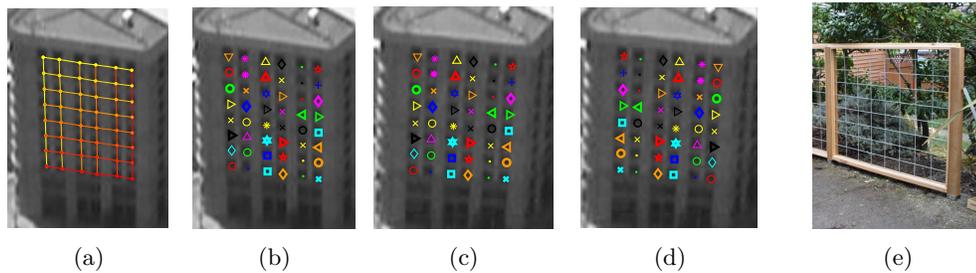


Figure 5.25: The solution of the proposed lattice correspondence problem is not unique under a degenerate configuration where an epipole and a lattice vanishing point coincide. Even applying the global constraints of an estimated lattice (a), a grid of points (b) in a reference image exhibits a feasible correspondence (c) and a reflected one (d) in a target image. A see-through object that has a lattice that can realistically agree with a reflected solution (when seen from two different sides of its plane) is illustrated in (e).

the last lines to be matched already have most of their point matches estimated ahead of time by enforcing consistency. This process repeats until no more lines can be appended to the grid, resulting in matched grids as in figure 5.26, which shows robustness to occlusion and oblique views useful in wide-baseline matching (discussed in detail in section 5.5.1).

Reflected match ambiguity. A lattice has two edge directions and each one has a vanishing point. Epipolar lines and lattice lines are coincident if an epipole coincides with a vanishing point, leading to a degenerate configuration of the proposed planar matching model where two solutions are possible. In the context of urban scenes, one solution is feasible, whereas the other is a reflection of the planar surface of the lattice (see figure 5.25). The reflected solution would require cameras located at opposite sides of the planar surface and also the features to be visible in both cameras. However, surfaces of building facades present no such property due to self-occlusion. In addition, seeing the object from behind would require a very large viewpoint change, which is out of the scope of this thesis. A see-through object that could physically exhibit a reflected solution is a thin fence (see figure 5.25e). In order to detect which solution is feasible, three arbitrary non-collinear points $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ and their matches $\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\}$ are selected and the cross product $(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$ is in the same direction as $(\mathbf{q}_2 - \mathbf{q}_1) \times (\mathbf{q}_3 - \mathbf{q}_1)$ only in the feasible solution.

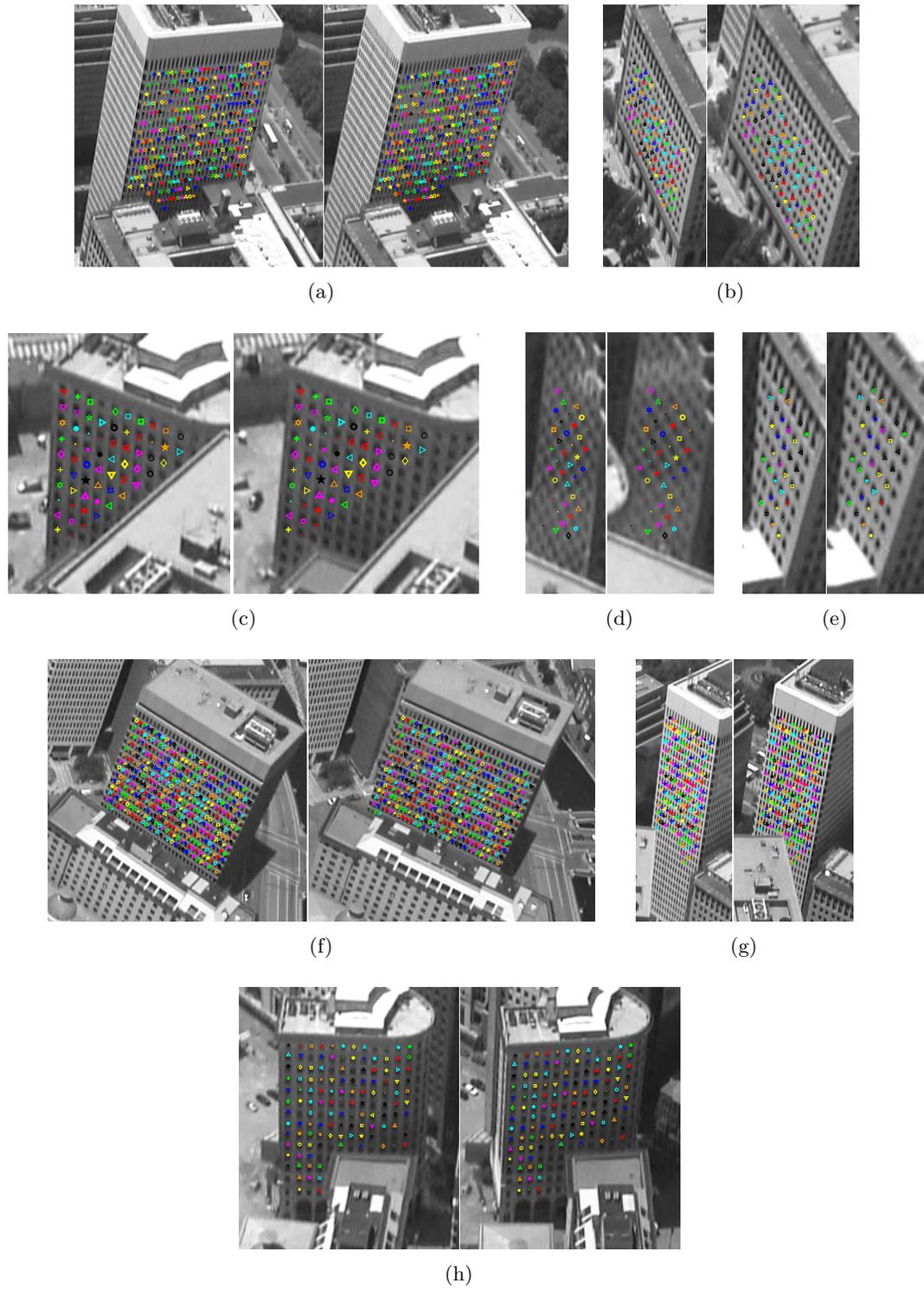


Figure 5.26: Examples of pairwise matches of building facade grid features seen from distinct angles.

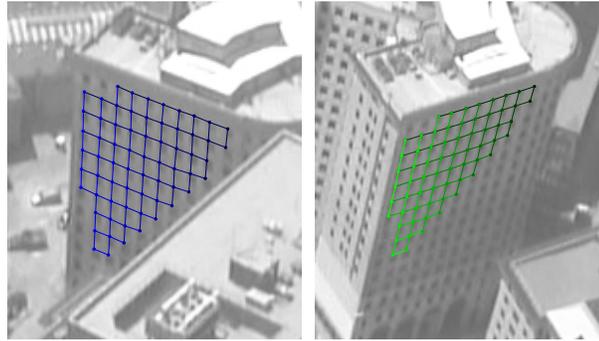


Figure 5.27: A lattice connectivity estimated in an image (right) is transferred to another image (left).

5.5 Wide-baseline matching via lattice tracking

The proposed lattice matching of method of section 5.4 is applicable for adjacent frames with appropriate baseline, which must be relatively short. This section describes how this method can be used in a sequence of adjacent frames to track a lattice over longer spans. The goal is to achieve ultra wide baseline matching of planar lattices to obtain high-quality matches that improve structure from motion.

At this stage, the algorithm assumes cameras are estimated and optimized via bundle adjustment using image matches found via method of chapter 4. A corresponding lattice may be transferred to a third view using three-view geometry (as in section 4.4.11) if cameras were not given. Lattice transfers is an essential step towards wide-baseline matching as it does not require additional correspondences search and does not suffer from the limitation of the lattice matching method described in section 5.1 (failure under alignment of lattice lines and epipolar lines). Thus, the tracking is relatively fast and without theoretical limitations. Assuming known cameras, lattice points are actually transferred using triangulation of matching pixel rays and forward projection of reconstructed point in a third image. The use of camera geometry for the transfers provided better localization accuracy than the use of three-view epipolar geometry. Note, the lattice structure and its connectivity estimated in a reference image can be straightforwardly transferred line by line to matching lattices in other images and only needs be estimated once per lattice track (see figure 5.27). A *track* is a list of corresponding locations of a 3-d point in multiple images. A lattice track is a set of tracks for the grid points of a lattice.

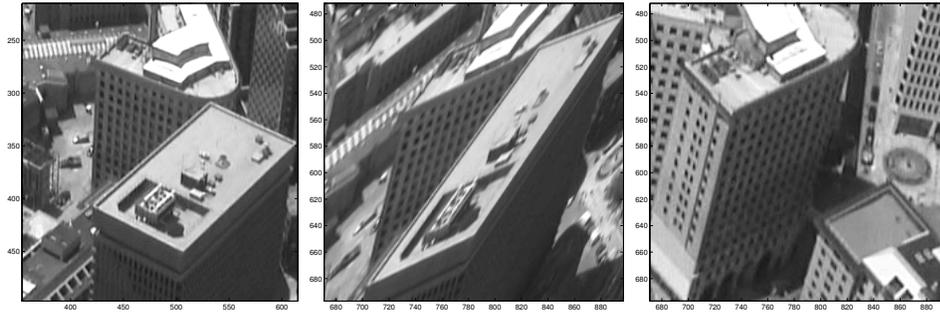


Figure 5.28: Illustration of facade registration. A planar homography estimated from a planar lattice correspondence is used to take an image (*left*) and warp it (*middle*) such that the lattice region is registered with respect to a second image (*right*). The homography is estimated from matched grid points as shown in figure 5.27. Note, the registered grid points statistically share the same coordinate and the unoccluded regions of the facade in the *middle* image is shown as if it was seen from the viewpoint of the camera of the *right* image.

5.5.1 Tracking lattice planes

Matching on wide-baselines typically require tracking points among multiple short-baseline frames. Normalized cross-correlation is still used here for wide-baseline matching since the perspective distortions of large viewpoint changes can be compensated for planar surfaces using planar homographies. The homographies are estimated from the matching grid features and used to warp images and register associated building facades, as in figure 5.28.

Given a sequence of images, a track is initialized with a lattice correspondence in first two adjacent views via lattice matching method of section 5.4. The lattice connectivity estimated for the first image is transferred to the lattice in the second image. To find the correspondence in the next view, the lattice points may be transferred then refined via normalized cross-correlation. However, the transfer is based on geometry alone and not image intensities so it may not be accurate. When dealing with dense repeating features, a transfer of a grid point that has an error of just a few pixels may coincide with another grid point. A local refinement of such incorrect match does not fix it. The transfer is instead used as a guide to reduce grid point correspondence search of the method of section 5.4 from an epipolar line to the vicinity of the transferred point. Thus, the method of chapter 5 is then applied on the adjacent second and third views, which returns a corresponding lattice on the third view and a homography between them. A track has been established between the first, second and third views, however, an error may accumulate in feature locations since there were two correlation matches

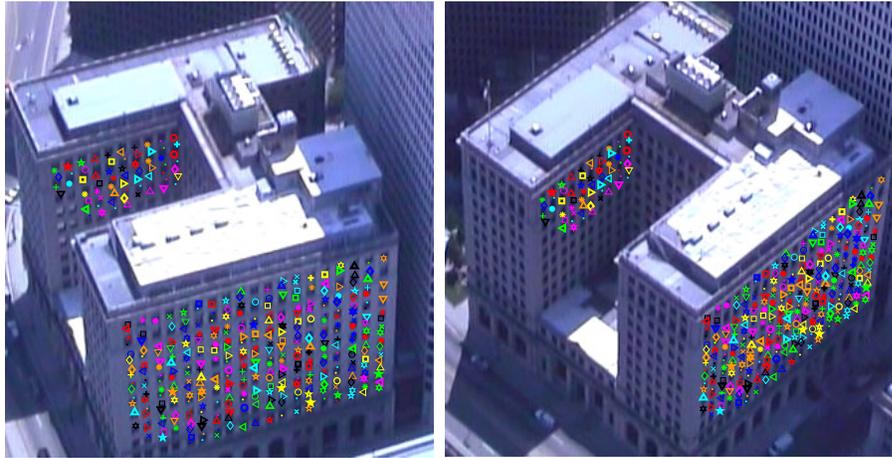


Figure 5.29: Result of the proposed wide-baseline matching of dense repetitive features showing robustness to occlusion and multiple planar surfaces.

between a point in the first image and its corresponding location in the third image. In order to prevent this type of error accumulation, a refinement is performed by first warping the third image to register the lattice region with the first image, as the warping illustrated in figure 5.28. This warping transformation is given by the composite homography from the homographies of the two adjacent image pairs. Second, a new correlation matching is performed directly using the intensities of the first image and warped third image. The search region is simply the small neighborhood around the already coarsely estimated matches, returning refined matches that are not affected by viewpoint distortion because the facades are registered, therefore the matching is very accurate. A refined homography between the first and the third views is also computed. Thus, error does not accumulate. By induction, this tracking process repeats visiting subsequent views. In general, tracking advances starting from estimated matches between the first (reference) and the n -th view and then extending the match to the $(n + 1)$ -th view via adjacent lattice matches between the n -th and $(n + 1)$ -th views followed by a refinement. Results of the tracking process are shown in figures 5.29 and 5.30.

The quality of the matches is validated during tracking using 3-d geometry of reconstructed grid points. When tracks are correct, triangulated 3-d points associated to the grid features are near each other regardless of which views were used for estimating them. If at some point the reconstructed grid is off in 3-d space, the tracking process stops.

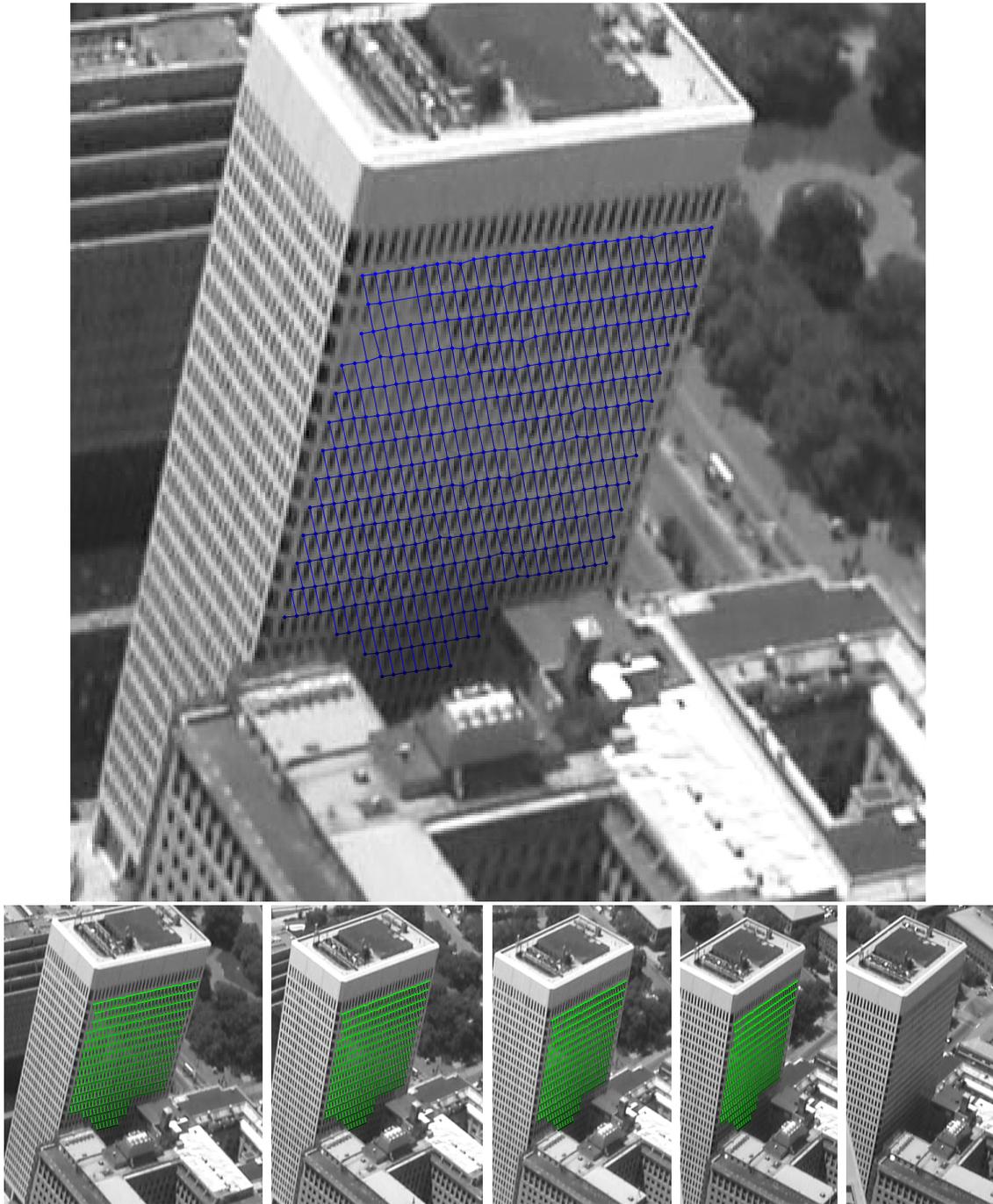


Figure 5.30: Example very wide-baseline tracking of a planar facade. *Top*: a facade lattice detected in a reference image. *Bottom*: the reference image facade lattice tracked on subsequent frames of increasing baseline. Track is lost only when viewing angle is too oblique and grid point corner features are not observable anymore. At such stage, tracking is no longer reliable since corner features have merged with neighbor corners exhibiting a smooth edge feature instead, and the algorithm automatically stops.

The approach described here using multiple view geometry resolves the remaining limitation of the two-view lattice matching method of section 5.4 (see limitations described in section 5.1), and is then robust to aligned epipolar and lattice lines. Note, not every grid point is necessarily tracked on every frame and some grid points to which track was lost can be rediscovered in further frames. This is useful to cope with occasional specular reflections seen on glass building windows, which provide common lattice features, among other factors that may disrupt the observed features.

5.6 Conclusion

The feature matching method presented in this chapter assumes the existence of grids of repeating features with planar geometry in the scene and exploits this global spatial information to resolve a highly ambiguous correspondence problem. The procedure handles the very ambiguous matching of very repetitive features common in urban scenes, such as building windows. Matching is solved via fitting of multiple planar models to subsets of features in an image pair assuming the subsets are piecewise planar. The procedure is fully automatic and assumes no prior knowledge of the scene. The correspondences estimated via such planar models can be used to track planar regions under very wide baselines, which improves the accuracy of structure from motion (see experiments in chapter 8), by tracking regions among successive short-baselines pairs using homographies. The proposed method uses the real-time normalized cross-correlation matching GPU implementation presented in chapter 7 to accelerate processing. The following are the major contributions of the proposed approach:

- the estimation of grid points, the fitting of lattices to grid points and the search for lattice correspondences;
- the use of global spatial constraints to resolve a very ambiguous correspondence problem;
- the detection of multiple grids of similar features in a single image;
- the tracking of lattices that yields very wide-baseline matches; and
- the handling of occlusions.

Experiments in chapter 8 demonstrate that the accuracy of correspondences found with this proposed system achieve accuracy much higher than the one of SIFT matching.

Chapter 6

Real-time Depth Estimation



Figure 6.1: Estimated depths computed in 1.1 seconds. A total of 550,000 rays and 75 million depth candidates along the rays are processed using five images, one reference view and 4 others, three of which are shown on the right column.

This chapter presents a parallel implementation of multiple view stereo for depth estimation inspired by the sequential methods of [124] and [48]. Similarities are discussed throughout the chapter and proposed contributions are summarized in section 6.5. The **entire** multiple view stereo pipeline is GPU accelerated and Gaussian-weighted normalized cross-correlation scores are robustly combined from multiple views based on local peaks of a correlation function along epipolar lines. The insights regarding the peaks will be used to define the depth continuously along each ray in the presence of uncertainties.

Section 6.7 presents a complementary approach to fit a surface to the estimated 3-d points using the implicit volumetric representation of [124]. The problem is formulated and solved through energy minimization via graph cuts. The solution of this volumetric approach is a volume and the surface is its boundary. This surface meshing method is a more efficient extension of the approach from [124] and runs on CPU.

6.1 Algorithm description

The input to the system is a set of calibrated images and a 3-d finite bounding box defining a *volume of interest* within which the object lies. The volume can be estimated at the camera calibration stage as the bounding box containing all reconstructed features used in bundle adjustment (see section 3.4.1). The approach consists of two steps: estimating a depth map from each input view, and fusing the maps into a volumetric model to extract a surface mesh. In the first step, reference camera rays are backprojected into the scene and depth is estimated using robust template matching based on weighted normalized cross-correlation and a few nearby views. In the second step, the results of learned depth, occupancy and visibility are merged into a volumetric voxel grid. A graphical model is constructed such that an associated cost function optimal solution is a watertight surface approximating the true observed surface and filling the gaps where there is no data. The following sections of this chapter describe the first step of the algorithm in more details.

6.2 The ray grid

For a given image ray, the depth defines the location of the associated observed surface point. In following sections, multiple view stereo is used to estimate depth along each ray independently. This section describes the choice of such rays.

Rays in question are 3-d lines back-projected into the scene from a reference camera center passing through given points in the volume of interest. The choice of these points define a pencil of rays of interest, the *ray grid*. These points may be chosen as the center of a pixel (aligned with the image grid) or as the center of a voxel (aligned with the voxel grid). As estimated 3-d points will be later

converted into an implicit volumetric representation, the *voxel aligned* grid (VAG) is chosen, instead of the *pixel aligned* grid. This choice is justified by two reasons:

Reason 1: The size of the problem is easily adjusted by a voxel grid resolution parameter.

Reason 2: Unlike pixel alignment, VAGs guarantee that every voxel is visited once by a ray.

The first reason allows for quick reconstructions if the volume has low resolution. The second motive is important for embedding dense visibility and occupancy information throughout the entire volume, in addition to the surface location (depth), regardless of image resolution.

One disadvantage of voxel aligned rays is that a voxel grid of size $N \times N \times N$ produces N^3 rays with inherent redundancy, as many rays essentially overlap. A solution is discussed in detail in section 6.6, providing a smaller subset of the voxel aligned rays that still visits all voxels of the rectangular volume. The reduction brings the number of rays to $O(N^2)$, suppressing redundancy and satisfying the two reasons above.

6.3 Rectification

The first step prior to depth estimation is to rectify a stereo pair. Rectification is an image transformation process based on epipolar geometry, the intrinsic projective geometry of stereo vision that defines a number of geometric relations among corresponding points in 2-d images (see section 3.1).

Motivation. Normalized cross-correlation is not invariant to scale changes, or rotations, or perspective distortions. In general, a square patch in a reference viewpoint corresponds to a non-square patch on another. The motivation for using rectification for GPU-accelerated multiple view stereo is then two-fold:

- High performance for correlation computations to search over horizontal epipolar lines, since contiguous GPU memory access is very efficient (cached), as opposed to strided access along vertical or tilted epipolar lines (discussed in detail in section 7.1.2);
- Invariance to vertical image scale changes and image rotations as the image rows are aligned after rectification.

After rectification, general perspective distortions between small correlation patches are reduced to non-isotropic scalings and deformations that essentially only affect the horizontal direction (*e.g.* skew transformations). According to Bradley *et al.* [17], the skew is negligible for moderate baselines (up to 45° between viewing rays) and moderately horizontally slanted surfaces (up to 60° between surface normal and epipolar plane). Therefore, skew is an issue mostly on wide-baseline multi-view stereo setups, which is not the case for image sequences with small inter-frame motion, such as the ones used in this thesis. For datasets where horizontal scaling may be present, a search over scale factors may be necessary. There is a trade-off between computational cost and accuracy in choosing the frequency of sampling in scale. Experimental results show that scale factors of $\sqrt{2}$, 1 and $\frac{1}{\sqrt{2}}$ (half an octave steps) present a good balance for matching relatively small baselines, which has little effect on an algorithm performance [17]. In this thesis, only the scale factor 1 is used.

The following sections explain the process of estimating range images from given rectified viewpoints.

6.4 Multiple view stereo matching

The proposed multiple view stereo approach for depth estimation is discussed in this section. Table 6.1 summarizes the most important notation used in the proposed method. Some notation dependencies are considered intuitive and omitted for clarity.

6.4.1 Similarity score along a ray

Let an arbitrary reference image i be chosen along with its M neighboring images $N(i)$, and V be the volume of interest in the scene. These views are expected to have small baselines and observe the same scene from slightly different viewpoints. The images share roughly the same scale, which is important for normalized cross-correlation methods. Let c_i be the camera center of image i and r_i an arbitrary ray from c_i through the volume V . The ray is uniformly sampled at a set $P_r \subset V$ of surface candidate points, each one associated with a depth. By construction, the samples do not have to be at voxel centers. A good sampling rate along the ray can be chosen as half the size of a voxel. The points in P_r are constrained to the volume V and project onto a single pixel in rectified

Notation	Description.
i or j	Index of an image.
$N(i)$	Set of closest images to image i .
M	Number of elements in set $N(i)$.
V	Volume of interest for depth estimation.
c_i	Camera center of image i .
r_i	An arbitrary ray from c_i through V .
P_r	Set of uniformly sampled points (depths) along a ray r .
$e_j(V, r)$	Epipolar line segment in j associated with a ray $r \cap V$.
γ_w	A correlation coefficient from weighted normalized cross-correlation.
$S_j(e_j)$	Discrete correlation scores γ_w along pixels of an epipolar line e_j .
$\mathcal{C}_j^w(e_j)$	Real, continuous and peak enhancing function derived from $S_j(e_j)$.
$Q_j(p)$	Backprojection of scores \mathcal{C}_j^w from image space back into $p \in P_r$.
τ	Minimum correlation score $Q_j(p)$ for p to be a valid depth candidate.
$\mathbf{Q}_r(p)$	Set of all images $j \in N(i)$ such that $Q_j(p) > \tau$ is valid for a ray r .
$ \mathbf{Q}_r(p) $	Number of elements in set $\mathbf{Q}_r(p)$.
$\mathcal{Q}(p)$	Mean score of the valid values $Q_j(p)$ of p .

Table 6.1: Notation for proposed depth estimation method.

image i and onto a segment of an epipolar line in rectified image $j \in N(i)$. The segment is denoted $e_j(V, r)$, or e_j for simplicity.

No information of the scene geometry is known *a priori* and camera models are estimated using proposed method in Chapter 4. In order to estimate depths along rays, a photo-consistency score must be associated to points in P_r . A location in 3-d space is photo-consistent if it displays similar appearances under viewpoint change. In general, surfaces are often photo-consistent, while interior or exterior of objects are not. The photo-consistency similarity score used is Gaussian-weighted normalized cross-correlation, computed for the entire epipolar line segment e_j and defined in equations (3.7) to (3.12). Template patches $t(x, y)$ have size $t_x \times t_y$ and the Gaussian weight function $w(x, y)$ has a diagonal covariance matrix $\Sigma_w = \text{diag}(\sigma_x^2, \sigma_y^2)$. As discussed in section 3.2.2, the use of a Gaussian weight function is to simultaneously reduce matching bias and noise for a given template size. In the proposed stereo reconstruction, the bias manifests itself as surfaces waves illustrated for a scene in figure 6.2.

The correlation is carried out on *image space* centered at each pixel of e_j , and $S_j(e_j) \in [-1, 1]$ denotes these scores. These computations are very efficient on GPUs, but what is ultimately required are the correlation values at points P_r in *voxel space*. A mapping is defined between these spaces

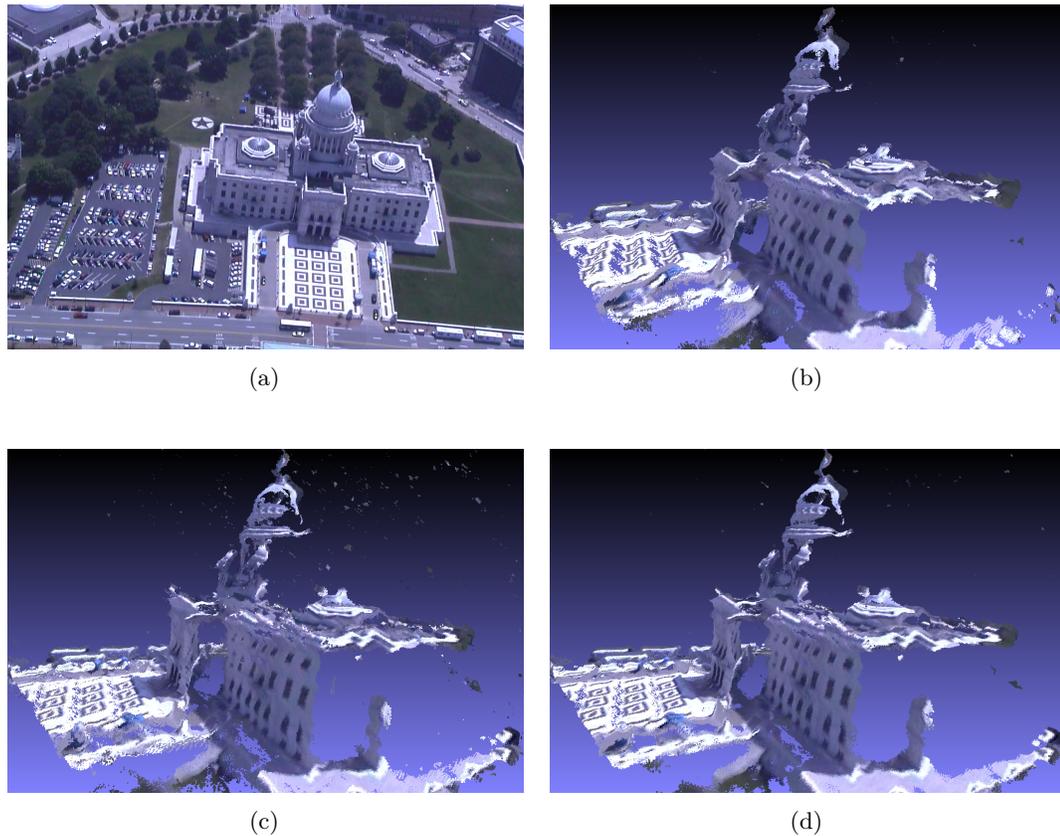


Figure 6.2: Comparison of normalized cross-correlation (NCC) with Gaussian-weighted normalized cross-correlation (WNCC) w.r.t. bias and noise effects on a dense and textured point clouds. The points are extracted from a single reference viewpoint and four nearby views as a result of the 3-d reconstructions from sections 6.4.6 and 6.4.7. The resolution of the points is high enough that it may resemble a continuous surface. (a) One of the five images used from the “capitol” dataset. (b) Reconstruction using NCC (constant weight function) with a template of size 11×11 . A significant amount of ripples is noticeable on the planar surfaces of the building walls and ground plane. (c) Reconstruction using NCC with a template of size 6×6 showing that the biased waves are reduced but many noisy spurious reconstructions appear. (d) Reconstruction using WNCC with a template of size 11×11 displaying the same bias reduction of a smaller template as in figure 6.2c with as little noise as of a larger template as in figure 6.2b.

(section 6.4.3) using a continuous representation of $S_j(e_j)$ based on its peaks (Section 6.4.2). Then, the scores from the multiple views j are merged into a single score (Section 6.4.5). And finally, a depth is estimated from the combined score (Section 6.4.6).

6.4.2 Peak representation

This section is performed as in [124], except that their normalized cross-correlation function was not weighted. Assuming a Lambertian surface, if $p^* \in P_r$ is the true surface point visible at the image associated pixel, it will in general have a high value of S_j . However, these photo-consistency measures are noisy due to unmodeled uncertainties such as occlusions, highlights and perspective distortions. Often, the global maximum of S_j does not correspond to the surface point p^* .

The key observation of [124] is that p^* does show at least a local maximum in S_j . In order to exploit this fact, a new *continuous* function $\mathcal{C}_j^w(e_j)$ is defined such that the local maxima neighboring p^* from multiple views will enhance each other, while meaningless scores between the peaks are ignored. This is accomplished by defining $\mathcal{C}_j^w(e_j)$ as the sum of weighted Gaussian kernels centered at each one of the peaks d_{j_k} of S_j , for all j . The weights are the values $S_j(d_{j_k})$ at the peaks:

$$\mathcal{C}_j^w(x) = \sum_k S_j(d_{j_k}) \cdot G(x - d_{j_k}), \quad (6.1)$$

$$\forall d_{j_k} : S_j'(d_{j_k}) = 0 \text{ and } S_j''(d_{j_k}) < 0.$$

where S_j' and S_j'' are the first and second derivatives of S_j . S_j is defined in discrete image space and the Gaussian $G(x)$ is continuous with variance σ_g^2 . Hence, $\mathcal{C}_j^w(x)$ is defined in continuous image space. The domain of its parameter x is the projection of r_i in image j . The peak representation is sparse w.r.t. storing all correlation scores S_j . Only peaks are required to evaluate the continuous function $\mathcal{C}_j^w(x)$. This sparse continuous representation is important for GPUs due to their relatively smaller memory sizes.

6.4.3 Score mapping from 2-d to 3-d

The scores Q_j for each 3-d point $p \in P_r$ are computed by evaluating C_j^w at x_j in image space, where x_j is the projection of p onto image j . The backprojected score Q_j is defined below:

$$Q_j(p) = C_j^w(x_j), \quad \text{for } j \in N(i) \quad (6.2)$$

6.4.4 Background rays

Background rays are the rays that intersect the surface in a point outside of the volume of interest. The intersection may happen before or after the ray traverses the volume. These rays are not properly modeled by the proposed approach and the scores associated to them are not meaningful. In general, the normalized cross-correlation scores for background rays are expected to be relatively small with peaks that do not often align in space. Under this assumption, background rays are filtered out, as desired, by having invalid scores as described in section 6.4.5.

6.4.5 Score fusion

For robustness, the measures $Q_j(p)$ must be combined into a single and more consistent score. The authors of [124] suggest combining by summing $Q_j(p)$ over all neighboring views and choose the depth along a ray as the global maximum of the sum. This function naturally captures the peaks reinforcement idea described in section 6.4.2, but its magnitude varies with occlusions and other uncertainties depending on how many local peaks are missing. This suggested fusion method also inadvertently estimate depth for all background rays.

In order to increase robustness, a more consistent combining method, similar to [48], to merge scores based on their confidence is used. Figure 6.3 compares the proposed fusion method with the one from [124]. In [48], the merged scores are the raw non-weighted correlations S_j and not the robust peak reinforcing Q_j scores derived from a Gaussian sum used here. First, the proposed fusion $Q(p)$ of the scores for a given point (or depth) p is considered *valid* if at least two neighboring views present $Q_j(p)$ higher than a threshold $\tau > 0$. The set of valid views for a ray r_i in question is denoted

$\mathbf{Q}_r(p)$:

$$\mathbf{Q}_r(p) = \{j \in N(i) \mid Q_j(p) > \tau\}. \quad (6.3)$$

Thus, the fused score $\mathcal{Q}(x)$ is valid if the cardinality of the $\mathbf{Q}_r(p)$ set

$$|\mathbf{Q}_r(p)| \geq 2, \quad (6.4)$$

where $|\mathbf{Q}_r(p)|$ denotes the number of elements in the set $\mathbf{Q}_r(p)$. Then the number of neighboring views defined in section 6.4.1 is limited to $M \geq 2$. Finally, a valid $\mathcal{Q}(p)$ is defined as a normalized score as follows:

$$\mathcal{Q}(p) = \frac{\sum_{j \in \mathbf{Q}_r} Q_j(p)}{|\mathbf{Q}_r(p)|}. \quad (6.5)$$

The valid values of $\mathcal{Q}(p)$ share the same range as a normalized cross-correlation function, $|\mathcal{Q}(p)| \leq 1$. Furthermore, $\mathcal{Q}(p) \geq \tau$. This function has more consistent values in the presence of highlights, structure out of an image field of view, occlusions and background rays. These problems are common in aerial views of tall buildings and other cluttered scenes. The pruning performed by equation (6.4) removes many erroneous depth estimates, especially the ones along background rays, while essentially preserving the correct ones. Figure 6.3 illustrates some interesting cases where the point being matched is occluded, or leaves the image borders, or is on the background or is an edge.

6.4.6 Depth estimation

The depth along the ray r_i is set at the global peak of a valid $\mathcal{Q}(p)$. A valid peak is a point p_k such that $\mathcal{Q}(p_k)$ is larger than of its neighbors and both itself and its neighbors have valid $\mathcal{Q}(\cdot)$ scores according to equation (6.4). The depth of the surface point is estimated as

$$q = \underset{\text{valid peak } p}{\arg \max} \mathcal{Q}(p). \quad (6.6)$$

If the global maximum of \mathcal{Q} is at a extreme, given by the volume of interest boundary, it is ignored. If no peak is valid, no geometry is reconstructed for the ray in question. The outcome of this is a dense point cloud for each reference image that handles background rays (figure 6.4).

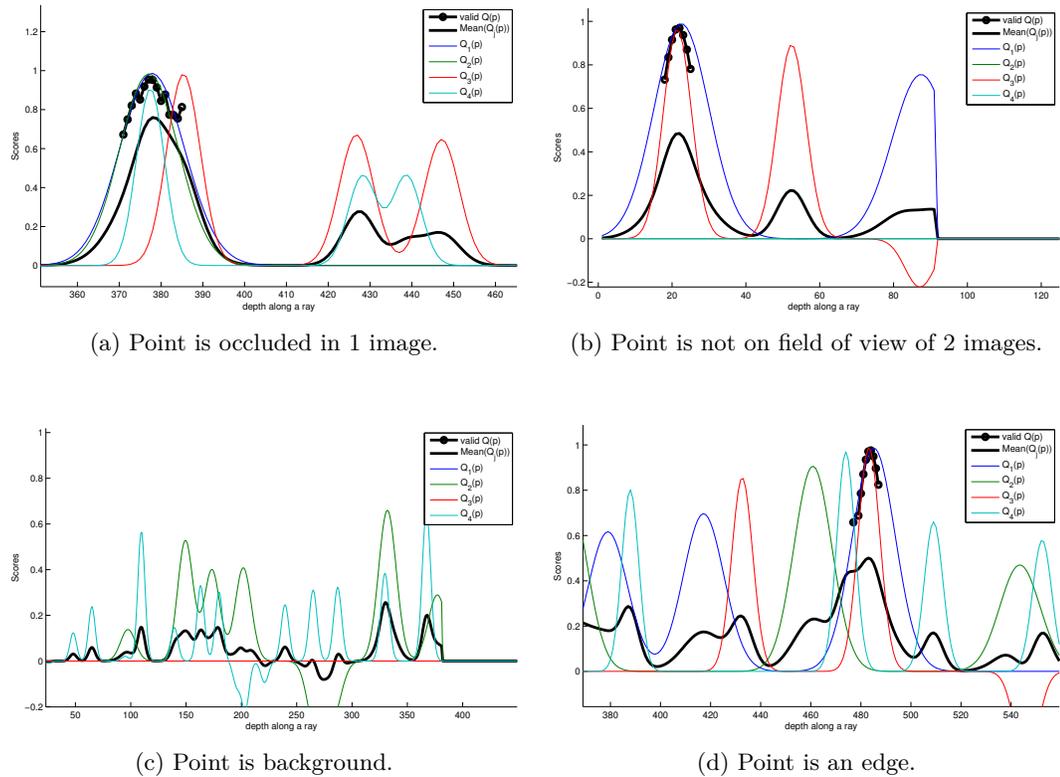


Figure 6.3: Results of fusion methods for combining similarity scores along rays from multiple viewpoints. Thin lines show the similarity scores $Q_j(p)$ (sums of Gaussians centered at correlation peaks) from nearby views of a reference camera. Thicker lines with circle markers show the proposed merged score based on consensus of multiple views, which is only plotted when it is valid according to equation (6.4). The ticker line with no marker is the merged score from the averaging method, which magnitude decreases away from ideal 1.0 at the true match for the following cases. (a) Shows partial occlusion of a point in one out of four views, which shifted one peak. (b) Displays the absence of a matching point inside the borders of two nearby images rendering no scores for two views. (c) Presents a case of a background ray where the point is not modeled by any of the nearby images and the proposed method is nowhere valid, as desired. (d) Shows the case where a point is an image edge roughly aligned with its epipolar line, so the locations of true matching peaks are inaccurate in two of the four views, but the proposed score was not affected.

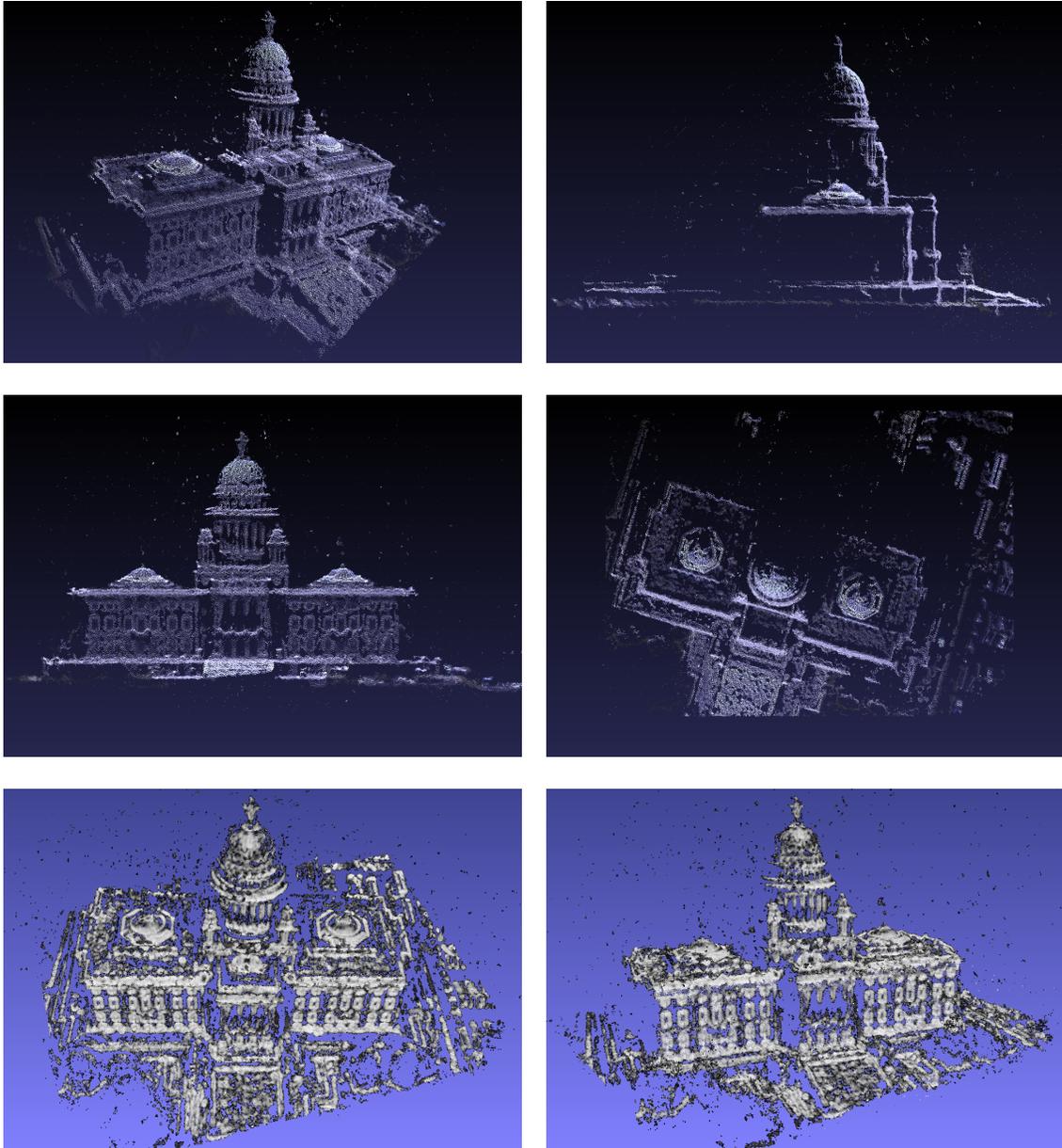


Figure 6.4: Estimated depths from a given reference viewpoint of the “capitol” dataset displayed as a point cloud from different angles. Only points with scores higher than $\tau = 0.6$ are shown. The points on the top and middle rows are textured with intensities from the reference image. In the bottom row, point intensities represent the correlation score as a confidence measure. Scores within the range $\tau = 0.6$ to $\tau = 1.0$ are linearly mapped to grayscale intensities from black to white. Note, spurious points have the lowest scores.

Parameter	Default value	Description
M	4 views	Number of neighboring views used for multiple view stereo.
t_x	11 pixels	Number of columns of the correlation template patch.
t_y	11 pixels	Number of rows of the correlation template patch.
σ_x	$t_x/6$ pixels	First parameter of Σ_w .
σ_y	$t_y/6$ pixels	Second parameter of Σ_w .
Σ_w	$\begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}$	Covariance of Gaussian weight function for weighted correlation.
σ_g	3 pixels	Standard deviation of Gaussian kernels in \mathcal{C}_j^w
τ	0.6	Minimum value of $Q_j(p)$ for p to be a valid depth candidate.

Table 6.2: List of depth estimation parameters used by proposed method.

6.4.7 Peak refinement

The approach described previously depends on image peak detection in sections 6.4.2 and 6.4.6. A local peak is detected in a discrete space 1-d function $f(\cdot)$ as a value $f(d_k)$ that is larger than its two neighboring values, $f(d_k - 1)$ and $f(d_k + 1)$.

Due to discretizations, an exact peak location may not have been sampled. In order to approximate it better, a quadratic function (concave down) is fit to the peak locations $d_k - 1$, d_k and $d_k + 1$. A refined peak is given by the parabolic interpolation:

$$d_{peak} = d_k - \frac{b}{2a}, \quad (6.7)$$

$$f(d_{peak}) = -\frac{(b^2 - 4ac)}{4a}, \quad (6.8)$$

where

$$a = \frac{1}{2}f(d_k + 1) - f(d_k) + \frac{1}{2}f(d_k - 1), \quad (6.9)$$

$$b = \frac{1}{2}f(d_k + 1) - \frac{1}{2}f(d_k - 1), \quad (6.10)$$

$$c = f(d_k). \quad (6.11)$$

This refinement is applied to peaks defining Gaussian centers in section 6.4.2 and also to the highest peak of each \mathcal{Q} (section 6.4.6) in order to increase accuracy of estimated depths.

6.4.8 Parameters

There are a number of free parameters in the method description. The default values are presented in table 6.2 and are chosen empirically. The entries of Σ_w in table 6.2 are chosen such that half of the sides of the template patch $t(x, y)$ match $3\sigma_x$ and $3\sigma_y$.

6.5 Contributions

The proposed multiple view stereo method is inspired by [124] and the proposed fusion of similarity scores is motivated by ideas from [48]. While the high level ideas are similar, many details are new, *e.g.*, the proposed method

- a) explicitly handles background rays;
- b) combines multiple view scores using a more consistent method similar to [48];
- c) uses Gaussian-weighted correlation operators, instead of uniform weights;
- d) proposes a real-time GPU implementation of the pipeline, meaning the GPU receives images, cameras and parameters as inputs, and returns estimated depths.

These details are all important. If background rays are not explicitly handled, many incorrect spurious points may be reconstructed. The proposed way of combining multiple view scores is more robust to occlusions. Gaussian weighted normalized cross-correlation presents more accurate reconstruction results reducing inherent wave patterns in estimated surfaces. The proposed framework is a depth map estimation from a reference image. Each image in a dataset can be used as a reference independently of other reference views and the computations for each ray are also independent from each other. Therefore, this method is online and suitable for parallelization on a GPU. Without a GPU implementation the method would take hours to run, instead of seconds. The GPU implementation is presented in section 7.5 of chapter 7.

6.6 Volume far sides

Section 6.2 presented the construction of the set of rays where depths are computed for a given volume of interest. The rays go through voxel centers (*voxel aligned*), as opposed to pixel centers. The construction advantages are listed in section 6.2, but a redundancy drawback arises from ray overlaps. This section proposes a solution to suppress redundancy assuming a *convex* rectangular volume of interest, V , divided into a *fixed* grid of cubic voxels. This is especially useful when converting the representation from a 3-d point cloud into a volumetric representation.

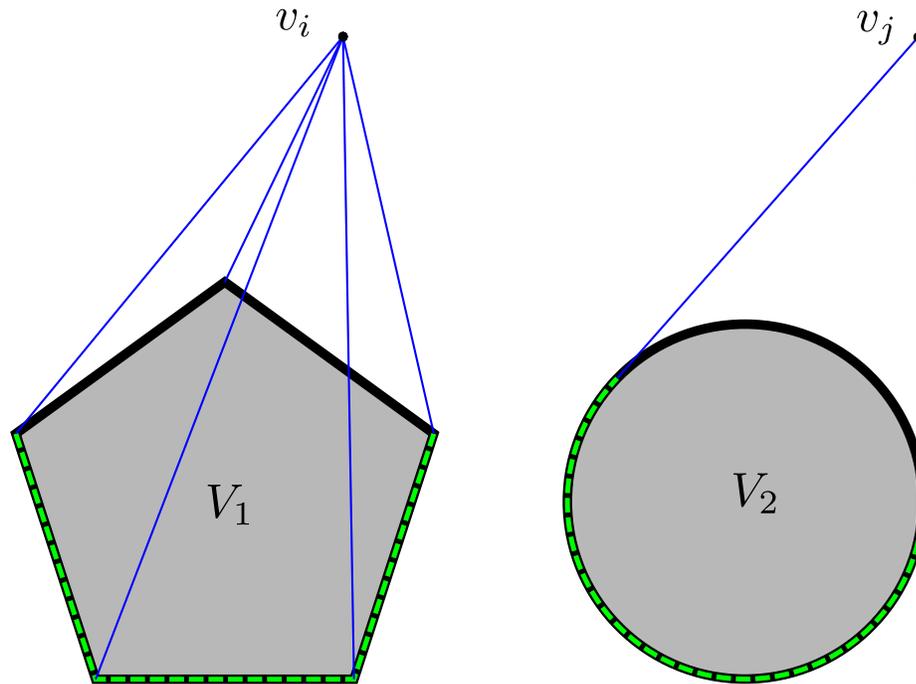


Figure 6.5: Two-dimensional illustration of the *far side* of two convex shapes from a given viewpoint. The shapes are represented by solid black lines and the far sides by dashed green lines. *Left image*: the far side of a pentagon V_1 from viewpoint v_i includes 3 sides of the polygon. *Right image*: the far side of a circle V_2 from viewpoint v_j is an arc. The far sides are always occluded by the shape from the corresponding viewpoints. The lines of sight, from the viewpoint through a point p of a far side F , always intersect the shape somewhere else first, except if p is in the border of F .

It is possible to learn dense visibility and occupancy information on all voxels of a convex volume¹ V by shooting rays only at the voxels on the *far sides* [39] of the volume w.r.t. each viewpoint.

Definition A far side of a convex volume V with respect to a viewpoint v_i (the camera center of reference image i) is a subset F of the boundary of V such that, for all points $p \in F$, V and v_i lie on the same side of the plane defined by the tangent space at p .

Intuitively, when $v_i \notin V$, the far side of V are its self-occluded boundaries from v_i . Figure 6.5 illustrates far side boundaries.

Remark As a consequence of the definition, a far side patch is always occluded by V from the lines of sight from v_i .

¹A convex solid is a solid whose interior is a convex set. Every line segment between two points of the convex set remains inside or on the boundary of the solid.

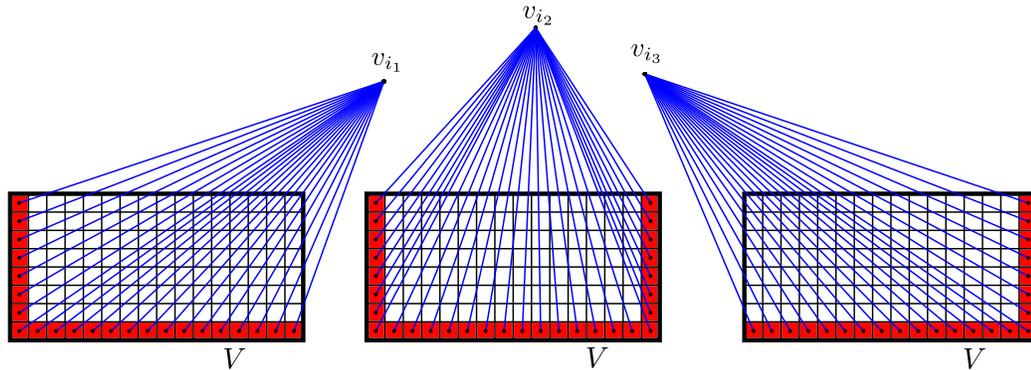


Figure 6.6: Two-dimensional representation of the *far side* of a rectangular grid V and how it varies if the viewpoint changes. The grid elements that belong to the far side are shown in red. Rays are drawn from the viewpoint through the center of far side elements for each one of three different viewpoints, showing that all elements are traversed by these rays.

Proof. Let p be a visible far side point from the viewpoint v_i . Hence, v_i and the interior of V are separated by the tangent plane at p , which is a contradiction of the definition. \square

It is shown below that analyzing the rays through the far side provides the information needed to estimate the visible observed surface from v_i , *i.e.* a depth map seen from v_i .

Remark The lines of sight L from v_i through the far side set $F_i \in V$ traverse the entire convex volume V .

Proof. Assuming the statement is false, there would be a nonempty subset $A \neq \emptyset$ of V , such that A is not intersected by any $l \in L$. Drawing an arbitrary ray $r \notin L$ from v_i that intersects A and leaves the volume at a point p , the tangent plane at p does not intersect the volume interior due to convexity. Thus, the volume, v_i and the segment from v_i to p are on the same side of the tangent plane, so p is at the far side F_i by definition, which is a contradiction. \square

Remark The far side of V is its entire boundary if v_i is inside V .

Proof. The tangent plane of any boundary point p of V separates the space into two parts and V is entirely contained in one part since V is convex. As V contains v_i , they are both on the same side of the plane. Hence p belongs to the far side. \square

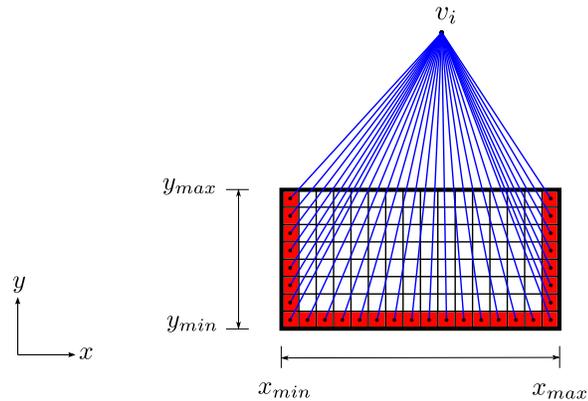


Figure 6.7: Two-dimensional representation of the *far side* of an axis-aligned rectangular grid showing the dependency of the location of the viewpoint relative to the bounding box limits in each dimension. For a dimension where the viewpoint coordinate is interior to the box limits, two sides are included, otherwise only the opposite side from the viewpoint is included.

6.6.1 Far side of a voxel grid

If a bounded convex volume is discretized into voxels, the finite number of lines of sight from a given viewpoint v_i through the voxels on the far side F_i also intersect every other voxel. As they intersect the voxels on the far side, where the density of rays is lower, they must also intersect the ones in the interior and other sides, where the rays are more concentrated. Hence, if one learns depth for all rays going through voxels $v \in F_i$, all voxels are visited during the process as in figure 6.6. Thus, the estimated depth map is as dense as the voxel grid with occasional holes on regions of low confidence.

6.6.2 Far side of a rectangular volume

In this thesis, the volume of interest is defined as a rectangular bounding box with sides aligned with the coordinate axes (figure 6.7). The volume is then bounded and convex, as required to define a far side, which computation is as follows: let the volume be defined as

$$x_{min} \leq x \leq x_{max}, \quad (6.12)$$

$$y_{min} \leq y \leq y_{max}, \quad (6.13)$$

$$z_{min} \leq z \leq z_{max}, \quad (6.14)$$

and the camera projection matrices be given by

$$P_i = K_i[R_i T_i], \quad (6.15)$$

where the viewpoints are the camera centers

$$\mathbf{v}_i = -R_i^t T_i. \quad (6.16)$$

Then the faces of the box that belong to the far side F_i are defined as follows. First the x dimension is analyzed independently:

$$\text{if } -\infty < v_{i_x} \leq x_{min}, \text{ then the face at } x = x_{max} \in F_i, \quad (6.17)$$

$$\text{if } x_{max} \leq v_{i_x} < +\infty, \text{ then the face at } x = x_{min} \in F_i, \quad (6.18)$$

$$\text{if } x_{min} < v_{i_x} < x_{max}, \text{ then both faces at } x = x_{min} \in F_i \text{ and } x = x_{max} \in F_i, \quad (6.19)$$

then analogously for y ,

$$\text{if } -\infty < v_{i_y} \leq y_{min}, \text{ then the face at } y = y_{max} \in F_i, \quad (6.20)$$

$$\text{if } y_{max} \leq v_{i_y} < +\infty, \text{ then the face at } y = y_{min} \in F_i, \quad (6.21)$$

$$\text{if } y_{min} < v_{i_y} < y_{max}, \text{ then both faces at } y = y_{min} \in F_i \text{ and } y = y_{max} \in F_i, \quad (6.22)$$

and z ,

$$\text{if } -\infty < v_{i_z} \leq z_{min}, \text{ then the face at } z = z_{max} \in F_i, \quad (6.23)$$

$$\text{if } z_{max} \leq v_{i_z} < +\infty, \text{ then the face at } z = z_{min} \in F_i, \quad (6.24)$$

$$\text{if } z_{min} < v_{i_z} < z_{max}, \text{ then both faces at } z = z_{min} \in F_i \text{ and } z = z_{max} \in F_i. \quad (6.25)$$

Note, for a cubic volume with N^3 voxels, the number of rays through far side voxels is $O(N^2)$, since the far side include only boundary elements. Computation complexity can be reduced even further by visibility reasoning on the far side (Section 6.6.3).

6.6.3 Far side visibility

Not every far side face needs to be processed to extract all the information from a reference image. Faces that are behind the camera, or not in the camera field of view, *i.e.* not within the image

borders, do not require processing since they do not project onto valid pixel observations on the image plane.

A way to check if a face is behind the camera is to check the sign of the z -coordinate of its N_v vertices in camera coordinates. If \mathbf{V}_j are the vertices in world coordinates, then check if

$$\text{sign}(z_j) \leq 0 \quad \text{for } j \in \{1, \dots, N_v\}, \quad (6.26)$$

where

$$z_j = (R_i \mathbf{V}_j + T_i) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \quad (6.27)$$

Checking for the visibility of a far side face is also simple and efficient, yet not as simple as in equation (6.26). Only the vertices and the camera projection matrix are needed. Breaking the planar rectangular face into two triangles leads to simpler implementation. A face is considered visible if any triangular part is visible. A triangular face is composed of vertices, edges and a planar patch. Emphasizing the camera may be *inside* the volume, there are 3 different ways a triangular face can be visible, as illustrated in figure 6.8:

1. **Vertex visibility:** at least one vertex projects onto the image.
2. **Edge visibility:** no vertices project onto the image, but at least one edge partially does.
3. **Interior visibility:** no vertices nor edges project onto the image, but the image borders lie inside the projected face.

Any portion of a triangle that lies behind the camera must be removed. Cropping is necessary to prevent non meaningful projection of these vertices. For this new possibly cropped triangle, project the vertices $\tilde{\mathbf{V}}_j$ onto the image as

$$\tilde{\mathbf{v}}_j = K_i (R_i \tilde{\mathbf{V}}_j + T_i) \quad (6.28)$$

and check if at least one of the 3 conditions above holds true.

In summary, if at least one of the conditions above holds true for at least one of the triangular parts of a far side face polygon, then the face is visible. Using visibility helps reduce unnecessary processing when the camera is close to or inside the volume, or if the volume is not entirely in the camera field of view.

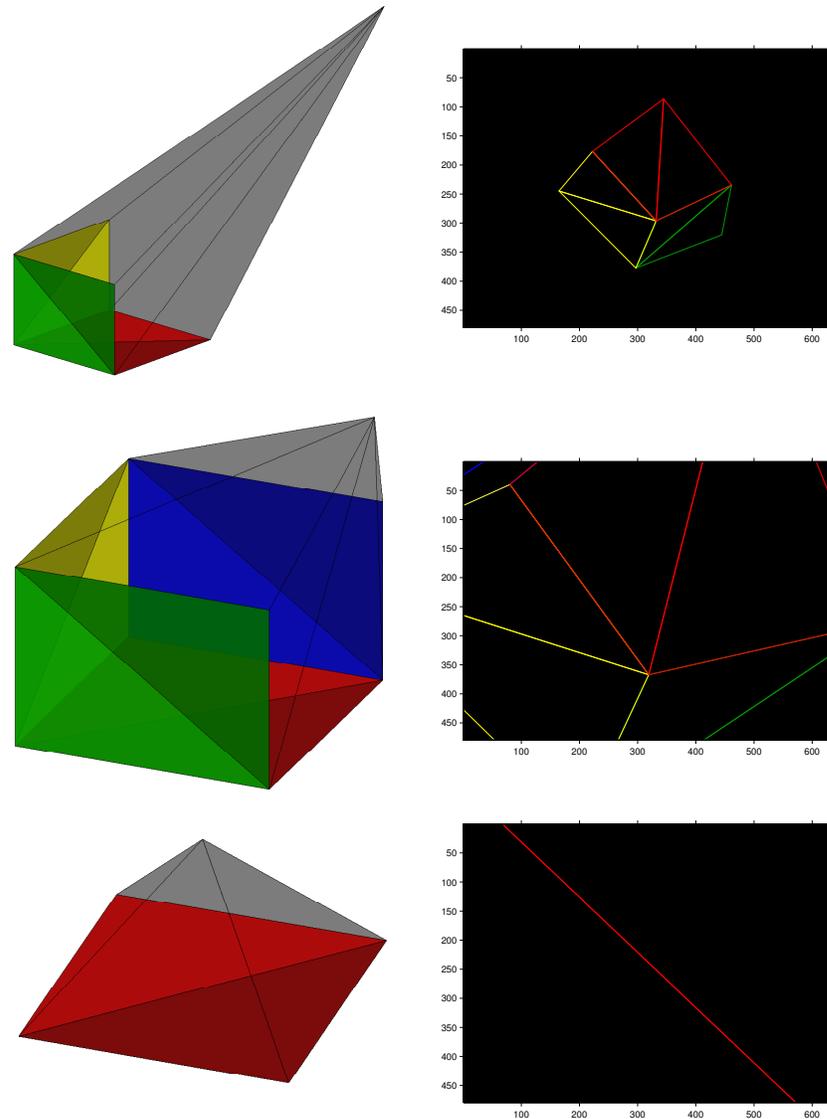


Figure 6.8: Three-dimensional representation of the *far side* of a rectangular bounding box illustrating how it changes as the viewpoint moves towards the volume. The bounding box has six faces, but only the far side faces are displayed for clarity. Faces are displayed as two coplanar triangles. *Left*: the volume far side faces from three different viewpoints and associated viewing cones. Viewing cones are plotted as transparent gray objects from the viewpoints towards the volume and intersect the far side borders. *Right*: rendered images of the far sides from the left column projected at a camera located at the associate viewpoints. At the top and center rows, the viewing cones contain the volume, meaning the rays through the far side traverse the entire volume. At the bottom row, only one far side face is considered important since others are out of the cameras field of view and may be ignored, as described in section 6.6.3.

6.7 Surface reconstruction

This section presents a method to reconstruct surfaces taking as input the point clouds obtained from multiple view stereo method of this chapter, which was entirely implemented on a GPU (see section 7.5). However, the algorithm of this section is implemented on a single-core CPU.

In order to get a complete geometric representation of a scene, an algorithm is required to fit a surface to the points, however this is an ill-posed problem with multiple solutions. State-of-the-art methods impose global constraints in order to specify a solution according to a criterion. In general, such methods define a binary cost function with an unary term and a smoothness cost that favors piecewise smooth objects. In this thesis, we employ an extension of the method of [124] where the smoothness is expressed in terms of fitting a surface that locally minimizes its area subject to a constraint (minimal surface). The chosen constraint is that the surface must be encouraged to be near the estimated point cloud. This type of meshing favors fitting planar surfaces to regions where there are no points, since area must be minimized, otherwise the surface takes other shapes supported by 3-d points and lies near them. The solution is defined through energy minimization via graph cuts, which is introduced in section 6.7.1 prior to the graph construction in section 6.7.2.

6.7.1 Energy minimization using graph cuts

Let a graph be a set of nodes \mathcal{V} and set of edges \mathcal{E} connecting the nodes where each connection has a cost value or weight. In the context of vision methods, nodes are typically representing a random label for pixels or voxels, and the edge costs model discontinuities between the connected nodes and are usually derived from neighbor-nodes interactions. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a weighted graph with nonnegative weights with two special terminal vertices called the source s and the sink t . Terminals here represent labels. There are two types of edges, the ones between nodes (called n-links) whose costs penalize neighbor label discontinuity, and, the ones between nodes and terminals (called t-links), which have costs representing the penalty for not assigning the corresponding label to the node. A *cut* is a partition of the nodes into two disjoint sets that separates the source from the sink (see figure 6.9). The cost of a cut is cost of edges between vertices in different partitioned sets shown as thin edges in figure 6.9, denoted *boundary edges*.

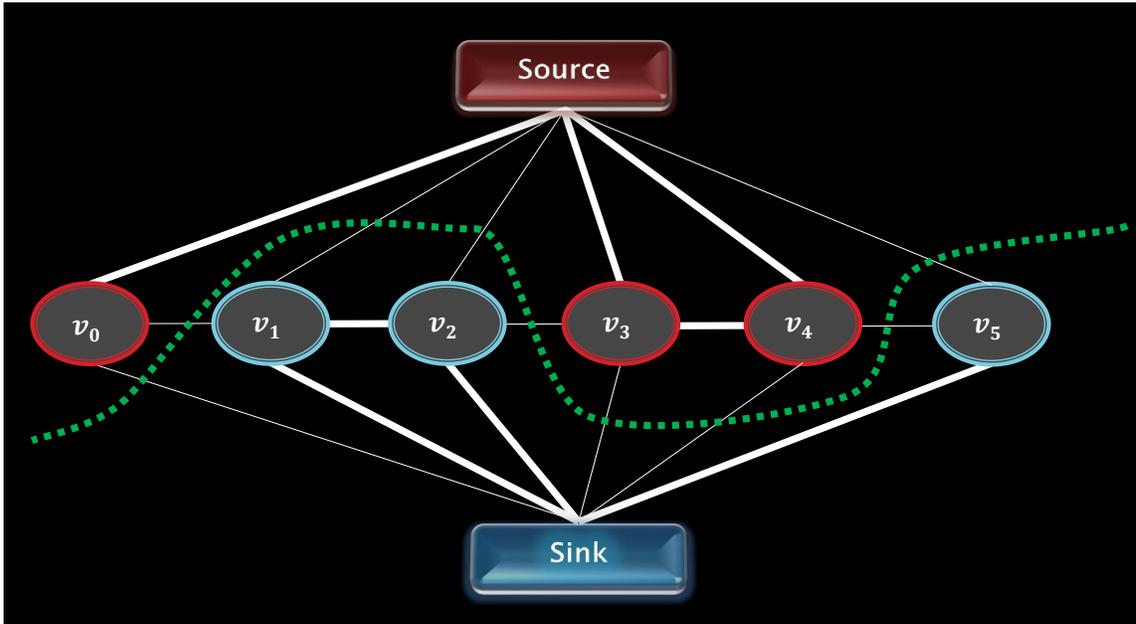


Figure 6.9: Illustration of a cut in a graph with six vertices and two special terminals called *source* and *sink*. The cut is abstractly shown as a dashed green line and partitions the vertices in two sets that separate the terminals. The edges it severs are associated to the cost of the partition. The vertices are colored according to the terminal they remain connected to. The thin edges, denoted boundary edges, are the ones that were severed by the cut and link vertices that are in different sets of the partition. There is a clear binary labeling associated to a cut's partition, which segments the vertices in two sets, where each set is associated to a label (corresponding connected terminal).

The minimum cut problem in combinatorial optimization finds the cut with the smallest cost. Ford and Fulkerson [40] have shown that minimum cut problems are equivalent to maximum flow problems, which are loosely associated to finding the highest flow of water from the source terminal to the sink terminal given that edge costs are pipe capacities. These dual problems can be solved efficiently in low-order polynomial time [3, 49], however in practice, the running time is close to linear for graphs with short paths between the source and the sink terminals [64], as the volumetric ones typically constructed for stereo reconstruction and is used here. Intuitively, the maximum flow is rather constrained by the pipes of smallest capacities as the minimum cut is analogously determined by the smallest costs, namely their sum. In fact, the numeric value of the maximum flow and the cost of the minimum cut are the same.

The partition defined by a cut is equivalent to a binary labeling and therefore graph cuts can be

used to solve such problems, *e.g.* certain types of Markov Random Fields with binary variables. A specialized graph must be constructed such that minimizing the graph also minimizes an associated energy function. In this thesis, an energy or cost is a vector-valued binary variable function with up to pairwise terms with nonnegative coefficients and the cost of neighbor vertices being of the same label is zero. These functions can be minimized using graph cuts. For more general conditions on which energy functions can be solved via graph cuts see [63].

Figure 6.9 illustrates an example of a cut with a 1-d chain graph and the boundary edges that are n-links indicate the *border* between of the segmentation given by the binary labeling. For a 2-d graph, as one constructed to model pixel neighborhoods in 2-d images, the abstract cut would be a surface and the border would be contours. In a volumetric 3-d graph to model voxels, the abstract cut would be a 4-d manifold and the border would be blob surfaces indicating an actual surface location estimate if the graph is constructed to model voxel occupancy.

6.7.2 Graph construction

The surface reconstruction problem is defined in 3-d via a voxel grid. A graph is constructed such that each voxel is associated to a vertex and the immediate voxels are neighbors in the graph as in [124]. Graph cuts minimizes the cost function to find the optimal solution. The binary problem has labels *inside* and *outside* meaning whether voxels are inside or outside an object. The estimated surface is the border between labels, *i.e.* the surface of the observed object.

A graph is constructed as in [124] in order to model the following energy function:

$$E[S] = \iint_S \rho(\mathbf{x}) dA - \lambda \iiint_V dV \quad (6.29)$$

that finds minimal surfaces subject to photo-consistent locations. Intuitively, the double integral term computes the area of a surface S weighted by a cost $\rho(\mathbf{x})$ where \mathbf{x} are points of S . The triple integral computes the volume of the interior of the surface.

The cost term ρ is chosen to be low only at photo-consistent locations to encourage the optimal surface to be near true surfaces and it is defined as

$$\rho(\mathbf{x}) = \exp\left(-\mu \sum_{\text{images}} \text{vote}_i(\mathbf{x})\right), \quad (6.30)$$

where rate-of-decay parameter is chosen to be $\mu = 15$ and the votes are given by

$$\text{vote}_i(\mathbf{x}) = \begin{cases} \mathcal{Q}(q) & \text{if } \mathbf{x} \text{ has an estimated 3-d point via reference image } i, \\ 0 & \text{otherwise.} \end{cases} \quad (6.31)$$

Note, $\mathcal{Q}(q)$ is defined via equation (6.6) as the peak score along a ray. Thus, the images vote for a voxel to be photo-consistent and therefore be a surface location.

In practice, the voxels that are adjacent to locations where there are 3-d points are given low pairwise cost to encourage a change of label and, therefore, to favor surface near points. The higher the score of a point, the smaller the cost. The double integral is modeled as the pairwise terms of the graph by giving edges the weight of $\rho(\mathbf{x})$ multiplied by the area of a voxel's face. Thus, the cost of a surface includes the double integral in equation (6.29) computed in a discrete way as the areas of voxels faces where there is a label change.

The unary cost encourages the volume inside the surface to grow in order to avoid a degenerate trivial solution where there is no surface, in which case the area is zero, if there was no unary cost. Graph cuts always finds a global optimum and would return a trivial solution without the unary term. This term induces an inflating force and is commonly referred to as ‘‘ballooning’’ cost. This is performed by giving no cost to voxels labeled *inside* and a cost proportional to a voxel volume to the ones labeled *outside*. The unary cost would be λh^3 if a voxel has size $h \times h \times h$. This discourages surfaces of zero volume (and zero area) because these surfaces pay a high ballooning cost. For any surface S , this unary cost is constructed such that it reflects the value of the triple integral in equation (6.29) computed in a discrete fashion.

Given a surface, voxel labels *inside* and *outside* are interchangeable generating the same surface. Thus, the global minimum of the problem would be found in at least two configurations where one is the inverse of the other by interchanging *inside* and *outside*. In order to drive the optimization to a determined solution, boundaries of the voxel volume are assumed to be of label *outside*.

6.7.3 Contributions

The previous graph construction summarizes the method proposed by Vogiatzis *et al.* [124]. The method is an minimization in 3-d and is very time consuming due to the high number of variables

defining a 3-d uniform voxel grid. It also has a large memory footprint. The method was modified to accelerate the solution, reduce memory usage and remove some limitations. Note, [124] only demonstrate their method on indoor scenes with images taken on controlled environment with a single object of simple geometry with nearly constant background.

The proposed contributions are summarized as follows. First, for a given reference image, the similarity score of [124] was a function of observations from all neighbor images regardless of occlusion of the modeled voxels in those images. This is not suitable for real-world scenes where objects often occlude each other or the unmodeled background (objects out of the volume of interest) is cluttered. As mentioned before, their similarity is replaced by the proposed one of equation (6.6), which takes care of occlusions by using only images such that their scores are higher than a threshold τ and minimum number of images I_{min} must satisfy this criterion. These criteria are provided in equations (6.3) and (6.4). Note, in experiments, the values of $\tau = 0.6$ and $I_{min} = 2$ images (*i.e.* $|\mathbf{Q}_r(p)| \geq 2$) are used. The proposed score generalizes the one from [124] and it is identical to it when using $\tau \leq -1$ and $I_{min} = 0$.

Second, this thesis also proposes using visibility constrains to assign labels to certain voxels with high confidence prior to optimization. For instance, voxels that lie in the line of sight of an estimated 3-d point between the 3-d point and the camera center, must be clearly unoccupied (*outside*). Conversely, voxels that lie immediately behind a surface are occupied. The proposed hypothesis is that voxels that lie near 3-d points (4 voxels or closer) are putative locations for the surface and must be left on the graph cuts optimization, while others may be classified as *inside* or *outside* in advance. Among these other voxels, the ones that are deemed unoccupied by lines of sight of at least 40% of the images from cameras that look towards them are assigned to label *outside*. Analogously, the ones that are beyond an estimated point for at least 95% of the modeling images are deemed of label *inside*. This pre-estimation drastically reduce the size of the optimization problem such that essentially only voxels near a thin crust around the estimated point cloud enter the optimization. Such voxels are normally near the surface. Thus, the problem was reduced from a $O(N^3)$ to $O(N^2)$ if the uniform voxel grid has size $N \times N \times N$ voxels, remarkably reducing computation time and memory usage. The resulting surface from the search for the minimum cost solution on a 3-d grid of

size $200 \times 200 \times 200$ can be found in average in a few seconds on a CPU using the max-flow/min-cut library (version 3.01) from [16].

Finally, the assumption that the voxels at the borders of the volume are of label *outside* only hold in controlled environment. In aerial scenes for instance, with a volume centered at the ground level to model a city, it is normal to expect the top layer of voxels to be air, *i.e.* empty space of label *outside*. However, the bottom layer is under the ground, *i.e.* *inside* the terrain. In addition, the border layers at the sides have, in general, *both* labels since the ground must extend to all directions. Assigning border voxels that are truly inside the surface to *outside* causes undesired effects as holes in the reconstructed ground.

The proposed pre-estimation based on learned visibility handles this problem by assigning appropriate prior labels to borders as well as the interior of the volume. Section 8.4 shows experimental results and evaluation of surfaces estimation via the proposed method including indoor and aerial scenes.

Chapter 7

GPU Accelerated Template Matching

This chapter describes the proposed parallel implementation of normalized cross-correlation that is used in this thesis for feature matching (chapters 4 and 5) and multiple view stereo (chapter 6). The additional auxiliary functions used by the applications are also discussed. The implementation is geared towards general purpose computing on modern commodity graphics hardware, whose architecture is introduced in detail due to its relevance for the proposed implementation.

The proposed parallel implementation of normalized cross-correlation runs on a Graphics Processing Unit (GPU), a high-performance many-core processor. This chapter presents the proposed distribution of work among GPU cores that **exploits the GPU memory cache** to achieve high speed memory access, avoids the use of synchronization barriers and circumvents the shortcomings of GPU local memory altogether. Moreover, the memory footprint of the proposed program can adapt to hardware with small memory sizes. The proposed implementation is contrasted to an optimized version for a sequential Central processing unit (CPU) and to alternative parallel methods. For instance, the attained speedup on the correlation computation (ignoring image transfer overhead) when compared to a very optimized sequential CPU implementation can be, under certain relevant conditions, remarkably up to 950X or more, and up to 14X or more when compared to the OpenCV

library's [114] GPU implementation of normalized cross-correlation (experiments run on the same GPU). Modern hardware is used in these experiments and are described in chapter 8.

7.1 General Purpose Graphics Hardware

GPUs are specialized hardware designed to rapidly process information to render images for display and are often used in computer graphics. Due to development of high level languages for parallel programming, modern GPUs that have phenomenal processing power are now suitable for general purpose computing, which consists of employing GPUs to handle computations that are traditionally intended for CPUs. Such general purpose GPUs are known as GPGPU [43]. GPUs present many processors making them more effective than CPUs for algorithms where processing of blocks of data can be done independently in parallel. Parallel computing consists of performing arithmetic calculations simultaneously assuming a problem can be solved by partitioning it into smaller problems that are concurrently handled by multiple processors.

7.1.1 Typical GPU Architecture

GPU hardware and functionalities vary with vendors and models, and also evolve with time. This section and the following ones are intended to discuss recent GPU technology. The designs of the major manufacturers to date have several common properties that allow their devices to be used almost interchangeably by developers implementing their parallel applications. Note, device-specific tweaks may improve performance. In this thesis, the discussion focuses on GPUs from Advanced Micro Devices (AMD) and Nvidia Corporation (Nvidia) manufactures, and the cross-platform language used in the proposed implementations is OpenCL [1, 88, 87]. OpenCL (Open Computing Language) is an open parallel computing framework for writing programs that execute across heterogeneous multi-core platforms, such as CPUs and GPUs, and on multiple operating systems. A competing proprietary framework for GPUs is CUDA from Nvidia.

Typical GPU devices are divided into independent multiprocessors denoted *compute units*, which share global GPU resources. Each compute unit (CU) is composed of multithreaded cores (a.k.a. scalar processors) and private resources, including local memory and registers. Each physical core is

responsible for the execution of a number of *threads*, elementary sets of instructions multiplexed in time sharing the processing time of the physical core. Registers are where threads store their local variables. Multiple threads are pipelined on single core (hyper-threading) to hide latencies due to memory accesses and thread operations [1]. GPU components often use time-division multiplexing to share a physical hardware component among multiple processes so that a fraction of the time is allocated for each process. The reason behind all multiplexing operations happening on a GPU is to hide memory latency, discussed in detail in section 7.1.2. In general, threads switch so frequently a human user will perceive them as running at the same time. Therefore, a single physical core is running the operation of multiple (virtual) cores, called *logical cores*. One logical core executes one thread.

Each CU is capable of executing as many simultaneous threads as the number of its cores times the amount of threads handled by a core. To date, modern GPUs have around 30 CUs and each one is composed of typically 8 or 16 multithreaded physical cores where each one operates roughly 4 threads, yielding hundreds to thousands of logical threads per GPU device. Some modern Nvidia GPUs effectively execute 32 simultaneous threads per CU. As for AMD counterparts, 64 threads per CU. This set of concurrent executing threads is denoted a *warp* (a.k.a. *wavefront*). Similarly to the multithreading of cores, a CU manages the operations of multiple warps multiplexed in time, given the hardware can physically support only one warp execution at a time. A group of warps executed by a CU is denoted a *thread block* (a.k.a. a *work-group*) or simply a *block*. A CU is in charge of multiple thread blocks and it executes them sequentially. Multiple blocks are evenly and automatically distributed by hardware among available CUs. CUs will handle more than one block when there are less units in the GPU than the number of requested blocks. In fact, implementations of methods that assign many thread blocks to each CU tend to be faster than others that distribute only a few, since a high number of thread blocks hides more memory latency.

The hierarchy of GPU parallelism is then summarized as follows: a GPU device has several CUs, each of which executes multiplexed thread blocks. Every thread block is in charge of one or more multiplexed warps. A warp runs on multiple physical cores of a CU, all of which execute a fixed number of multiplexed threads. Often the number of physical hardware components is less than

the number of associated logical requests, demanding the aforementioned time-division multiplexing (TDM) of threads, warps and thread blocks. TDM is actually sequential, so the truly parallel components of the GPU are the multiple CUs and their multiple cores, although TDM is essential to achieve high computational throughput by hiding memory latency.

7.1.2 GPU memory model and cache

There are four typical memory domains: global, local, constant and private (see figure 7.1). Each domain has some benefits and limitations.

Global memory. Global memory is the largest memory, it can be accessed by any thread, but has high latency, usually requiring hundreds of cycles to respond to an access request. The global memory is shared by all CUs of the GPU device. Global memory fulfills parallel requests to access data from multiple concurrent threads, however, its bandwidth may drop significantly if the parallel access does not follow specific patterns and present spatial locality.

Local memory. Local memory is an on-chip low-latency high-bandwidth memory specific of a thread block of a CU. This memory is divided into a few banks of limited size and can only be accessed by threads from a given thread block. Local memory (also known as *shared memory*) can be accessed in parallel to fulfill requests within a few cycles (very low latency), but access may be serialized reducing bandwidth if multiple threads access the same bank simultaneously, causing a *bank conflict*. Local memory is the way threads share information efficiently within a thread block and is an appropriate location to store data that is frequently accessed. Global memory can also be used for sharing and is recommended for large data structures that are too large to fit in other memory domains.

Private memory. Private memory consists of registers where a thread stores its local variables and is not visible to other threads. Private memory is very fast, but limited in size. Registers are not indexable, then arrays of arbitrary size cannot be allocated in private scope.

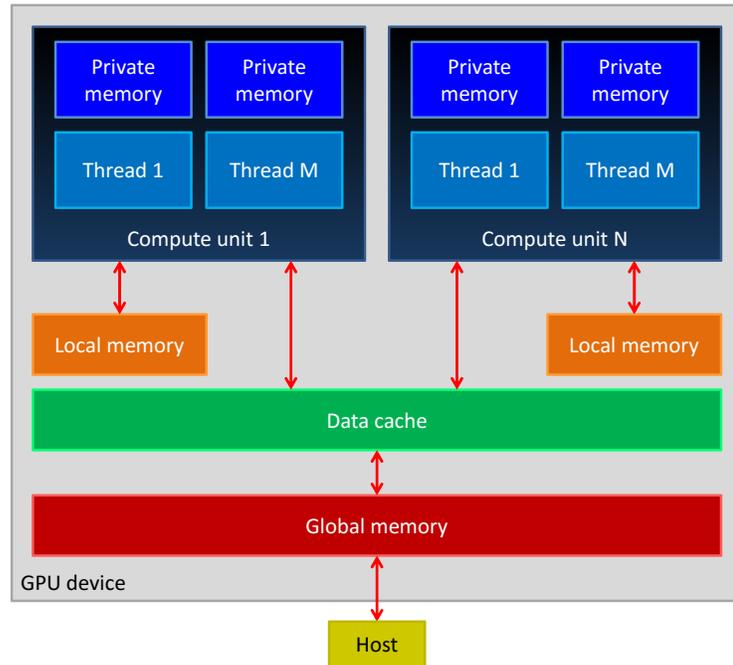


Figure 7.1: GPU memory spaces. Host is normally a computer that controls the GPU.

Constant memory. The term *constant memory* usually refers to a read-only region of global memory specialized in broadcasting data.

Cache. Another important memory space is *cache* memory, which has low latency and high bandwidth. The cache automatically stores data recently accessed or computed to avoid latency costs on future requests to the same data. Global memory is often cached and local memory may also be cached. Cache operates both on reading and writing operations. It is able to accurately predict requests when the access patterns exhibit temporal locality (when the same value is requested multiple times) or spatial locality (the accessed data is stored in the same physical neighborhood). When requested data is in the cache (cache hit), it is simply read from (or written to) the cache. Otherwise (cache miss), the data is fetched from memory as usual. The cache bandwidth is comparatively higher than global memory and comparable to local memory. Cache sizes are normally orders of magnitude smaller than their associated memories. The larger the cache size, the faster are the overall memory requests contributing to the peak calculation speed.

Access patterns. Global and local memories are parallel storage locations that simultaneously serve requests to multiple threads, with the disadvantage of decreased bandwidth if the access does not follow particular patterns. Local memory is divided into a fixed number of independent storage banks. If each thread in a warp access a distinct bank, read or write instructions are performed in a single parallel access. Otherwise, the access is serialized into as many successive accesses as necessary to resolve the bank conflicts. Global memory operates in an analogous way. Global memory highest speeds are achieved when accessing *contiguous* and *aligned* memory addresses (**coalesced** access pattern), otherwise access may be serialized (non-coalesced access pattern). Simple examples of global memory access patterns are given in figure 7.2.

Coalescing global memory can be a difficult task as it is not always trivial or possible to design a data storage structure that can always be accessed in such patterns. In addition, the supported patterns may change remarkably between GPU models and manufactures. In earlier models, coalescing was supported only by very limited patterns, whereas newer models tolerate more diverse coalesced patterns. Developers should refer to programming guides that are specific to target hardware, *e.g.* [1, 87], to learn their supported coalesced patterns, as there may be very peculiar differences that are too detailed to discuss here.

Cache importance. The aforementioned drawbacks of limited patterns for achieving GPU memory high-speed access suggest it is important to take advantage of the cache, as its peak memory bandwidth is in the same order of magnitude to one of the local memory and cache does not require special access patterns, except temporal and spatial locality. Cache and local memory offer significant advantage when the data is re-used and reduce global memory requests. Although the bandwidth of cache is regularly lower than the peak bandwidth of local memory, cache may often be faster given that local memory suffering from bank conflicts reaches only a fraction of its peak bandwidth. The cache has a key role in the proposed GPU implementation of normalized cross-correlation matching.

Data flow. The flow of data between host device (CPU) memory and GPU memory is illustrated in figure 7.3. The data must first be transferred from host memory to GPU global memory, from where it may go to local memory or directly to private memory for processing. Communication

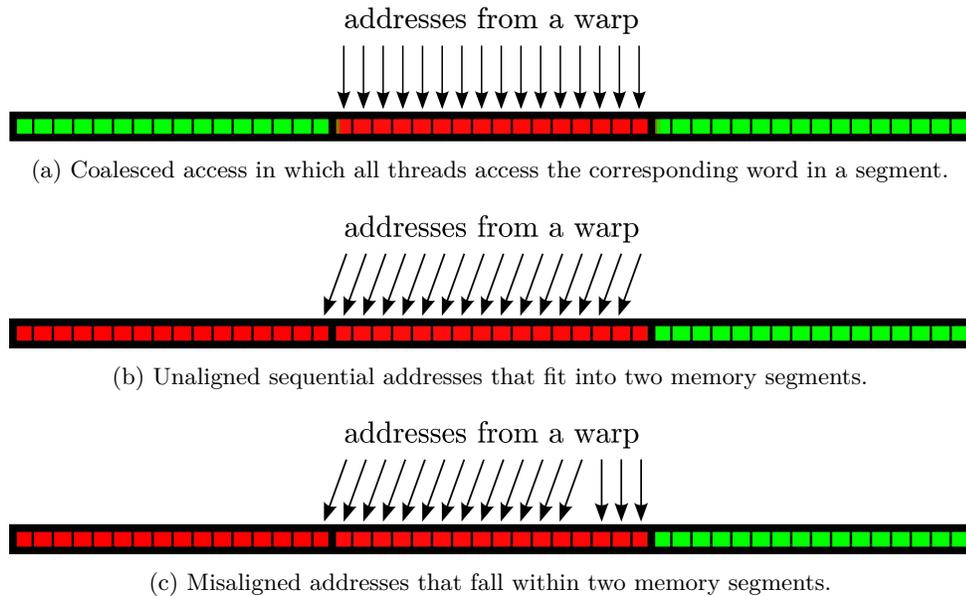


Figure 7.2: Simple examples of global memory access patterns.

between local and private memory is also supported. Transfers from host to device memory are performed through a bus and are faster for large data blocks, whereas a large number of small ones are transferred at low bandwidth.

7.1.3 Parallel synchronization

In sequential algorithms, instructions are executed in order. Thus, variables that have been defined at some point in time are immediately ready to be accessed at any further instruction, but this is not guaranteed in parallel computing.

In parallel algorithms, the address where data is shared by threads can be accessed (in parallel) by other threads at any time and there is no inherent control of the sharing and fetching order as in sequential programs. If a fetching access happens in a location before the sharing is complete, meaningless data is read and processed inducing a silent failure. These problems of parallel threads interfering with each other are called *race conditions*.

Synchronization of threads prevents the access of data before it is actually ready. The synchronization *barriers* are software instructions placed after a sharing write instruction and before the

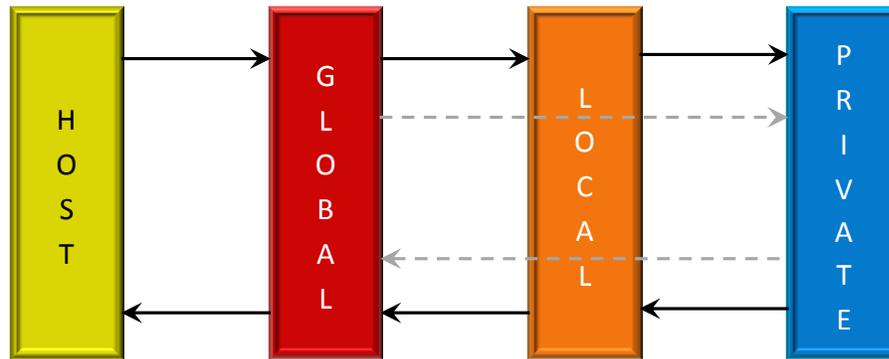


Figure 7.3: GPU memory data flow. The data is originally resident on host memory. It is transferred to the GPU through the PCI Express bus and stored in global memory. From global memory, data may flow to shared memory or directly to private memory, where it is processed.

associated fetching read instruction. Synchronization barriers require all threads in a block to wait at the instruction until all threads have reached it, thus enabling control of memory access order. Synchronization barriers prevent the hardware cores from running without interruption, causing them to stall until all are ready for the sharing operation. The GPU performance may be highly affected by interrupting too often. Barriers can be very detrimental as all threads need to be interrupted even to synchronize just a few.

Unlike threads in a thread block, compute units in a GPU carry their operations completely independent of each other and cannot be synchronized in the same way using barriers. Even if one attempts to synchronize CUs by reading and writing to a location in global memory, which all CUs have access to, and let processing continue conditionally on all CUs setting a flag indicating they reached an instruction, the effort can fail. The attempt will not succeed if a CU is handling multiple thread blocks and a thread block never stalls, and as a result other blocks are never activated. Hence, all active blocks will be endlessly waiting for the others that are indefinitely idle and are never initialized to set their flags.

Atomic operations. Atomic operations are operations capable of reading, modifying and writing to a shared memory location without interference of other threads. For instance, assume two threads try to increment the value of a memory location by one. Depending on the order the threads access the value, both may access the same value, then increment it by one and write the result back, however

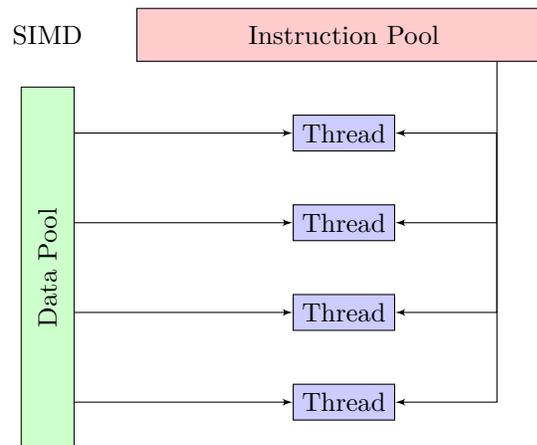


Figure 7.4: SIMD (Single Instruction, Multiple Data) multi-core computer architecture. Threads concurrently process different data streams, but all execute the same instruction at any given time.

the values were supposed to be incremented twice. Atomic operations are designed to avoid these race conditions since they happen atomically (without interruption) and ensure only one thread has access to the memory location while the operation completes. Furthermore, atomic functions appear to occur in parallel, but are in fact serialized and synchronized by hardware barriers. Atomic serialization has no specific order and its synchronization is relatively faster than equivalent synchronization by software barriers. Thus, atomic operations should be preferred to software synchronizations when synchronization is unavoidable. Atomic operations can perform on local or global memory scopes, and when done on global scope, they may provide a way to global synchronization [1].

7.1.4 Data-parallel instructions

Thread instructions execute sequentially in SIMD (Single Instruction, Multiple Data) paradigm. As depicted in figure 7.4, the data-parallel process exploits parallelism by accessing multiple data streams (one per thread) against a single instruction stream (all threads on a warp perform a common command at any given time). Note, warps are multiplexed in time and threads of a single warp execute instructions concurrently (SIMD paradigm), so only threads across different warps of a block need synchronization.

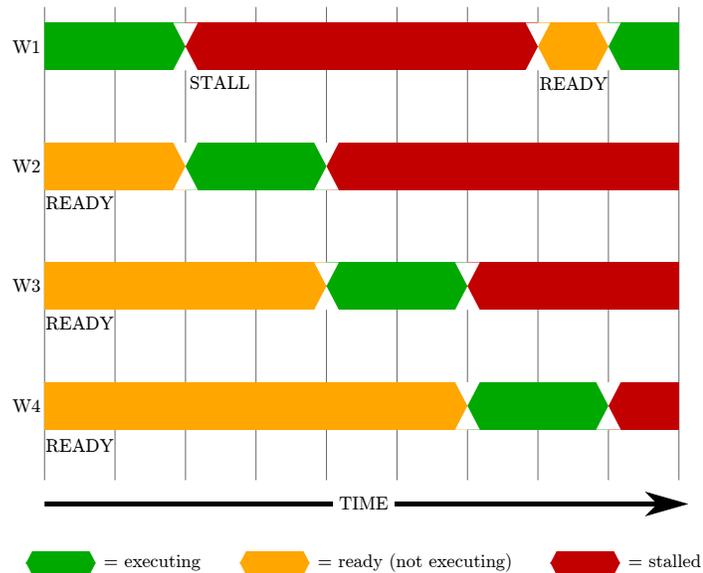


Figure 7.5: Schematic representation of the execution of warps on a single thread block for hiding latency. Analogously, the diagram also represents the execution of thread blocks in a single CU. When warp W4 stalls, execution returns to warp W1 if W1 is ready, otherwise execution returns to a new active thread block that is ready.

7.1.5 Memory latency and performance

Each CU has its own sets of resources: scalar processors, registers and low-latency local memory. The way these resources are allocated for a given program may have a substantial impact in performance.

Hiding latency. A CU executes SIMD instructions on multiple warps by first executing a set of them on the first warp until it stalls, typically waiting for a high-latency memory access. The way to keep the hardware busy after a warp is interrupted is to immediately execute other warps. Then the CU switches execution to a another warp which will execute the same stream of instructions and likely stall at the same place as the first warp (see figure 7.5). The switching repeats until the last warp. Once the last warp stalls, the CU switches back to the first warp if it is ready, otherwise it switches to another block (a different set of warps) and returns later. Switching between active blocks hides block latency when all warps in a thread block have stalled waiting for memory. In summary, memory transactions and mathematical computations are overlapping in time so latency can be hidden. Note, it is necessary that there is enough instructions between each programmed stall in order to achieve the overlap, otherwise memory latency is exposed and the hardware is idle.

7.1.6 Active threads, warps and blocks.

In order to switch back and forth between warps and blocks, their data must be allocated and stored somewhere, and be ready to execute at any time, which means to be *active*. The higher the number of active (allocated) warps and blocks, the more latency can be hidden and better is the utilization of the available computing resources. For instance, running only one block per CU will likely force the CU to idle during thread synchronization and also during memory access. The number of active threads per block is designed by user and defined by software, unlike active warps and active blocks, which are determined by hardware in terms of availability of registers and local memory, which are limited commodities allocated to an entire CU all at once and shared among all resident threads. The amount of resources allocated for a thread, warp or block depends on the program, and the less the better since higher is the number of active warps and more latency is hidden. The hardware becomes temporarily idle if the very first warp that stalled is not ready after the CU has switched to all other active warps of all other active blocks.

Occupancy. The ratio of the number of active warps to the highest possible number of active warps is denoted *occupancy*. There is a performance trade-off between high occupancy and code optimization as using more registers may translate into a more optimized code, but with less occupancy and more exposed latency. Developers need to be very experienced to determine which implementations may result in performance gains without trial and error.

7.1.7 Register spills

As described in section 7.1.6, high resources usage impacts performance by decreasing occupancy and increasing the risk of exposing memory latency and keeping the hardware idle. Another issue of concern is register *spills*. When private memory usage is higher than availability, a program may still run, however registers that cannot fit in private scope are allocated in larger memory scopes of higher latency. For instance, excess of registers may spill over cache, local memory, or global memory. This memory allocation happens automatically and is denoted a *spill*. Spilled registers may cause a large impact on performance as they are allocated in long-latency memory and access patterns are

not necessarily optimized for such memory. In addition, little latency is hidden if only one warp or block is active at a time due to high register usage. Register spills can be detected by compilers and then avoided, requiring possibly complicated code redesigns.

7.1.8 Parallel functions

In order for algorithms to be suitable for a GPU, they must be defined as a series of independent computations with minimal sharing of information among them. Recursive algorithms are not suitable for GPU. Typically a function is defined as set of instructions to simultaneously run on multiple data. This function is denoted *kernel*. If a kernel instructs a thread to read a small amount of data and do a vast amount of processing with it, this kernel is suitable for general purpose computing on a GPU by optimally exploiting GPU architecture to achieve remarkable speedups compared to an equivalent sequential CPU program.

7.1.9 Summary of performance requirements

According to discussion of section 7.1, quality parallel algorithms are difficult to design as they are expected to satisfy many requirements in order to present outstanding acceleration . These expected attributes are summarized as follows:

- the algorithm is not sequential;
- global memory accesses are coalesced;
- local memory bank conflicts are avoided;
- data fits in global memory;
- data fits in local memory;
- register usage is relatively low;
- memory latency is hidden;
- there are no register spills;
- there is little synchronization;
- many warps and blocks are active;
- communication with host has high bandwidth;
- cache hits happen more often than cache misses;
- processing instructions occur more often than memory accesses.

7.2 Normalized cross-correlation

The facts about GPU architecture and performance optimizations presented in section 7.1 are adopted to introduce a very efficient high-performance parallel implementation of template matching using normalized cross-correlation (section 7.3). This proposed method was designed for any general template matching application where there is enough data and processing requests to enable high data parallelism. The proposed parallel implementation is demonstrated on three applications presented in this thesis: feature matching (discussed in chapter 4), match disambiguation for dense repeating features (discussed in chapter 5) and multiple view stereo (discussed in chapter 6). Other parallel portions of the implementations of these applications are described in sections 7.4 and 7.5. This section discusses details pertinent to the normalized cross-correlation algorithm, such as its complexity and an optimized sequential implementation for CPU that is contrasted to the proposed GPU method. Experiments and evaluations are provided in chapter 8.

7.2.1 Expressing NCC in terms of sums

Normalized cross-correlation (NCC) is computed between a rectangular template window of size $t_x \times t_y$ and a larger search window of size $S_x \times S_y$. For every position for which the template fits inside the search window, a numerical correlation coefficient is computed using the pixel values of the template and the associated pixel values in the underlying search window. The output is a correlation array of size $C_x \times C_y$ such that $C_x = S_x - t_x + 1$ and $C_y = S_y - t_y + 1$.

The correlation function is discussed in all details in section 3.2 and it is defined in equation (3.7). The definition is repeated here in equation (7.1) for convenience:

$$C(u, v) = \frac{\sum_{x,y} \left\{ w(x, y) [S(x+u, y+v) - \overline{S_w(u, v)}] [t(x, y) - \overline{t_w}] \right\}}{\sqrt{\sum_{x,y} \left\{ w(x, y) [S(x+u, y+v) - \overline{S_w(u, v)}]^2 \right\} \sum_{x,y} \left\{ w(x, y) [t(x, y) - \overline{t_w}]^2 \right\}}}. \quad (7.1)$$

To simplify notation, let $w_{xy} = w(x, y)$, $S_{xyuv} = S(x+u, y+v)$ and $t_{xy} = t(x, y)$. By expanding products and plugging the mean terms defined in equations (3.8) and (3.9), Equation (7.1) can be

reformulated as

$$C(u, v) = \frac{\sum_{x,y} [w_{xy} S_{xyuv} t_{xy}] - \frac{\sum_{x,y} [w_{xy} S_{xyuv}] \sum_{x,y} [w_{xy} t_{xy}]}{\sum_{x,y} w_{xy}}}{\sqrt{\sum_{x,y} w_{xy} S_{xyuv}^2 - \frac{(\sum_{x,y} w_{xy} S_{xyuv})^2}{\sum_{x,y} w_{xy}}} \sqrt{\sum_{x,y} w_{xy} t_{xy}^2 - \frac{(\sum_{x,y} w_{xy} t_{xy})^2}{\sum_{x,y} w_{xy}}}}. \quad (7.2)$$

Assuming arbitrary weights that are normalized,

$$\sum_{x,y} w_{xy} = 1, \quad (7.3)$$

and using the notation $\sum_{x,y} [w_{xy} H(x, y)] = \sum_{x,y,w} [H(x, y)]$, equation (7.2) can be further simplified to

$$C(u, v) = \frac{\sum_{x,y,w} [S_{xyuv} t_{xy}] - \sum_{x,y,w} [S_{xyuv}] \sum_{x,y,w} [t_{xy}]}{\sqrt{\sum_{x,y,w} S_{xyuv}^2 - (\sum_{x,y,w} S_{xyuv})^2} \sqrt{\sum_{x,y,w} t_{xy}^2 - (\sum_{x,y,w} t_{xy})^2}}. \quad (7.4)$$

Let

$$S_1 = \sum_{x,y,w} t_{xy}, \quad (7.5)$$

$$S_2 = \sum_{x,y,w} t_{xy}, \quad (7.6)$$

$$S_3 = \sum_{x,y,w} [S_{xyuv} t_{xy}], \quad (7.7)$$

$$S_4 = \sum_{x,y,w} S_{xyuv}, \quad (7.8)$$

$$S_5 = \sum_{x,y,w} S_{xyuv}^2, \quad (7.9)$$

then equation (7.4) may be rewritten in terms of five weighted sums:

$$C(u, v) = \frac{S_1 - S_2 S_4}{\sqrt{(S_3 - S_2^2)(S_5 - S_4^2)}}, \quad (7.10)$$

Note, S_1 and S_2 depend only on the template and need only be computed once for all (u, v) , therefore, the correlation coefficient computation is reduced to computing the three weighted sums S_3 , S_4 and S_5 for every (u, v) in equation (7.10) and a few additional algebraic operations. The sums are $O(N)$ where N is the number of pixels in the template.

7.2.2 Sequential NCC algorithm complexity

The computation of NCC from equation (7.1) can be expressed either directly in the spatial domain or in the frequency domain using the Fourier transform. This applies to both a sequential or a

parallel implementation. In practice, spectral computation is faster only when the size of S and t are approximately the same and are both large [69], as discussed below.

Complexity of the denominator of NCC

The energy terms in the denominator of equation (7.1) represent sample standard deviations responsible for normalization of the signal amplitudes. Lewis [69] presents a very efficient way to compute the energies and the means of $S(x, y)$ in terms of the integral images (running sums) [28] of S and S^2 for any position (u, v) . After the construction of sum-tables the energies can be computed in constant time using only four array references independent of the size of the template. An alternative similar approach is to exploit the redundancy of the sum terms as the template shifts only by one pixel: compute a sum S_α at a starting location and when the template window shifts by one pixel, *e.g.*, to the right, subtract from S_α the sum of the removed column and add the sum of the inserted column. The result is the sum for the new location, which achieves similar performance as the sum-tables. Hence, the bulk of the computation of NCC lies on the cross-correlation term in the numerator as it cannot be precomputed.

Complexity of the numerator of NCC

The numerator of equation (7.1) can be reformulated as a convolution, which is expensive to compute in the spatial domain, but less complex in the transform domain via a simple product of Fast Fourier Transforms (FFT). However, the additional costs of computing the FFT of $S(x, y)$ and $t(x, y)$ and the inverse FFT of the resulting product is not practical for cases where the template is much smaller than the search window since S and t must be extended with zeros to a common power of two (zero padding). In addition, FFT involves floating-point computations, whereas the spatial domain methods can be implemented mostly using integer operations [119], which are often faster. In fact, assuming the size of $S(x, y)$ is M^2 and the size of $t(x, y)$ is N^2 , [69] indicates that the complexity of computing the numerator of equation (7.1) ignoring the mean terms is: (1) $12M^2 \log_2 M$ real multiplications and $18M^2 \log_2 M$ real additions/subtractions via FFT, and (2) $N^2(M - N + 1)^2$ additions/multiplications via the direct method. Hence, when $M \gg N$ the complexity of the spatial method for computing the same numerator is approximately N^2M^2 multiplications/additions and the spatial method is faster

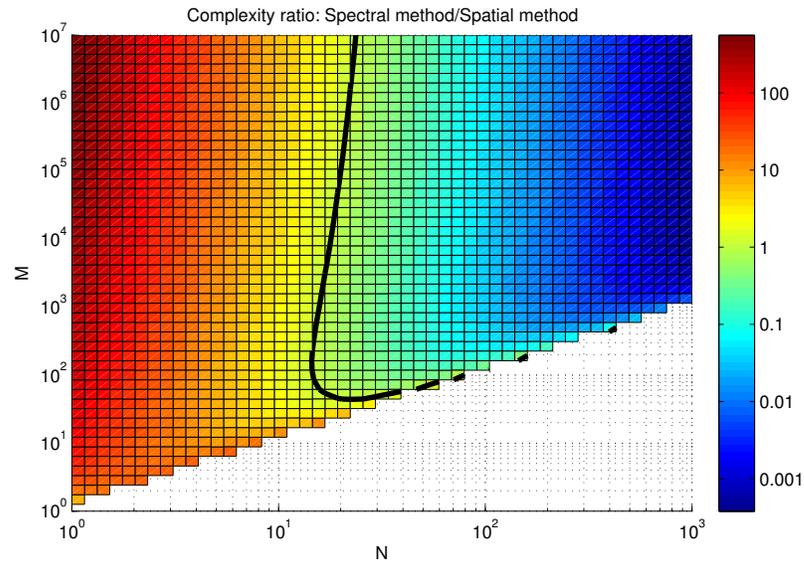


Figure 7.6: Ratio of the algorithm complexities for computing the numerator of NCC using a spectral method ($48M^2 \log_2 M$) versus a spatial method ($2N^2(M - N + 1)^2$). The axes (N, M) represent a square template of size N^2 and a search window of size M^2 . The implementation complexities are equal at the 1 level contour, shown in black. Note, the algorithms are only meaningful when $M \geq N$.

than the spectral one. Figure 7.6 illustrates the ratio of the aforementioned complexities of the NCC numerator algorithm in the two domains, showing that for small templates the spatial method is more efficient. Note, the ratio plot presented in figure 7.6 can only motivate approximate conclusions since additions and multiplications in either integer or floating-point representations are considered equally complex, and transcendental function evaluations are omitted.

Preferred domain for parallel NCC algorithm

Parallel implementations running on a parallel computer with p processors reduce an algorithm complexity by a factor of at most p , according to Amdahl's law [99, 7]. The upper bound p of the speedup can be reached only if the fraction of the algorithm that is strictly serial is 0. Spatial and spectral parallel implementations must follow similar complexity reductions as both methods allow broad parallelization. Therefore, in light of the complexity analysis in figure 7.6, a spatial parallel implementation is appropriate for the stereo matching applications of this thesis, which use small templates.

In chapter 8 a comparison of the proposed spatial parallel algorithm of NCC with the optimized parallel implementation of NCC of OpenCV [114] is provided. OpenCV has a spectral and a spatial method, and both are outperformed for small templates and large search windows (see section 8.1.3).

The proposed implementation speedup over an optimized sequential CPU version is in the order of several hundreds on modern high-performance hardware (see chapter 8). The optimized CPU method is discussed in section 7.2.3.

7.2.3 Optimized CPU implementation

This section describes the optimized CPU implementation of NCC that competes with the proposed parallel one. The sequential implementation of NCC is carried out by computing the sums S_1 to S_5 in equations (7.5) to (7.9) in section 7.2.1. As suggested in section 7.2.2, the template terms S_1 and S_2 are evaluated only once, taking negligible time in general, and the denominator terms S_4 and S_5 are efficiently computed in terms of running sums (integral images). The numerator of the NCC expression cannot be precomputed in the same way as the denominator so the numerator crossed term S_3 is carried out as a convolution. Note, the number of operations to compute the integral images is very small compared to the cost of computing S_3 [69], and other fast algorithms have been proposed [135, 18, 45, 120, 91], however these are approximations and have shortcomings.

7.2.4 Disregarding integral images for parallel NCC

The construction of integral images discussed in section 7.2.2 involves cumulative sums which are mainly sequential operations and harder to parallelize. Integral images are not used in the proposed parallel algorithm as the additional time to redundantly compute the energy terms everywhere in parallel is likely smaller or in the same order of the short time to compute the tables. The reason is that the elements t_i and s_i of a template and of an underlying patch, as illustrated in in figure 7.10, need be accessed anyhow by a thread to compute the inner product $\sum_i t_i s_i$ of equation (7.7), and the additional time for computing $\sum_i s_i$ and $\sum_i s_i^2$ in parallel is small since the values are already in private memory (the fastest memory). Moreover, the tables would be stored in global memory due to their sizes being as large as $S(x, y)$, and this incurs additional memory access overhead.

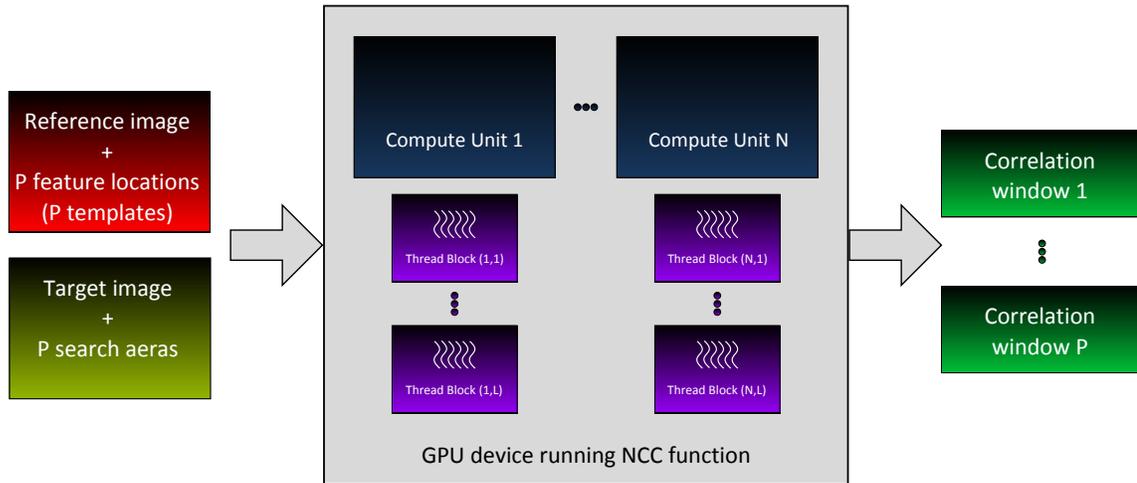


Figure 7.7: Overview of proposed parallel NCC implementation running on a GPU device responsible for computing P correlation arrays. The computations are performed via thread blocks and are distributed as evenly as possible among the N CUs. If the division is not exact, some CUs will be responsible for one less block than the others, while jointly they complete all P tasks.

7.3 Proposed parallel NCC

The goal of the algorithm is to compute $C(u, v)$ given a template $t(x, y)$, a weight function $w(x, y)$ and a search window $S(x, y)$. The implementation of NCC can be parallelized in several different ways, but in order to get a high computational throughput the entire GPU hardware must be busy at all times and hide as much global memory latency as possible, which varies with the amount of active warps and blocks, use of synchronization barriers, access patterns and other factors.

Memory latency may be completely hidden if each CU is responsible for many active blocks and each block executes many active warps. Thus, the proposed solution requires every CU to be responsible for the computation of a number of correlation arrays and each thread of a warp to handle a number of elements of the array. Many correlation applications can be formulated in such way. For instance, in the context of stereo matching, thousands of templates $t(x, y)$ arise as patches centered at interest point locations, while each of their search windows $S(x, y)$ may have thousands of pixels, as the case of long strips around associated epipolar lines, which leads to many tasks per processing units as typical high-end GPUs to date have in the order of 30 CUs and 32 threads per warp.

The parallel implementation also requires limited use resources since overuse will limit active warps and blocks, and reduce block and warp switching used for hiding memory latency. Registers

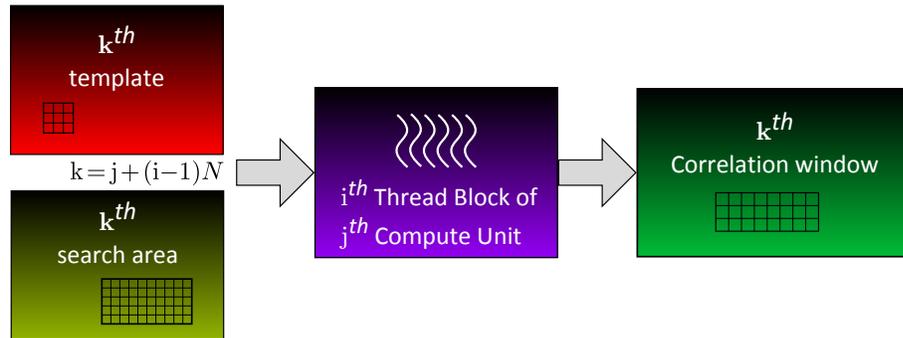


Figure 7.8: An operation of *loop 1*, the outermost loop of the proposed GPU implementation. The parallel operations indexed by k run on thread blocks of CUs, each receiving a template and a search window, and returning a correlation array. The indices i , j and k indicate to which CU each task is assigned and that the thread block jobs are distributed as evenly as possible among N CUs.

(private memory) are more limited than local or global memory for this parallel implementation, then the code transfers some storage to local memory when necessary. Other requirements are listed in section 7.1.9.

7.3.1 Distribution of work

The parallel computation of NCC for multiple template matching operations in an image pair is outlined in figure 7.7 and has three main loops specifically designed to be faithful in accordance with performance requirements listed in section 7.1.9. Details on how these goals are achieved and the descriptions of the loops are given below:

- Loop 1:** For every $t(x, y)$ and $S(x, y)$, a correlation array C is computed;
- Loop 2:** for each C , a correlation coefficient $C(u, v)$ is computed for every position (u, v) ; and,
- Loop 3:** for each $C(u, v)$, the sums in equation (7.1) are computed iterating over (x, y) .

Loop 1 is parallelized among CUs, *i.e.*, thread blocks run in parallel in CUs and each one handles a feature point. A CU is responsible for multiple thread block jobs, as in figure 7.8. Loop 2 computes $C(u, v)$ for every position that $t(x, y)$ fits inside $S(x, y)$ and is distributed among threads of a block, meaning threads handle the computations of elements of C in parallel. The pixels read from $w(x, y)$, $t(x, y)$, $S(x, y)$, and written to $C(u, v)$ are accessed in coalesced patterns (see figure 7.9). Loop 3 iterates over all pixel locations (x, y) of a template $t(x, y)$ and its associated underlying pixels

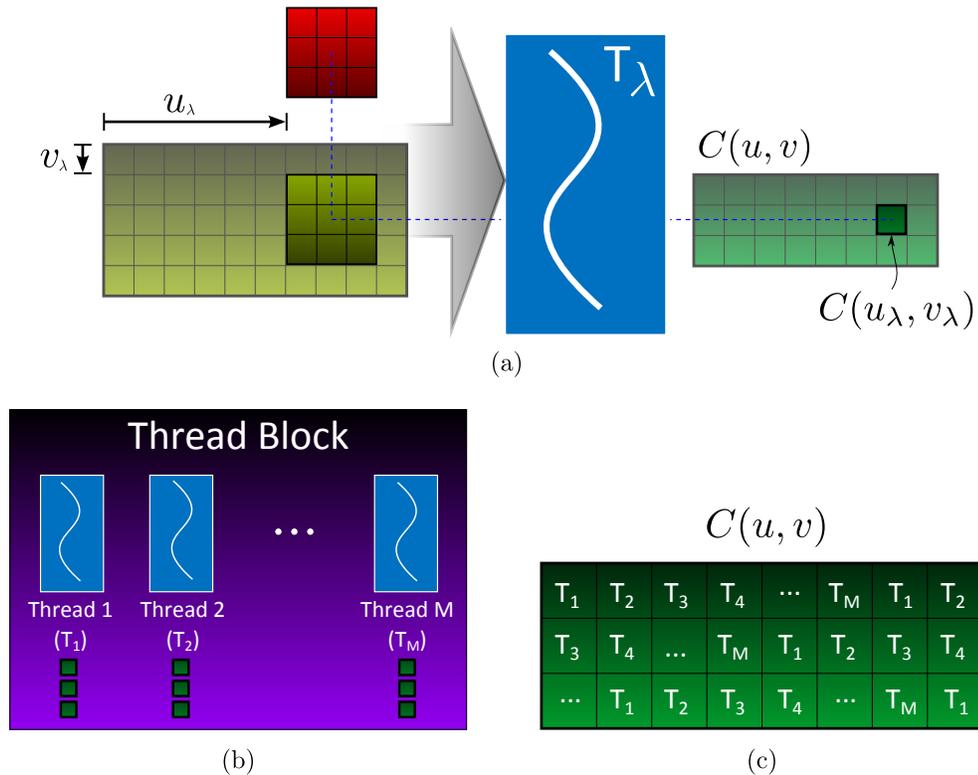


Figure 7.9: Operations of *loop 2*, the intermediate loop of the proposed GPU implementation. (a) The parallel operation indexed by λ runs on a thread T_λ of a thread block, receives a template patch and an underlying patch of the search window, and returns one element of the correlation array. The set of all such parallel operations composes *loop 2*, resulting in multiple correlation elements per thread. (b) The division of work among threads working in parallel. Each one of M threads is responsible for the sequential computation of multiple coefficients of $C(u, v)$. (c) Assignment table associating elements of the correlation array to their respective computing threads. This assignment achieves a coalesced access pattern for elements stored in row-major order (rows of the array are contiguous in memory). Note, the distribution of tasks is as evenly as possible.

in $S(x, y)$ and $w(x, y)$. *Loop 3* is implemented sequentially within each thread and access pixels in a coalesced pattern in the order they appear in the template array. According to discussion at section 7.3, each thread computes the coefficient $C(u, v)$ via equation (7.10) by computing the sums S_1, S_2, S_3, S_4 and S_5 from equations (7.7) to (7.8). In fact, S_1 and S_2 from equations (7.5) and (7.6) are constant within the loop, therefore, they need only be computed once (per block) by the first thread prior to starting *loop 3* and are shared with the others through via local memory (see figure 7.10). Furthermore, if the center coordinates of the template are not integers, its pixel values are refined using bilinear interpolation. Note, the search window is required to be aligned

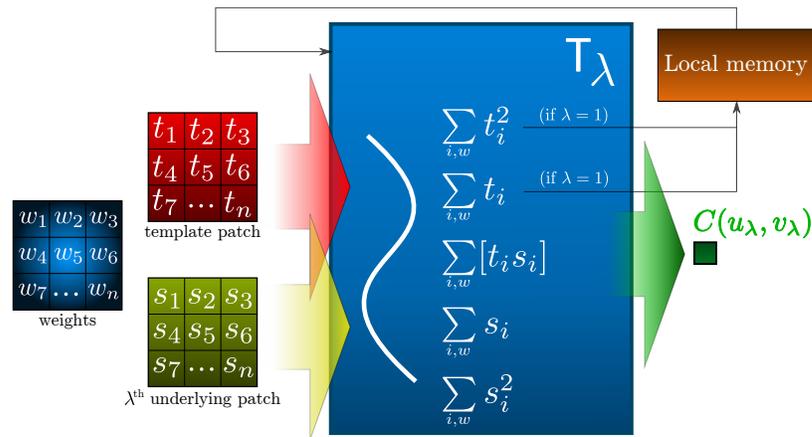


Figure 7.10: The operations of *loop 3*, the sequential innermost loop of the proposed GPU implementation. The loop runs serially on a single thread to accumulate sums indexed by i used to compute a correlation coefficient via equations (7.5) to (7.10). Only one thread (T_1) needs computing the top two sums and it shares them with other threads through local memory.

with the pixel grid and no interpolation is necessary. Moreover, subpixel refinements of interest point matching locations on the search window are discussed in section 7.4.6.

The proposed implementation does not use thread synchronization except once for computing S_1 and S_2 ahead of loop 3, and then loop 3 iterates only over S_3 , S_4 and S_5 computations. Thus, synchronization latency is negligible and this is very important for hiding latency, as described in section 7.1.5), since the warps stall only at memory accesses.

7.3.2 Memory allocations

The input arrays $t(x, y)$ and $S(x, y)$ must be transferred from host computer to global memory as in figure 7.3. For stereo matching applications two images must be transferred. The reference image provides templates as patches centered at interest points and search windows are located in the other image. The template has arbitrary size, but in general is much smaller than $S(x, y)$ for the stereo applications. The template is accessed for every position (u, v) of $C(u, v)$. Local memory is chosen to store the template since the template is small and accessed very often, which is appropriate for a fast and limited memory. Note, arrays cannot be allocated in private memory since registers are not indexable, therefore, local memory is the fastest storage location for the template array. The pixel values of $t(x, y)$ are stored in sizes matching local memory banks to avoid bank conflicts. The weight

array $w(x, y)$ is also small since it has the same size of the template, so it is computed once and also stored in local memory.

$S(x, y)$ is relatively larger and is kept in global memory. In fact, $S(x, y)$ is accessed directly as a subimage of its parent image, which resides in global memory. Transferring pieces of $S(x, y)$ to local memory for faster access seems reasonable, but it introduces complexity on the parallel code in order to handle the transfers, increasing resource usage (private memory registers) and reducing the number of active warps and blocks, potentially causing exposed latency and memory spills. Hence, $S(x, y)$ is left in global memory and accessed in a coalesced pattern. In addition, the proposed method also achieves cached accesses to $S(x, y)$ due to spatial and temporal locality of the accesses. In the example portrayed in figure 7.10, the pixels identified as $\{s_1, s_2, s_3\}$ of the subimage of $S(x, y)$ underlying the template are contiguous in memory (spatial locality), and, according to the assignment table in figure 7.9c, while a thread access $\{s_1, s_2, s_3\}$, the next thread also access $\{s_2, s_3\}$ and the following thread also access s_3 (temporal and spatial locality). Analogously, the access to the other rows are also localized. Hence, the proposed access pattern attains high speeds due to the high probability of cache hits. Cached access is very fast and comparable to local memory speeds (section 7.1.2), suggesting the use of local memory to be unnecessary. The use of cache is very important as it makes the implementation simpler than one using local memory, which could potentially suffer from bank conflicts, have more complex code and not be faster. These alternative approaches are discussed in section 7.3.3.

7.3.3 Alternative parallel NCC implementations

In light of the countless number of configurations that control GPU performance discussed in section 7.1, GPU programming may become very complicated as it is hard for the average programmer to predict overall performance of a parallel program. As stated by Lu *et al.* [75], most of the published GPU simulations achieving two or more orders of magnitude speedups have authors in computer-related fields with far more experience than typical experimentalists and the most impressive results are often achieved via programs specifically written for a particular GPU model. Although processor-specific programming may achieve speedups that are off the charts, they may require spectacular

programming feats.

One common way of unveiling whether a parallel method is better than an alternative one is in a trial-and-error fashion and since GPU programming is very time consuming, presenting the performance of other parallel NCC methods is also of importance. Therefore, a brief description of two alternative attempted methods that presented lower performance than the proposed one is also provided for information purposes. One alternative method attempts to transfer the search window to local memory to access it at higher bandwidth and the transfer is done in small increments as local memory has limited size. The other method distributes the parallel processing of correlation window coefficients among CUs in a similar but slightly different way than the proposed method. A summary of properties of the alternative methods is presented in table 7.1 and a more detailed description is given below.

Method A: search window in local memory

As already mentioned in section 7.3.2, the proposed algorithm transfers two images to global memory to perform matching using NCC. Templates are extracted from the reference image and stored in local memory, whereas search images $S(x, y)$ are accessed directly in global memory as subimages of the other image. Cache is exploited for fast access to $S(x, y)$.

As briefly discussed in section 7.3.2, transferring portions of $S(x, y)$ to local memory for faster access during loop 2 and loop 3 is an alternative approach. Experiments showed that using a sliding window for this purpose was not efficient. The sliding window stores in local memory a few rows of $S(x, y)$ where correlation is performed. After all operations in a given row are complete it is replaced with a new row where new operations will take place. The speedup of the attempted implementation over the single-core CPU implementation was poor and presented little speedup. The excessive synchronization and additional registers necessary to manage the sliding window, besides possible local memory bank conflicts, are likely accountable for the unsatisfactory speedup as these issues easily offset the advantage of parallelism.

Method properties	Proposed method	Method A	Method B
Search image location	Global memory	Local memory	Global memory
Heavy usage of global memory cache	Yes	No	Yes
Heavy use of synchronization	No	Yes	No
Subject to bank conflicts	No	Yes	No
Number of computed coefficients	Multiple per thread per block	Multiple per thread per block	One per thread per block
Number of search windows per block	One	One	Multiple
Typical usage of threads per block	Partial on uneven divisions	Partial on uneven divisions	100%
Processing of template data	Only once per block	Only once per block	Always: once per thread
Recommended usage case	Large $S(x, y)$	None	Small $S(x, y)$
Typical relative performance	Fastest	Slowest	Reasonably fast

Table 7.1: Comparison of different GPU implementations of normalized cross-correlation.

Method B: bypassing assignment of search windows to CUs

An alternative approach to NCC is to group together all the N_p pixels of all correlation arrays and distribute their computation among N_p threads, *i.e.*, each correlation coefficient is assigned to a different thread. The computation is done as in figure 7.10. Within a thread block, a thread handles only one coefficient and the block may access multiple search windows, unlike the proposed algorithm that distributes a search window to a single thread block regardless of size, in which a thread may handle multiple coefficients. Note, the workload of a thread is similar in the two cases since threads still compute multiple correlation coefficients, but in the alternative approach, the coefficients are associated to a distinct $S(x, y)$. The correlation coefficients of a search window are still kept contiguous in memory to enable coalesced write operations of each $C(u, v)$ window.

A benefit of this alternative method is to guarantee work for all threads of a block (except possibly one block if division is uneven), unlike the proposed method where some threads may work more than others (see figure 7.9c), or may not work at all if the correlation window is small and has less elements than the number of threads of the block. Idle threads are rare for the applications of this thesis under the proposed implementation.

The alternative approach has a drawback. In the proposed application, the template is constant in a thread block and need be accessed only once in global memory before it is transfered to local memory where it is shared among all threads of the block. Moreover, operations on the template, such as the sums in equations (7.5) and (7.6) also need be computed only once. Conversely, in the alternative approach, these operations that depend only on a template are redundantly repeated by many threads since the concurrent threads of a block may be processing different templates and sharing precomputed template information becomes impractical. Therefore, the template is always broadcast to many threads from global memory or its cache.

It is not obvious which parallel NCC method is faster given only the trade-offs highlighted above. The alternative method was also implemented and under some experiments it attains performance similar to the proposed one, but the alternative implementation is in roughly 2X to 4X slower. The variability depends on GPU models and window sizes.

7.4 Feature matching application

This section describes additional parallel functions used to implement the GPU portion of the feature matching method of chapter 4. This part of the method is discussed in detail in section 4.3.1 and involves finding the local maxima (peaks) of the correlation window and pruning them.

7.4.1 Overview of GPU function

Correlation coefficients compose the correlation window $C(u, v)$, but only the local peaks are of interest as they are an indication of a potential feature correspondence. Peaks are defined as values that are higher than their neighbors in the window grid. At the moment a coefficient is computed using the proposed parallel NCC method (section 7.3) there is no guarantee its neighborhood is

already computed. Therefore, the local maxima of $C(u, v)$ are computed separately by additional subsequent calls to a GPU peak detection kernel functions and the complete correlation windows must be stored in GPU global memory at that moment of the peak computations. Detected peaks are also pruned and refined by the GPU.

7.4.2 Handling memory allocations

Several windows $C(u, v)$ are computed by a single call to the parallel NCC kernel function (loop 1 in section 7.3) in order to achieve high computational throughput. However, the storage requirements for all correlation windows may be too large and may exceed the storage capacity of even some modern GPUs (global) memories. For instance, the allocation size for the windows may be 1.4 GB if an image has 7000 interest points, their associated correlation windows are each 50×1000 pixels, and each pixel stores a real number (4 bytes) that is the value of a correlation coefficient, then the allocation size is $7000 \times 50 \times 1000 \times 4 = 1.4 \cdot 10^9$ bytes or 1.4 GB. The GPU memory also needs to store other variables such as the image pairs and/or the ones to render a computer display. In addition, GPU manufacturers may impose a maximum limit on the size of an array stored in global memory, which may be only a few hundred megabytes. For these reasons, the implementation partitions the NCC kernel call into a sequential number of calls in which the data fits in GPU memory. In each partition, a block of windows is allocated in available memory space, their peaks are detected and stored, and the correlation windows are then promptly discarded. There is no harm to performance in carrying out such divisions as large correlation windows imply reaching high data parallelism required by the proposed method to achieve fast parallel acceleration. After each partition, only peaks are stored and the effective memory footprint of the pipeline is small.

7.4.3 Finding correlation local maxima

The computation of the peaks of $C(u, v)$ is distributed among threads, similarly to loop 2 (see figure 7.9c) where each pixel of $C(u, v)$ is assigned to a thread. Threads consider their assigned pixel location a peak if the correlation value is strictly higher than values of the 8-nearest neighbors. This computation is very efficient on GPUs and the access is coalesced and often cached.

7.4.4 Finding the highest peak

As the peaks are detected, their location and correlation scores are analyzed. The scores of the peaks are accessed in parallel by threads and their maximum is computed in parallel by atomic operations (discussed in section 7.1.3). The majority of atomic operations in modern GPUs are currently limited to support only integer values. Thus, the floating point correlation coefficient is first encoded as a signed integer prior to applying the atomic maximum operation.

7.4.5 Pruning the peaks

The peaks that satisfy the quality criterion of having a score higher than a fraction of the highest peak score, as defined in equation (4.13), are preserved and the others are discarded. This operation is also done in parallel by threads of a block.

7.4.6 Refining peak locations

Stored peak locations are refined to subpixel accuracy by a local biquadratic function fit defined in equation (4.16). Each peak is handled by a thread and this operation is fast since only a few peaks exist in practice. Refined peaks are more accurate potential correspondences to the associated interest points.

7.4.7 Storing the correlation peaks

Only the peaks and their peak correlation values are of interest for the feature matching application because peaks are the putative locations for true feature correspondences. Hence, the correlation window may be discarded after the peaks are detected and refined. In practice, there are peaks only at a few peak locations in $C(u, v)$, but the number of peaks is unknown and in theory can require a large allocation space in the same order of $C(u, v)$. Allocating space for an upper bound is not practical since the size of $C(u, v)$ can be huge as described in section 7.4.2. The peaks are then pruned and counted first, as these operation are fast, then space for counted peaks is allocated in GPU memory, and, in a second pass through $C(u, v)$, the peaks are stored in the designated allocated arrays.

This concludes the GPU algorithm implementation discussion for the feature matching application of chapter 4, since the corner detection method (section 4.2) and the disambiguation steps (section 4.4) are performed on CPU. Porting the CPU portions of the algorithm to GPU will be addressed in future work.

7.5 Multiple view stereo application

Chapter 6 presents a multiple view stereo pipeline that returns reconstructed 3-d points and is suitable for real-time GPU implementation. The entire pipeline is implemented on a GPU and implementation details are described in this section. The GPU receives as inputs a set of $M + 1$ images, their estimated cameras and a parameter vector that controls the algorithm, and returns estimated depths. A depth map is computed for a reference image based on its M neighboring views. These images are transferred from the host to the GPU along with their estimated cameras and parameters that control the reconstruction.

7.5.1 Image rectification

Rectification is an operation that warps an image pair such that corresponding points do not have any vertical disparity. As cameras are estimated, calibrated rectification is performed by finding two transformations that sends the epipoles to infinity by mapping the images planes into a common plane, as discussed in section 3.1 (see [53] for more details). Rectification is reduced to two image transformations, which are simply 2-d interpolations.

Image interpolation is very efficient in GPUs and a thread is responsible for computing one interpolated pixel of the rectified image via bilinear interpolation. Adjacent threads process adjacent pixels for coalesced access.

7.5.2 Weighted Normalized cross-correlation

Section 6.2 introduces a set of rays, the ray grid, emanating from the reference view camera center along which depths are to be estimated via a photo-consistency function between reference pixel locations and epipolar lines associated to the rays, as discussed in section 6.4.1. Weighted NCC

is adopted as the photo-consistency measure and is denoted S_j in section 6.4.1. Weighted NCC is carried out exactly as the parallel implementation proposed in section 7.3, therefore computed via equation (7.10) where the weight function is given in section 6.4.1. Each template is located at the reference pixel location and its search window is a thin strip centered at its epipolar line. The strip is a rectangle since the images are rectified. The resulting correlation window is chosen to be one pixel thick (1-d) and to represent a similarity function between the template and its epipolar line. If the grid has N_r rays, and each ray defines M epipolar lines, one in each of M neighboring views, there is a total of $N_r \times M = T_c$ correlation arrays to be computed and these array computations are distributed among T_c thread blocks.

For a typical multiple view stereo application, T_c is in the order of hundreds of thousands to millions of arrays and each array has hundreds or thousands of correlation coefficients. This amount of computation is suitable to achieve remarkable accelerations on a GPU via a massive number of concurrent tasks, each one computing intensive operations.

7.5.3 Peak detection

Peaks of the correlation function are the potential corresponding locations along epipolar lines that define the depth along a ray. The peaks are counted and stored in the same way as it is done in sections 7.4.3 and 7.4.7 for the feature matching application. In addition, the location is similarly refined as proposed in section 7.4.6, however, since the correlation array here is 1-d, the refinement is achieved by fitting a quadratic function to the peak (section 6.4.7), as opposed to a biquadratic function, as performed in section 7.4.6.

7.5.4 Evaluating scores at each depth

The continuous function \mathcal{C}_j^w from equation (6.1) is a peak enhancement function derived from peaks of S_j , which is computed as described in section 7.5.3. \mathcal{C}_j^w sums Gaussian kernels centered at the peaks of S_j . Both \mathcal{C}_j^w and S_j are in image space and in order to register the scores from different views to common depths, the continuous \mathcal{C}_j^w is sampled at a range of ray depths according to equation (6.2) in section 6.4.3. The depths are uniformly sampled along the rays.

Each depth has a projection onto the epipolar line where the continuous \mathcal{C}_j^w is evaluated. Note, evaluating a function that is a sum of Gaussians at some coordinate, requires evaluating the sum of Gaussians at every coordinate. However, these computations are independent, there is one Gaussian per peak and many depths, hence there is a massive amount of computation suitable for parallel processing.

Each thread block is responsible for one ray and each thread analyzes one depth along this ray by computing its 3-d point, projecting it into neighboring views and evaluating the sum of Gaussians associated with the projected coordinates to arrive at Q_j from equation (6.2). Q_j is registered to depths along rays and is no longer viewpoint dependent as \mathcal{C}_j^w .

7.5.5 Score fusion

Once similarity scores Q_j , from multiple viewpoints indexed by j , are registered according to depths along rays, fusing them is based on robustly averaging multiple view scores of a particular depth, as in equation (6.5). The averaging is in accordance to quality criteria of equations (6.3) and (6.4), where only reliable scores are averaged. Continuing as in section 7.5.4, each thread independently computes the average of each depth p , arriving at the fused score $\mathcal{Q}(p)$. Still in the same thread block, threads stop and synchronize at a synchronization barrier after the averaging is complete. After the barrier, each thread checks a location of the fused score \mathcal{Q} to see it is a valid peak as described in section 6.4.6. The true depth of the surface along a ray is regularly associated to the highest peak of \mathcal{Q} . The highest valid peak is found by an atomic maximum operation to finally determine the depth along the ray according to equation (6.6) as the depth q associated to this highest peak.

7.5.6 Depth refinement

Refinement of an estimated depth q is achieved by the same parabolic interpolation of peaks performed in section 7.5.3. As only the highest peak is refined, only one thread performs such quick refinement instructions. This concludes the GPU algorithm implementation discussion for the multiple view stereo application. Note, section 6.6 regarding “volume far sides” proposes very simple and fast computations that are performed on CPU.

Chapter 8

Experiments and Evaluation

This chapter contains comprehensive experiments evaluating the performance of the proposed normalized cross-correlation (NCC) methods. First, the performance of the GPU implementation of the proposed parallel NCC algorithm is presented (section 8.1), followed by the applications. Section 8.2 discusses NCC for camera pose estimation. Section 8.3 describes the results in wide-baseline matching of dense repeating features and planar lattice tracking. Experiments on multiple view stereo and surface reconstruction are provided in section 8.4. Finally, section 8.5 concludes this chapter.

Many different datasets are used in the evaluations of this chapter. A notation is used to indicate the number of learning images of a dataset by appending the number to the dataset name. For instance, the dataset “capitol26” has 26 images and “downtown46” has 46.

8.1 Parallel NCC Performance

The proposed spatial domain parallel implementation of NCC is compared with state-of-the-art implementations that run on a CPU or a GPU. The most relevant part of the evaluation is on spatial domain methods for reasonably small templates, as the proposed method is geared towards these conditions. However, some results for alternative frequency domain methods and large templates are also shown. The use of small templates with sizes varying from 5×5 to 11×11 is appropriate

for the image matching applications of this thesis to avoid the bias that larger templates introduce under significant perspective distortions (see section 3.2.2). The experiments show that the proposed method is remarkably faster for small templates. The complexity of the spatial method increases faster with template size compared with the spectral method, as theoretically analyzed in figure 7.6. Therefore, the frequency domain is preferred for applications that require large templates.

This section demonstrates that the proposed implementation achieves remarkable speedup with respect to other efficient well-supported CPU and GPU methods. Comparisons of runtime between several combinations of square template sizes and square search window sizes are shown.

8.1.1 Alternative methods

Five alternative implementations of NCC that use the optimizations of [69] to compute the denominator term efficiently using sum-tables (see section 7.2.2) are presented:

- an efficient CPU spatial C-implementation from the Jet Propulsion Laboratory (JPL),
- the Matlab CPU spatial/spectral method provided by the function *normxcorr2*,
- the OpenCV CPU frequency domain FFT-based method,
- the OpenCV GPU spatial domain parallel method, and
- the OpenCV GPU frequency domain parallel FFT-based method.

The method from JPL is entirely written in C. JPL is a federally funded research and development center managed for NASA by the California Institute of Technology (Caltech). The Matlab code spatial method uses efficient built-in convolution computations written in C/C++ to compute the numerator of NCC, and vectorized Matlab language code to compute the denominator. Matlab and the Matlab code are developed by MathWorks. The OpenCV library [114] correlation functions are written in C/C++/CUDA and the version used in the experiments is 2.4.2. The CPU and GPU implementations of OpenCV are provided by functions called *matchTemplate* using parameter `CV_TM_CCOEFF_NORMED`. OpenCV has a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms that are open source, support multiple platforms and are extensively used in the field by well-established companies, startups, research groups and by governmental bodies.

By default, the OpenCV library and Matlab use hybrid domain implementations that decide during runtime whether to use a spatial method or a spectral one based mainly on template area. The experiments with OpenCV run on modified versions that impose one domain method or the other. As for Matlab, the original hybrid domain function was left unchanged. The switch point from spatial to spectral methods are at templates of size approximately between 16×16 to 25×25 .

8.1.2 Devices used in the experiments

Three devices are designated for NCC performance evaluations:

- the 4-core Intel Core **i7-950** CPU 3.07GHz with 12GB RAM, which performs at roughly 50 gigaFLOPS¹ for single precision computations,
- the Nvidia **GTX 275** GPU with 1792 MB of global memory and 240 CUDA cores that performs at 1010.88 gigaFLOPS (single precision), and
- the AMD Radeon **HD 7970** GHz Edition GPU with 3072 MB of global memory and 2048 stream cores that performs at 17.2 teraFLOPS (single precision).

All evaluations run on desktop PC with 64-bit OS equipped with the designate devices. Note, OpenCV GPU code is written in CUDA language and can only run on Nvidia devices. Thus, the OpenCV GPU experiments are limited to the GTX 275 device.

8.1.3 Runtime of NCC on designated devices

The runtime of specified NCC methods and the associated speedup of the proposed implementation is given in figures 8.1 to 8.12. Figures 8.1 to 8.3 show running times of specified CPU normalized cross-correlation methods and figures 8.4 to 8.7 provides the ones of the GPU methods. The speedup of the proposed GPU algorithm over others are plotted in figures 8.8 to 8.12. Each figure has an accompanying table with numerical values from the associated figure.

The reported times are for computation only and do not include memory transfers. For the applications of this thesis and many others, memory transfers of input images or data can overlap with GPU kernel computations and therefore be hidden. In addition, output correlation values can

¹FLOPS is an abbreviation of FLoating-point Operations Per Second.

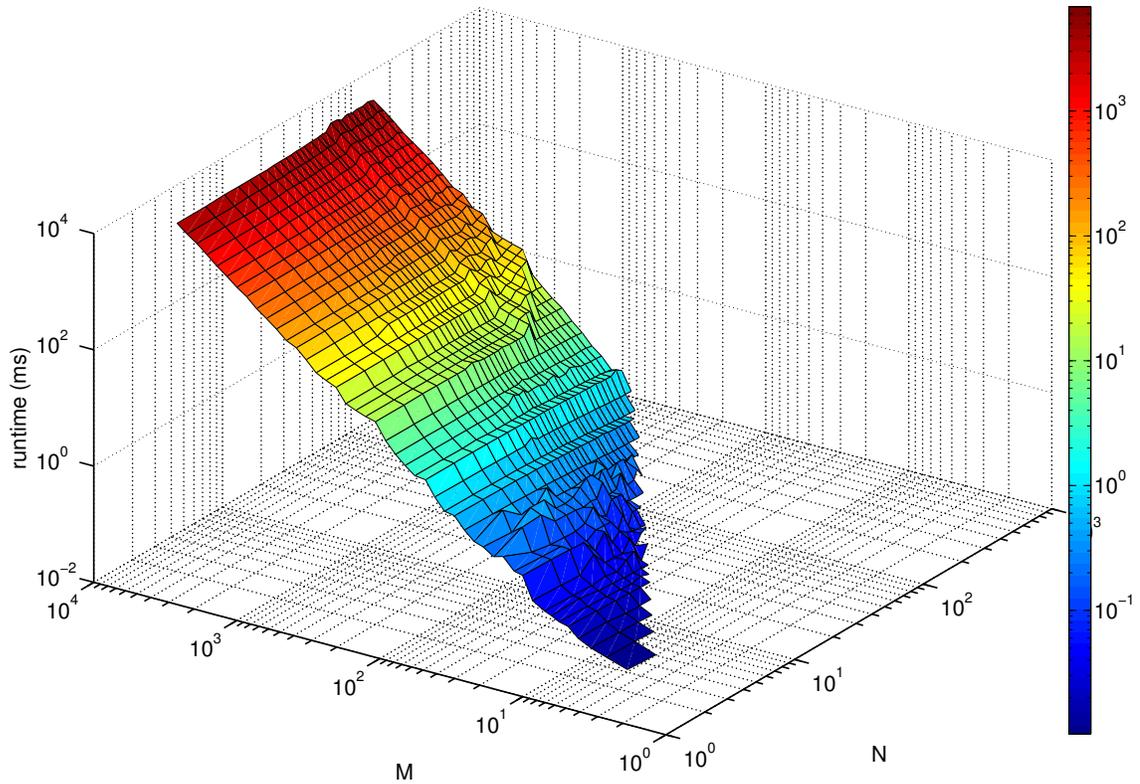


Figure 8.1: Runtime (ms) of frequency domain NCC implementation of **OpenCV** running on the “i7-950” CPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	0.524	1.01	8.26	28.1	114	852	2412
	5×5	0.512	0.998	8.19	28	113	851	2418
	7×7	0.502	0.982	8.11	28	114	853	2378
	11×11	0.479	0.961	8	27.8	114	855	2370
	13×13	0.474	0.951	7.96	27.8	115	856	2367
	15×15	0.465	0.929	7.93	28	114	908	2462
	25×25	0.424	0.88	7.91	27.7	114	912	2475

Table 8.1: Running times (ms) taken from figure 8.1.

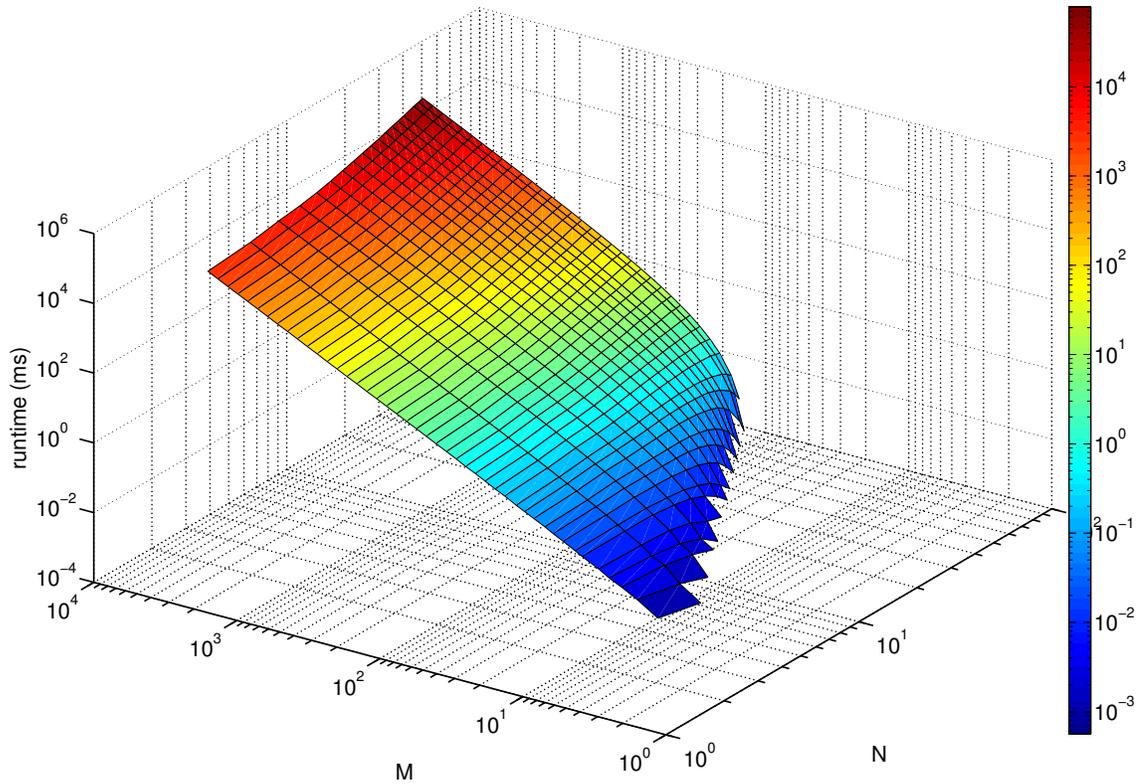


Figure 8.2: Runtime (ms) of spatial domain NCC implementation of **JPL** running on the “i7-950” CPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	0.324	0.679	4.35	17.5	72.1	633	1747
	5×5	0.404	0.852	5.55	22.6	95	824	2283
	7×7	0.52	1.12	7.5	30.8	130	1134	3139
	11×11	0.685	1.54	10.8	45.3	194	1689	4677
	13×13	0.796	1.85	13.3	56.3	240	2108	5855
	15×15	0.935	2.21	16.5	70	301	2643	7357
	25×25	1.51	4.11	35.8	160	702	6233	17358

Table 8.2: Running times (ms) taken from figure 8.2.

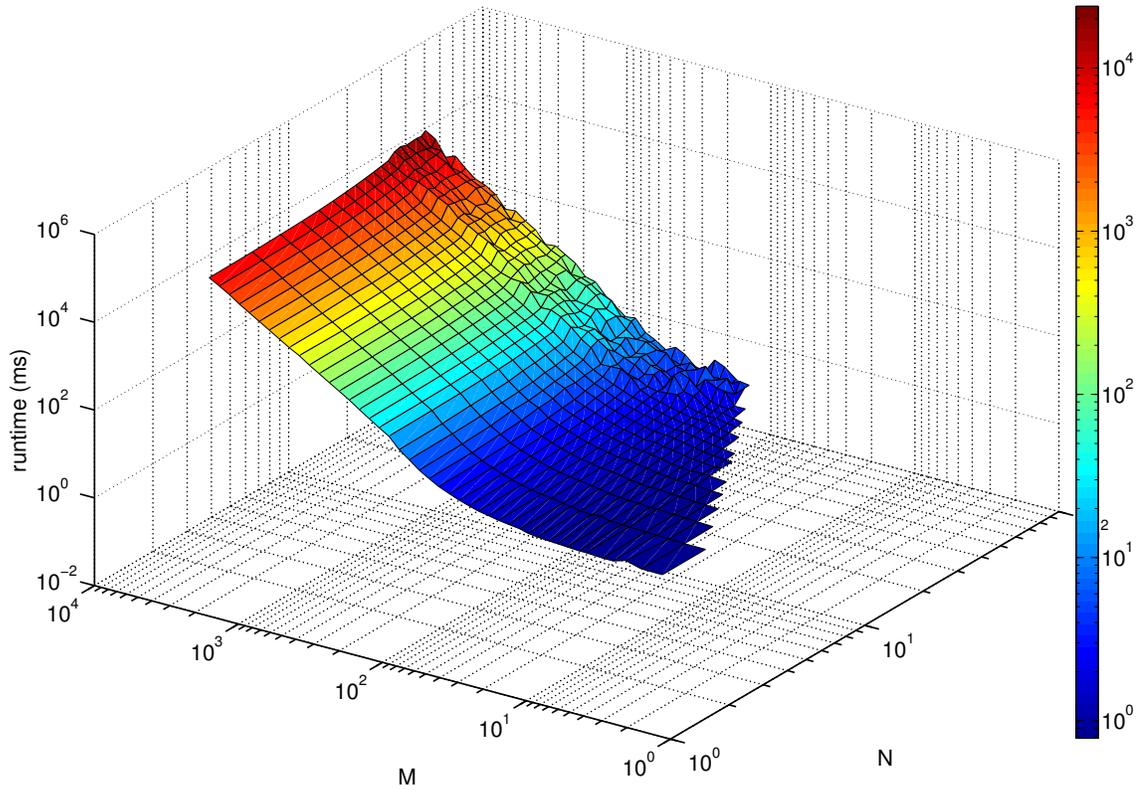


Figure 8.3: Runtime (ms) of hybrid domain NCC implementation of **Matlab** running on the “i7-950” CPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	1.4	1.79	6.75	39.6	177	1513	4312
	5×5	1.45	1.86	7.04	40.6	180	1538	4409
	7×7	1.52	1.96	7.61	41.9	184	1573	4623
	11×11	1.7	2.23	8.77	45.8	198	1706	5060
	13×13	1.82	2.41	9.58	48.2	208	1806	5382
	15×15	1.96	2.6	10.8	50.7	216	1905	5670
	25×25	4.14	3.88	24.9	104	415	4201	8483

Table 8.3: Running times (ms) taken from figure 8.3.

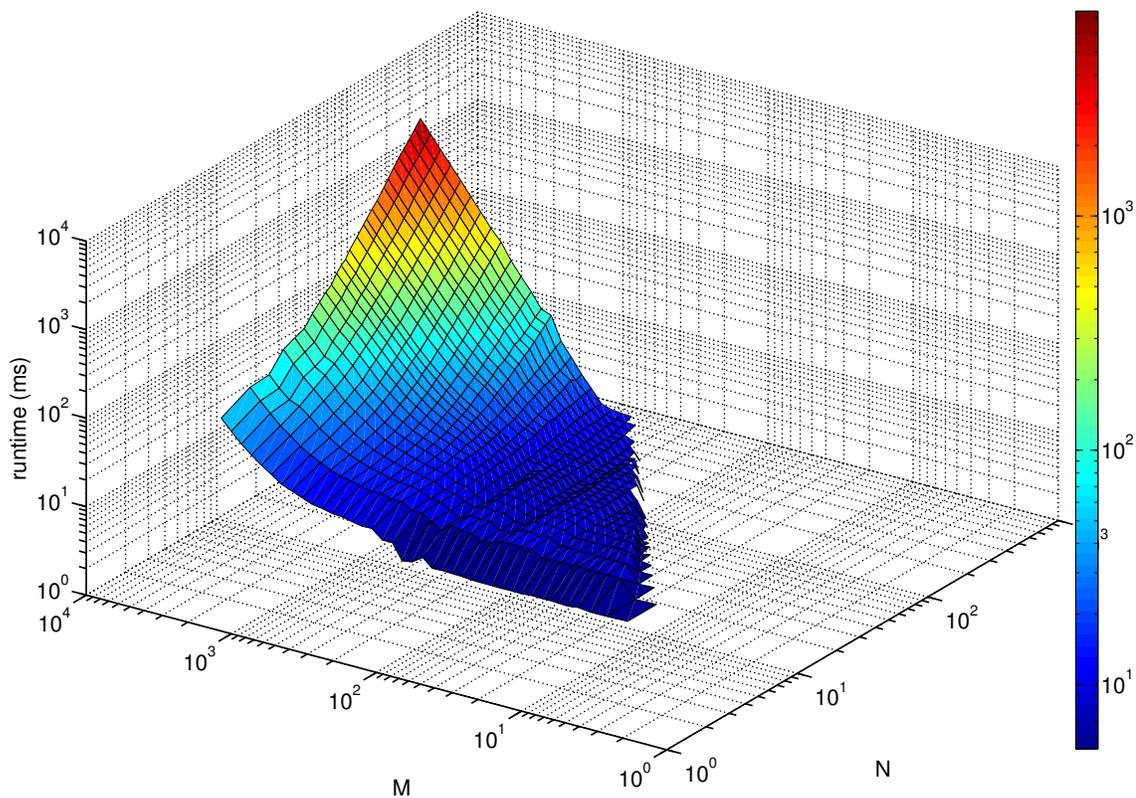


Figure 8.4: Runtime (ms) of spatial domain NCC implementation of **OpenCV** running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	5.51	5.6	8.44	11.5	15.7	58.9	N/A
	5×5	8.54	9.59	11.6	14.7	21	84	N/A
	7×7	8.34	9.65	11.7	15.4	23.5	87.3	N/A
	11×11	8.3	9.53	12.2	17.3	30.9	153	N/A
	13×13	8.45	9.69	12.4	18.6	34.5	170	N/A
	15×15	8.69	9.68	12.8	19.7	39.1	220	N/A
	25×25	8.57	10.1	14.7	27.6	71.7	497	N/A

Table 8.4: Running times (ms) taken from figure 8.4.

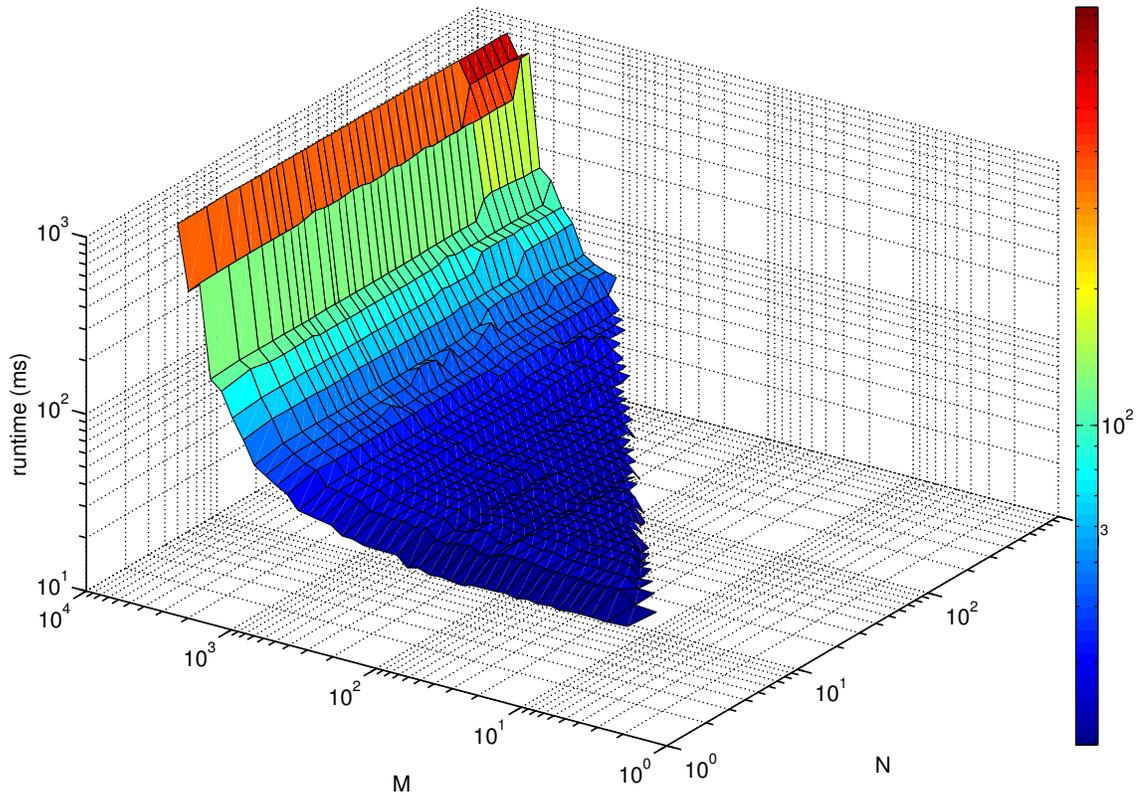


Figure 8.5: Runtime (ms) of frequency domain NCC implementation of **OpenCV** running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	19.9	20.5	23.6	26	32.7	99.7	373
	5×5	23.2	24.7	26.5	28.6	35.9	106	375
	7×7	23.3	25.2	26.1	28.8	35.8	102	376
	11×11	23.5	24.9	26.6	28.8	35.8	101	375
	13×13	23.5	24.8	26.4	28.7	35.8	102	374
	15×15	24.1	24.7	26.3	28.8	35.8	102	376
	25×25	23.5	24.9	26.6	28.9	35.9	99.3	381

Table 8.5: Running times (ms) taken from figure 8.5.

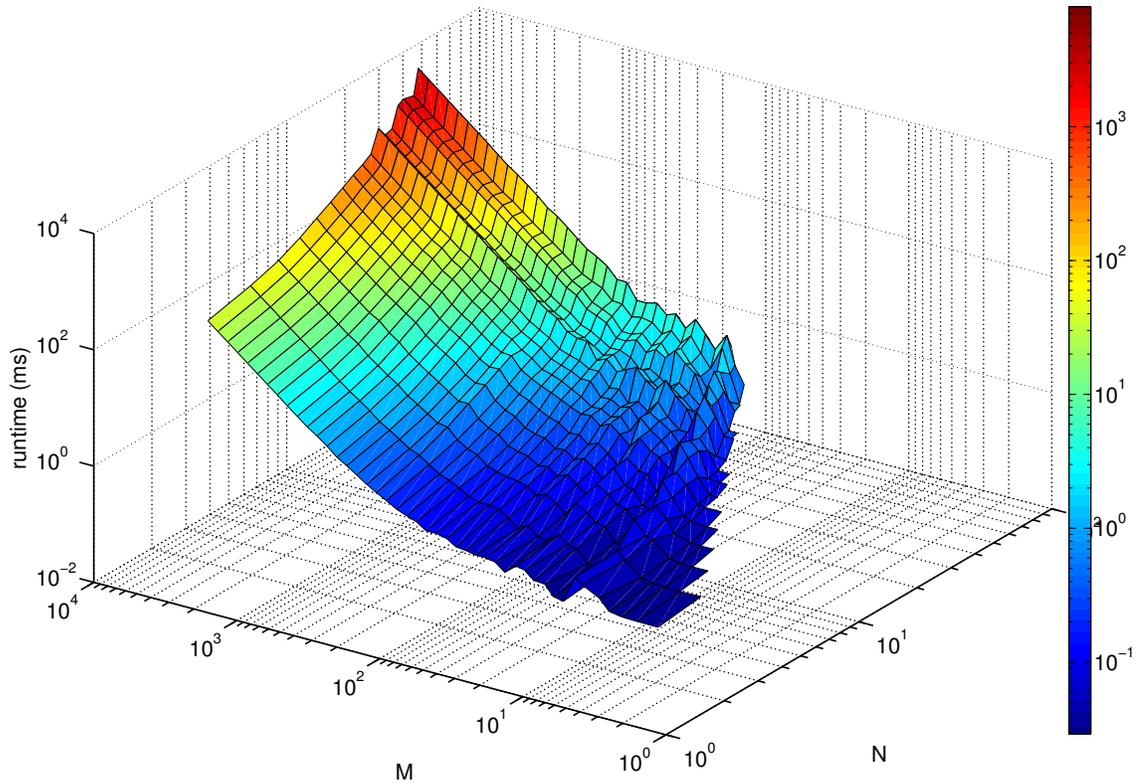


Figure 8.6: Runtime (ms) of **proposed** spatial domain NCC implementation running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	0.0697	0.0762	0.151	0.342	1.14	8.84	24.2
	5×5	0.0728	0.0868	0.217	0.538	1.71	13.1	35.7
	7×7	0.0919	0.152	0.275	0.818	2.49	18.4	50.3
	11×11	0.148	0.259	0.514	1.73	5.02	38.2	103
	13×13	0.193	0.342	0.748	2.36	6.86	52.6	142
	15×15	0.304	0.553	1.1	3.1	9.18	69	186
	25×25	1.06	1.27	2.62	7.36	28	222	609

Table 8.6: Running times (ms) taken from figure 8.6.

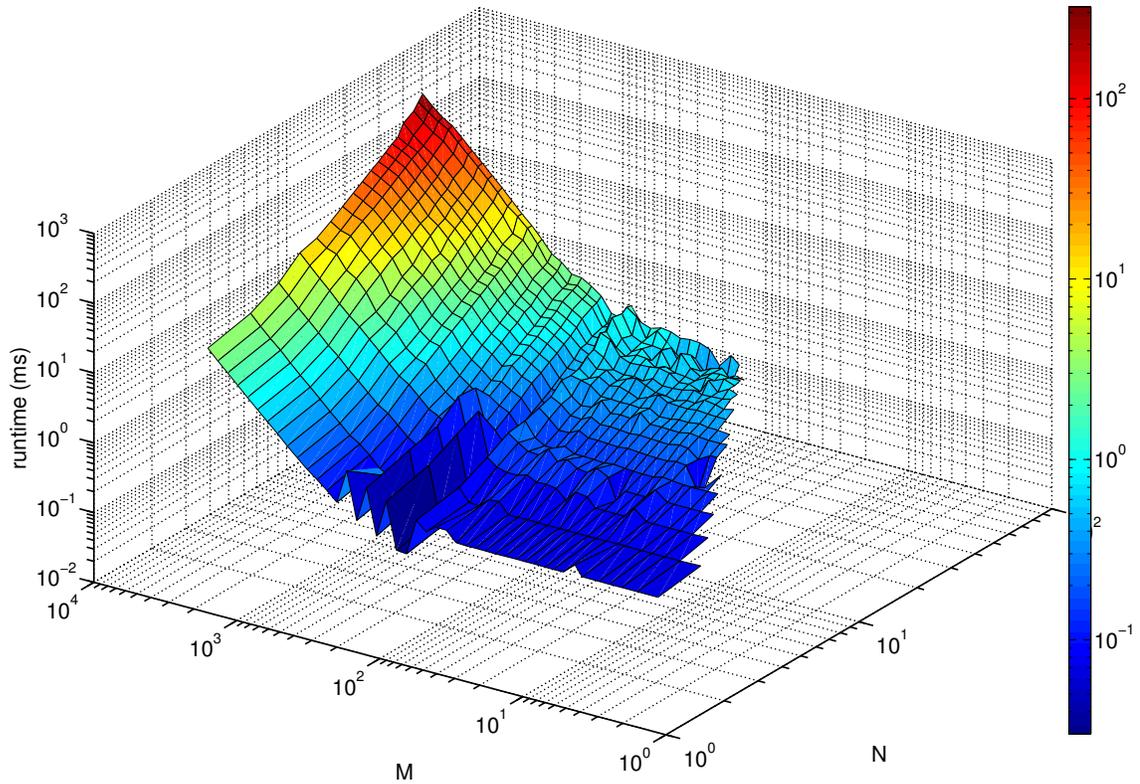


Figure 8.7: Runtime (ms) of **proposed** spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	0.0657	0.0663	0.0911	0.105	0.119	0.861	2.35
	5×5	0.0803	0.0801	0.117	0.159	0.167	1.17	3.2
	7×7	0.1	0.138	0.184	0.215	0.26	1.83	5.03
	11×11	0.16	0.166	0.297	0.39	0.562	4.7	14.3
	13×13	0.208	0.216	0.35	0.32	0.747	6.44	20
	15×15	0.256	0.269	0.404	0.41	0.98	8.83	26
	25×25	0.497	0.55	0.858	1.11	2.75	23.8	66.4

Table 8.7: Running times (ms) taken from figure 8.7.

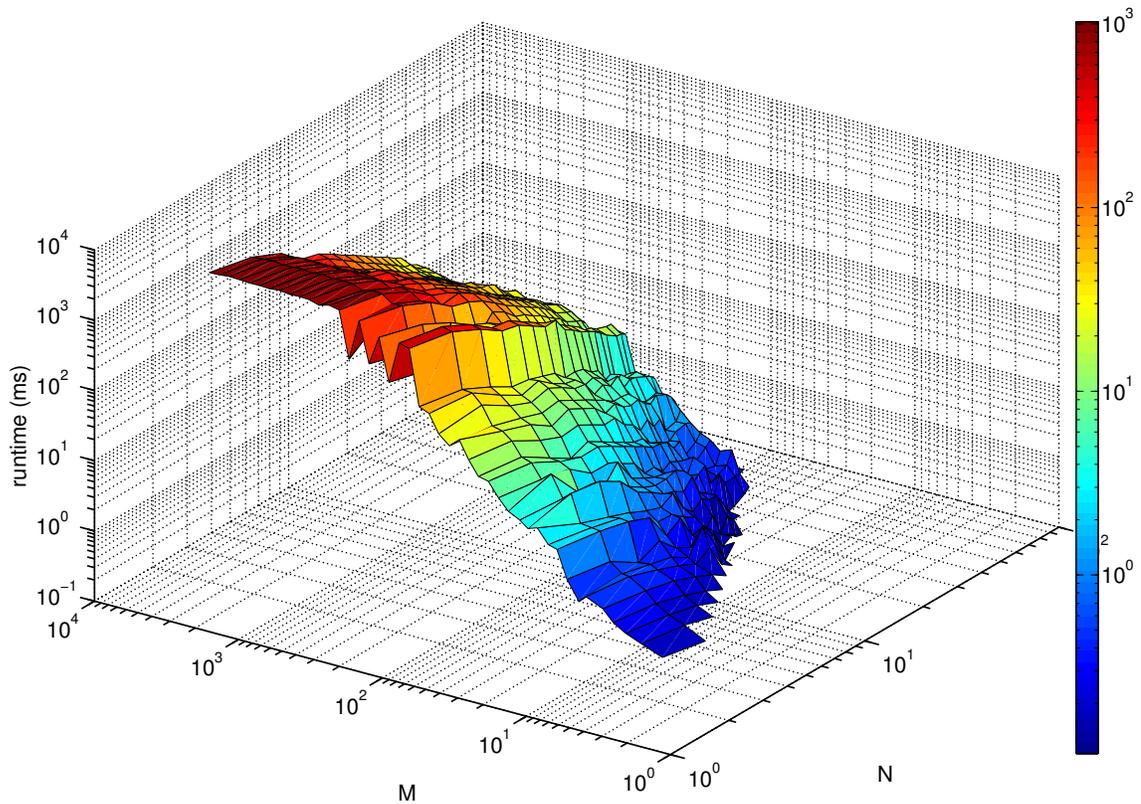


Figure 8.8: Speedup of **proposed** spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images over frequency domain NCC implementation of **OpenCV** running on the “i7-950” CPU.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	7.97	15.2	122	376	951	988	1025
	5×5	6.37	12.5	93	249	678	727	754
	7×7	4.99	7.14	58.5	163	436	466	473
	11×11	2.98	5.78	34.3	81.9	203	187	167
	13×13	2.28	4.41	27.5	85.5	153	138	119
	15×15	1.81	3.46	22.9	67	117	105	94.4
	25×25	0.855	1.64	10.1	24.5	41.4	38.5	37.3

Table 8.8: Speedup ratio values taken from figure 8.8.

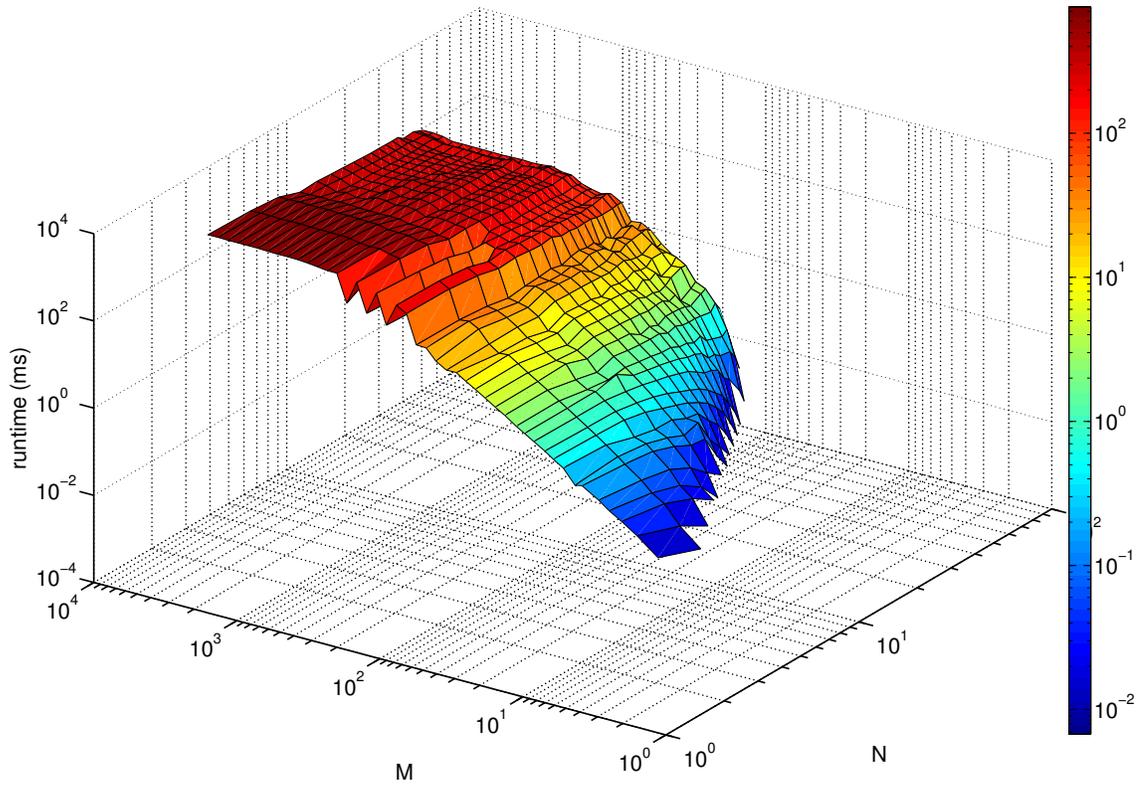


Figure 8.9: Speedup of **proposed** spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images over spatial domain NCC implementation of **JPL** running on the “i7-950” CPU.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	4.92	10.2	59.6	253	605	734	742
	5×5	5.03	10.6	58.6	216	569	704	712
	7×7	5.18	8.15	50.1	192	500	619	624
	11×11	4.27	9.25	43.4	141	345	369	329
	13×13	3.83	8.6	43.6	175	322	339	293
	15×15	3.65	8.22	45.3	170	307	306	282
	25×25	3.04	7.64	44.4	143	255	263	261

Table 8.9: Speedup ratio values taken from figure 8.9.

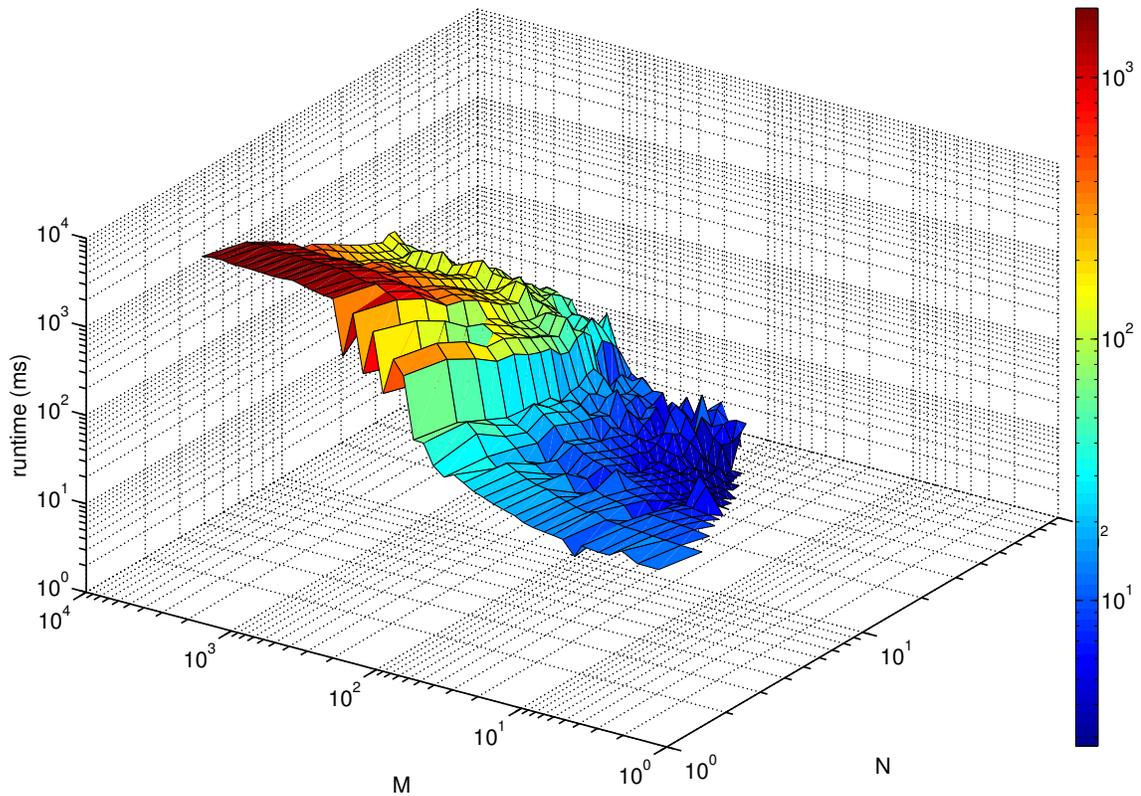


Figure 8.10: Speedup of **proposed** spatial domain NCC implementation running on the “HD 7970” GPU for $N \times N$ templates and $M \times M$ images over hybrid domain NCC implementation of **Matlab** running on the “i7-950” CPU.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	21.3	26.9	94.3	568	1484	1755	1830
	5×5	18	23.2	75.8	387	1079	1314	1374
	7×7	15.3	14.2	51.4	260	709	858	918
	11×11	10.6	13.4	35.2	142	352	372	356
	13×13	8.78	11.2	31.3	150	278	290	269
	15×15	7.69	9.69	29.6	123	221	221	217
	25×25	8.33	7.12	30.8	92.5	151	178	129

Table 8.10: Speedup ratio values taken from figure 8.10.

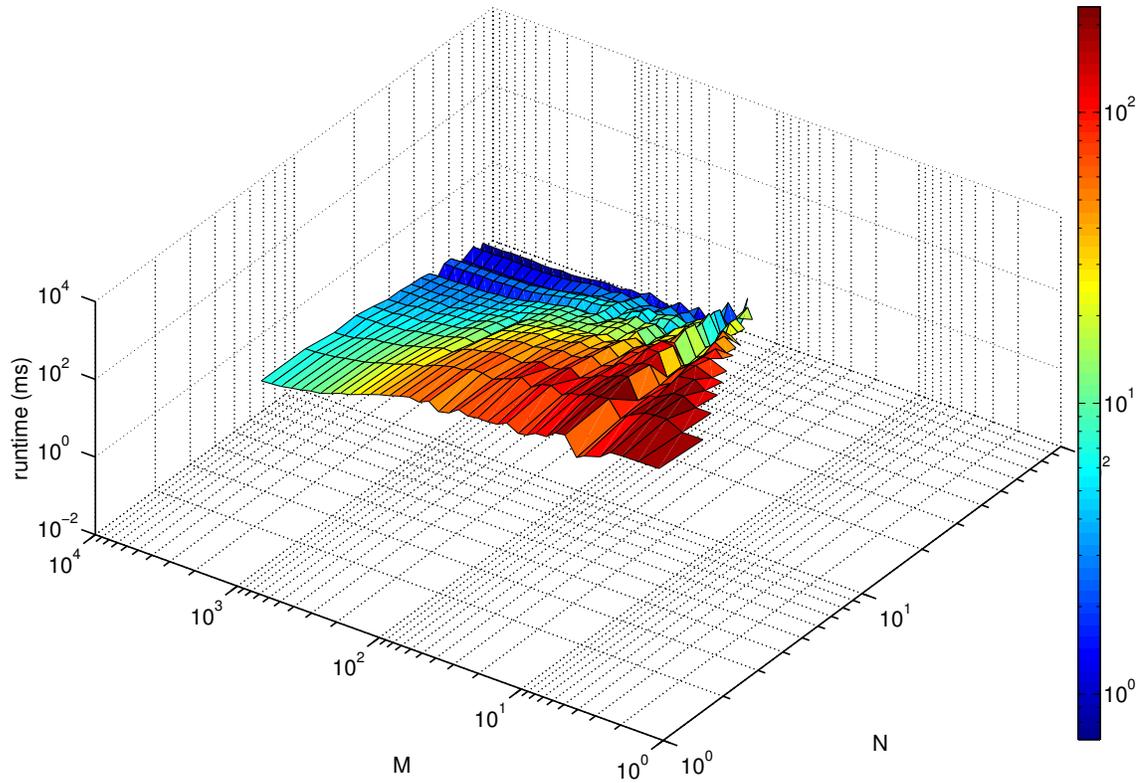


Figure 8.11: Speedup of **proposed** spatial domain NCC implementation running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images over spatial domain NCC implementation of **OpenCV** running on the “GTX 275” GPU.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	79.1	73.6	55.8	33.8	13.8	6.69	N/A
	5×5	118	113	53.7	27.5	12.3	6.44	N/A
	7×7	91.4	64	42.9	18.8	9.47	4.75	N/A
	11×11	56.8	36.9	23.8	10	6.17	4.02	N/A
	13×13	44.3	28.5	16.7	7.86	5.03	3.23	N/A
	15×15	28.9	17.6	11.7	6.33	4.26	3.2	N/A
	25×25	8.15	8.28	5.62	3.79	2.56	2.25	N/A

Table 8.11: Speedup ratio values taken from figure 8.11.

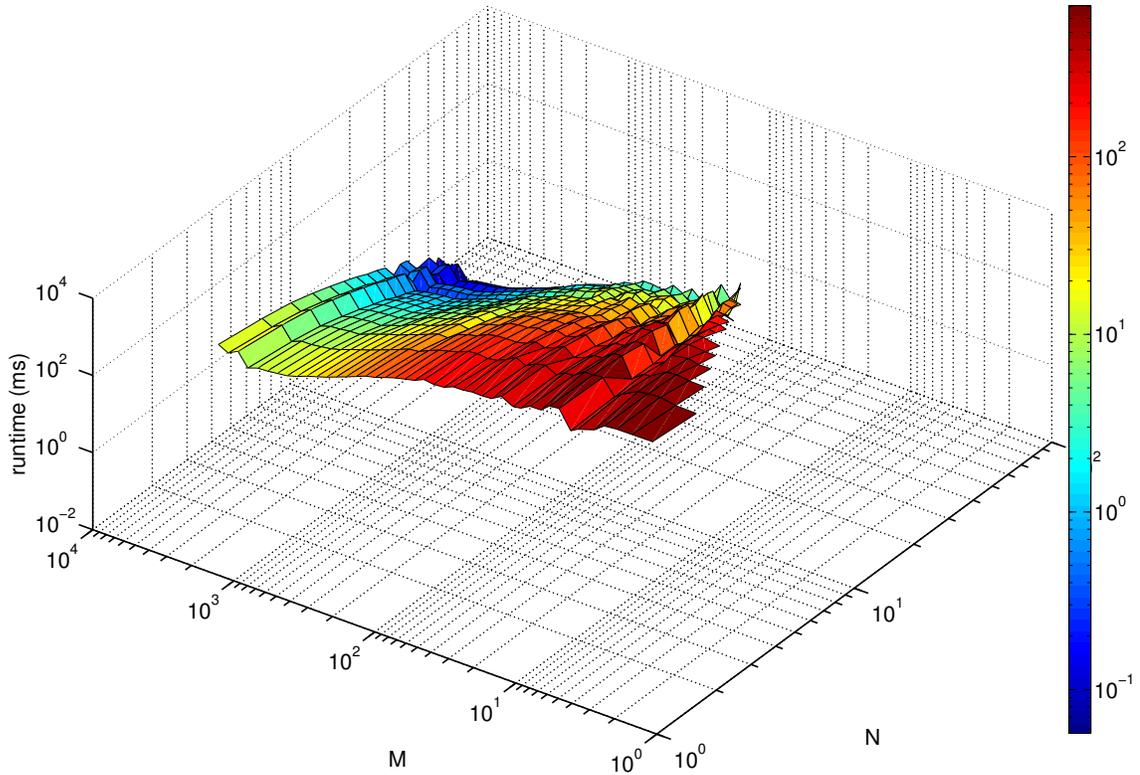


Figure 8.12: Speedup of **proposed** spatial domain NCC implementation running on the “GTX 275” GPU for $N \times N$ templates and $M \times M$ images over frequency domain NCC implementation of **OpenCV** running on the “GTX 275” GPU.

		Search window sizes						
		70×70	100×100	250×250	500×500	1024×1024	3000×3000	5000×5000
Template sizes	3×3	285	269	156	76.6	28.9	11.3	15.3
	5×5	321	289	123	53.4	21.1	8.07	10.4
	7×7	256	166	95.9	35.2	14.4	5.52	7.44
	11×11	161	95.9	52.3	16.7	7.16	2.66	3.61
	13×13	123	72.6	35.5	12.2	5.23	1.93	2.62
	15×15	80.3	44.7	24.1	9.28	3.91	1.48	2.01
	25×25	22.3	20.2	10.2	3.99	1.29	0.45	0.621

Table 8.12: Speedup ratio values taken from figure 8.12.

be processed in the GPU if the application pipeline is GPU-accelerated and do not require output transfers back to CPU. Thus, transfer times are not reported here since they are considered negligible, otherwise, they must be reported [50].

Experiment description. This experiment consists of distributing on the GPU the computation of NCC for an image of size $M \times M$ with a template of size $N \times N$ and measure the runtime. For a CPU, the method runs sequentially. Each experiment is repeated 10 times and the results are averaged to mitigate variabilities. Runtimes and speedup for a broad range of values for N and M are provided for information purposes. However, more relevance must be given to smaller templates (small N) and large images (large M) since spectral methods are preferred for large templates (discussed in section 7.2.2) and larger images require significant time to process on CPU, unlike tiny ones, which demand no GPU acceleration. Note, GPU speedup for small images is limited since there is little data to distribute among the GPU cores.

The applications of this thesis are analogous to the processing of this experiment with large M and small N since each of many thread blocks on the GPU process a fraction of the large $M \times M$ image, equivalent to NCC searches along thin strips around epipolar lines (chapters 4 and 6) where each of many blocks is in charge of a small portions of the image, *i.e.* the strip.

In summary, attention must be given essentially to performance speedup of the proposed method over others when using small templates and large images, *i.e.*, the upper right portion of the tables 8.8 to 8.12. Denote such image sizes of the tables as *target sizes*. Note, a “N/A” symbol is displayed on the tables if runtime is not available for the a device among reasons as lack of memory or lack of other computing resources to run the experiment.

8.1.4 Discussion

As seen on the target sizes of tables 8.8 to 8.10, the proposed method running on “HD 7970” achieves remarkable speed up over alternative CPU implementations in the range of approximately 200X to 1000X with respect to OpenCV, 330X to 750X with respect to JPL and 350X to 1800X with respect to Matlab. Similarly, comparing the performance of parallel GPU methods in tables 8.11 to 8.12, the proposed method outperforms the OpenCV equivalent spatial method by 4X to 13X and

outperforms the OpenCV spectral method by 2.6X to 29X, when all algorithms run for target sizes on the same “GTX 275” GPU device. The performance of the proposed method tends to decrease as the template size increases since the method is designed for small templates and does not exploit sum-tables. Nevertheless, at the target sizes performance gains are noticeable over OpenCV, one of the fastest publicly available NCC implementations.

8.1.5 Comparison to additional NCC methods

To put the results into perspective, other alternative methods have reported speedups for normalized cross-correlation of less than 10X [8] or on average 22X [23]. Wang and Wang [127] report computing NCC between a 512×512 image and a 80×80 template image using an FPGA in 224ms. The proposed approach is not designed for high performance on such high template sizes, yet achieves the same computation in 10.3ms on “HD 7970.”

Lu *et al.* [75] report a maximum speedup of 130X for the correlation of images of size roughly 750×1500 with shifted versions of themselves (*auto-correlation*). In this case, the template image is also very large being of the size of the actual image. The speedup was attained from CUDA-specific precise coding running on a single Nvidia Tesla C1060, which contains a GT200 GPU, over a C implementation running on a single Dell XPS 730 with an Intel Q6600 Core2 Quad at 2.4 GHz. As a comparison, the maximum reported speedup by the proposed method with respect to a C implementation is in the order of 750X, as shown in table 8.9, which is nearly 6X larger speedup than [75]. The results are comparable despite hardware differences since they were running experiments in older generations of both GPU and of CPU, therefore the performance ratio measured by the speedup is similar to when using newer generation hardware, as the ones in the proposed experiments.

Idzenga *et al.* [55] propose a CUDA implementation of NCC running on the Nvidia Tesla K20 (2496 CUDA cores), one of the latest graphics cards from Nvidia. Their speedup with respect to Matlab CPU implementation running on Intel Xeon CPU L5420 at 2.50 GHz is up to 376X, whereas the proposed speedup on similar hardware w.r.t. Matlab is up to 1830X (see table 8.10).

Matching method	MSE	Number of 3-d points	Number of 2-d matches
“capitol26”	0.349	26809	41690
“downtown46”	0.305	48963	76280
“campus29”	0.252	21671	29068
“horse21”	0.184	10538	14620

Table 8.13: Mean squared reprojection error (MSE), number of correspondences and number of 3-d points estimated via structure from motion using SIFT matching.

Matching method	MSE	Number of 3-d points	Number of 2-d matches	MSE ratio w.r.t. SIFT
“capitol26”	0.131	54013	94416	0.38
“downtown46”	0.161	134686	214026	0.53
“campus29”	0.176	64316	88275	0.69
“horse21”	0.172	24352	34377	0.93

Table 8.14: Mean squared reprojection error (MSE), number of correspondences and number of 3-d points estimated via structure from motion using proposed feature matching method (chapter 4).

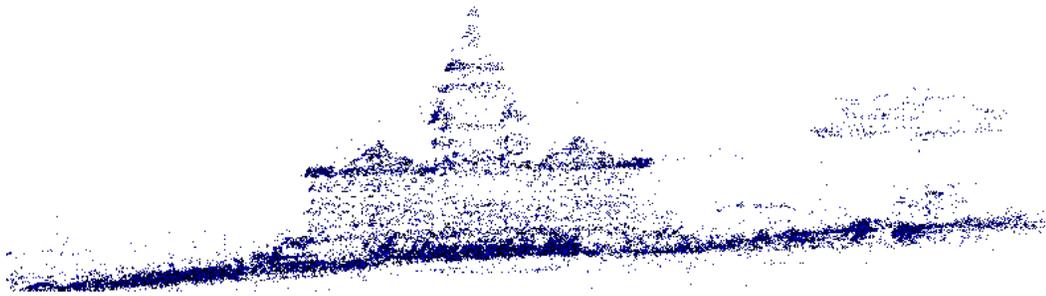
8.2 NCC for camera pose estimation

This section presents results in the accuracy of structure from motion when using features matched via the proposed method of chapter 4. Accuracy is measured in two steps: the features are used for estimating camera models that are refined by bundle adjustment nonlinear optimization (see section 3.4.1) and then, given the cameras, the mean reprojection error MSE of the matching features, which is defined in equation (3.15), is used as a measure of camera estimation accuracy.

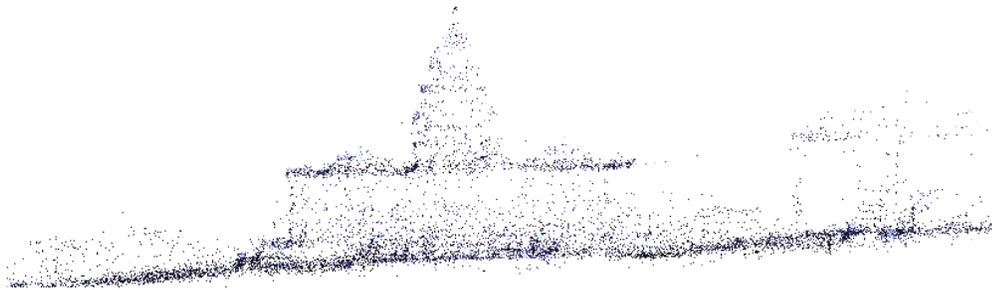
Tables 8.13 and 8.14 show the quantitative results demonstrating that the proposed method achieves better accuracy than SIFT matching in the four tested datasets. The tables present the error measure, the total number of correspondences and total number of 3-d points recovered from structure from motion. Table 8.14 also show the ratio between the estimation error for the evaluated methods. Figures 8.13 to 8.15 illustrate the reconstructed structure, which, besides being more accurate, is visually denser when estimated using the proposed method because it recovers more points. A qualitative comparison of matches found via proposed NCC method and via traditional SIFT matching is provided in figures 8.16 and 8.17 to illustrate that the proposed method finds more matches, and only the proposed approach handles repetition.



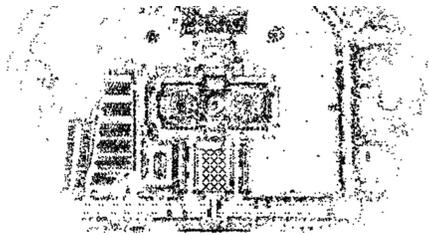
(a)



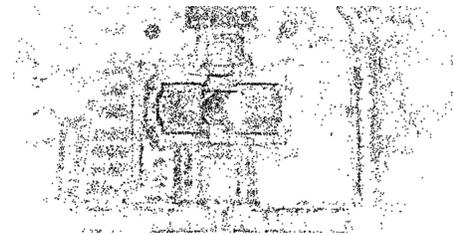
(b)



(c)



(d)

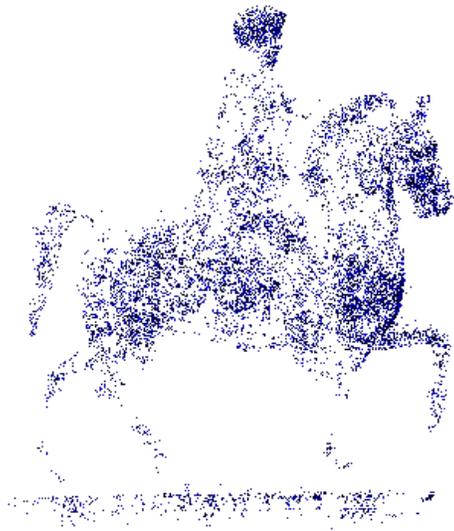


(e)

Figure 8.13: Structure from motion reconstruction of the “capitol26” dataset. (a) Dataset images. (b,d) Reconstructed 3-d points using the proposed matching method (denser) and (c,e) using SIFT.



(a)



(b)



(c)

Figure 8.14: Structure from motion reconstruction of the “horse21” dataset. (a) Images of the dataset. (b) Reconstructed 3-d points using the proposed matching method (denser) and (c) using SIFT.

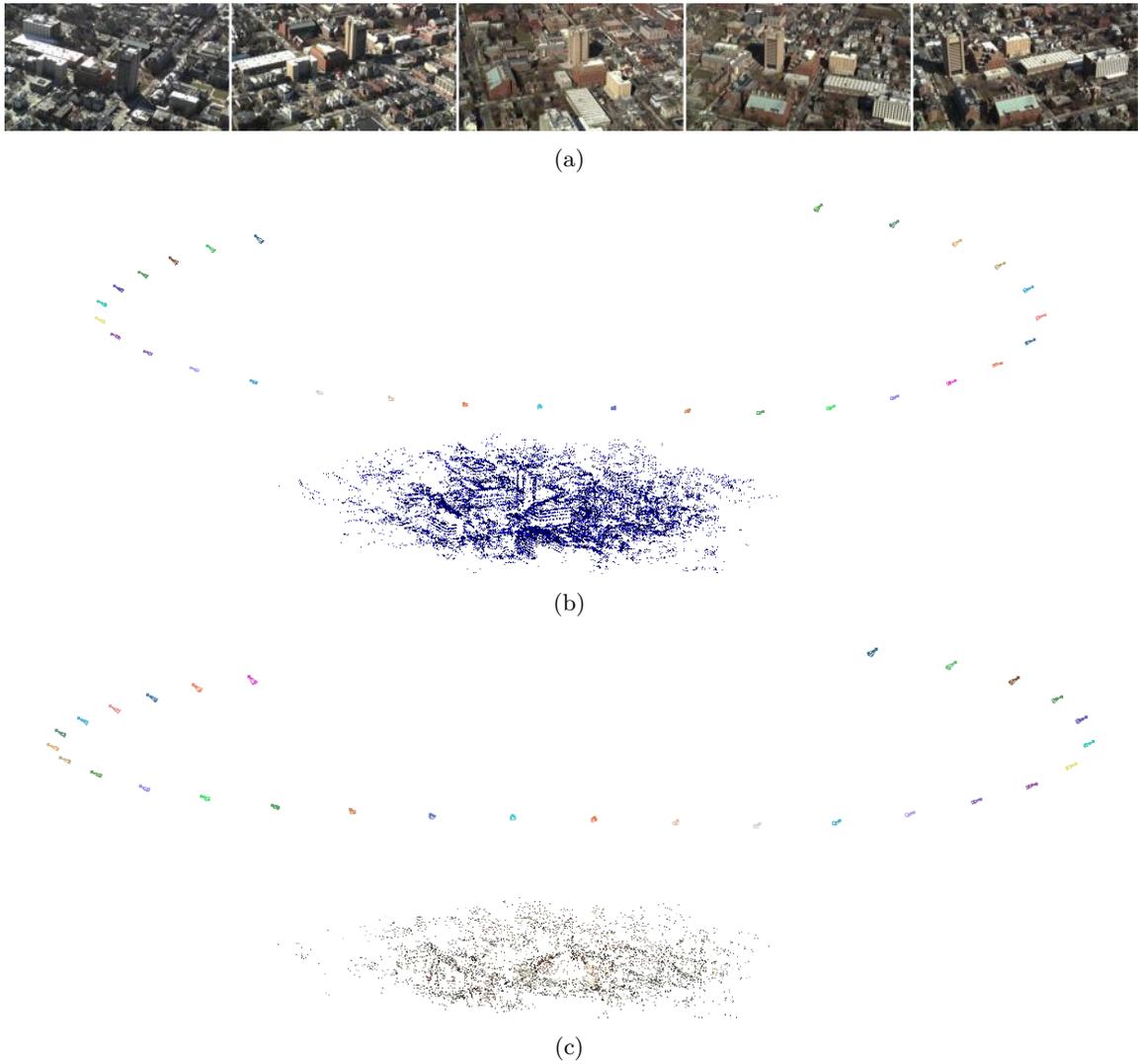


Figure 8.15: Structure from motion reconstruction of the “campus29” dataset. (a) Images of the dataset. (b) Reconstructed 3-d points using the proposed matching method (denser) and (c) using SIFT. Estimated cameras are shown above the scene.



Figure 8.16: Typical example of pairwise matching via the proposed NCC method. The estimation method handles repetition and matches are denser than the ones found via traditional SIFT matching (*c.f.* figure 8.17).



Figure 8.17: Typical example of pairwise matching via traditional SIFT method. The matching does not handle repetition and estimated matches are sparser than the ones found via proposed method (*c.f.* figure 8.16).



Figure 8.18: Images of the “downtown46” dataset.

8.3 Planar lattice matching and tracking

This section shows the improvement in accuracy of structure from motion when using features matched via the planar lattice model of chapter 5. Accuracy is measured in the same two steps as in section 8.2: the features are used for estimating camera models that are refined by bundle adjustment nonlinear optimization (see section 3.4.1) and then, given the cameras, the mean squared reprojection error MSE of the matching features, which is defined in equation (3.15), is used as a measure of camera estimation accuracy.

The lattice matching improves camera estimation accuracy in two ways: providing an additional dense set of reliable feature matches coming from building facades of urban scenes and filtering out incorrect matches that occasionally appear in such facades. For comparison, reprojection errors are computed and provided using matches from SIFT, from the proposed method of chapter 4, from the lattices tracked across multiple image frames and from proposed methods combined. The reprojection errors of all proposed methods are much smaller than the errors resulting from the same process using SIFT matching and often provides much more feature correspondences to perform bundle adjustment.

8.3.1 Results

The proposed matching method of chapter 4, denoted here by “NccD” finds correspondences among sparse interest points via normalized cross-correlation and performs disambiguation when necessary. NccD stands for *normalized cross-correlation* and *disambiguation*. The proposed matching method of chapter 5 is denoted here by “NccDRF” for *normalized cross-correlation on dense repeating features*. NccDRF complements NccD by finding correspondences for dense repetitive features distributed in

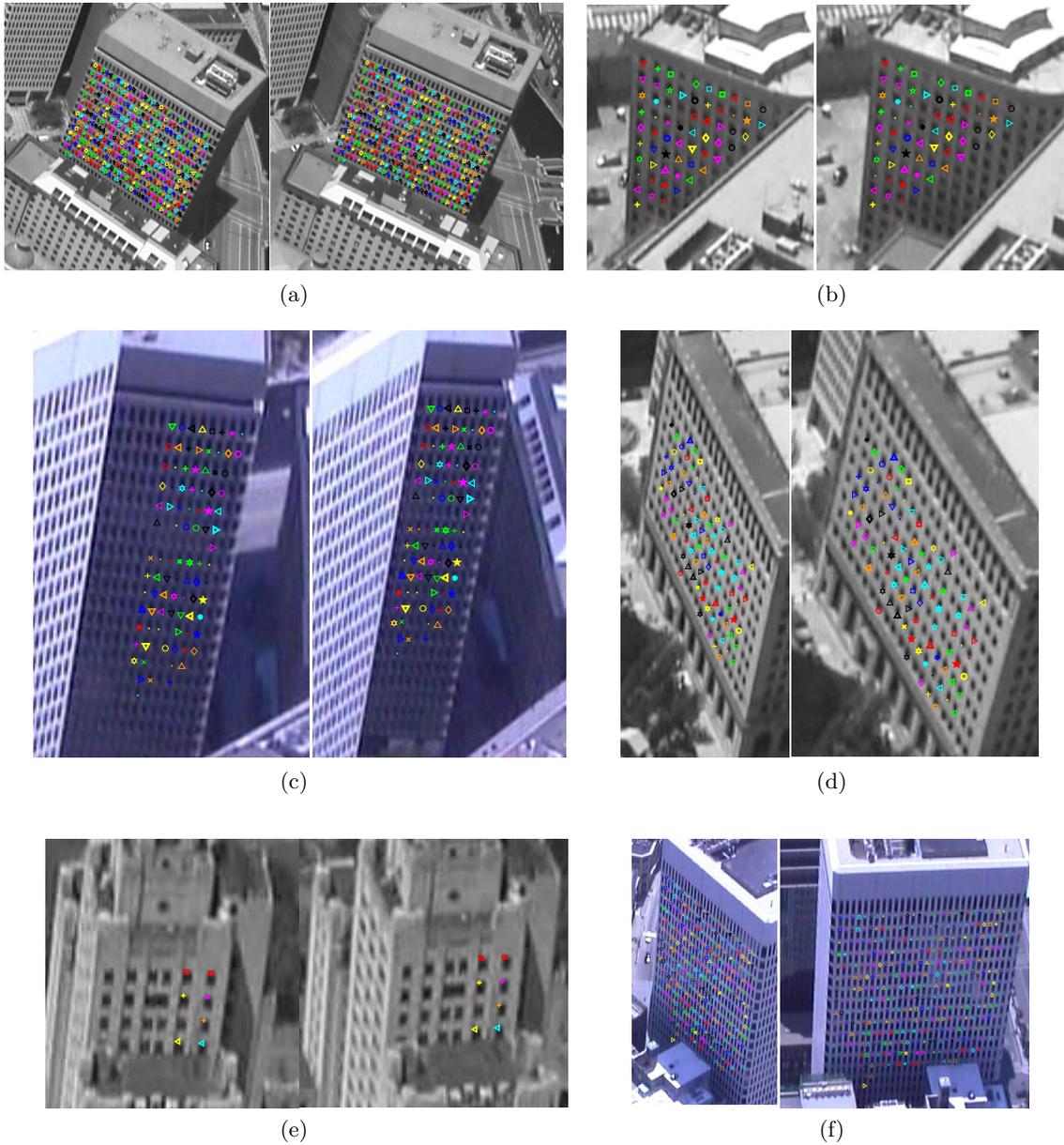


Figure 8.19: Results of proposed matching of dense repetitive features for adjacent frames. Short-baseline matching results show robustness to: (a) large grids of repeating elements, (b) partial occlusion of lattice in reference image, (c) partial occlusion of the lattice in the target image including several complete vertical columns of windows, (d) oblique views and (e) small lattices. A wide-baseline failure case due to a horizontal shift of the entire matching lattice by one window is shown in (f).

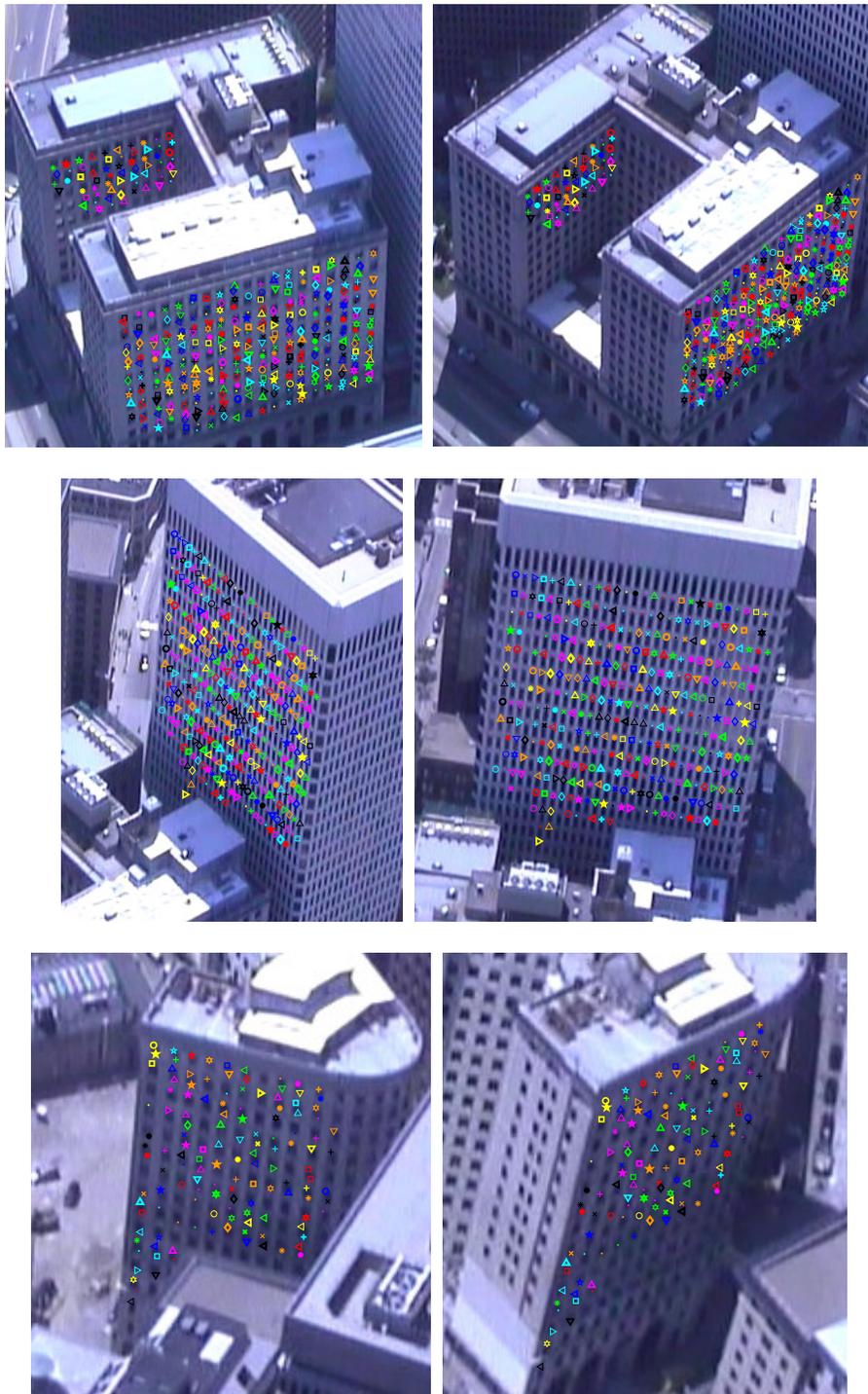


Figure 8.20: Results of the proposed wide-baseline matching of dense repetitive features. The top row shows robustness to occlusion and multiple planar surfaces with the same repeating elements. The other rows show wide-baseline matches for other buildings.

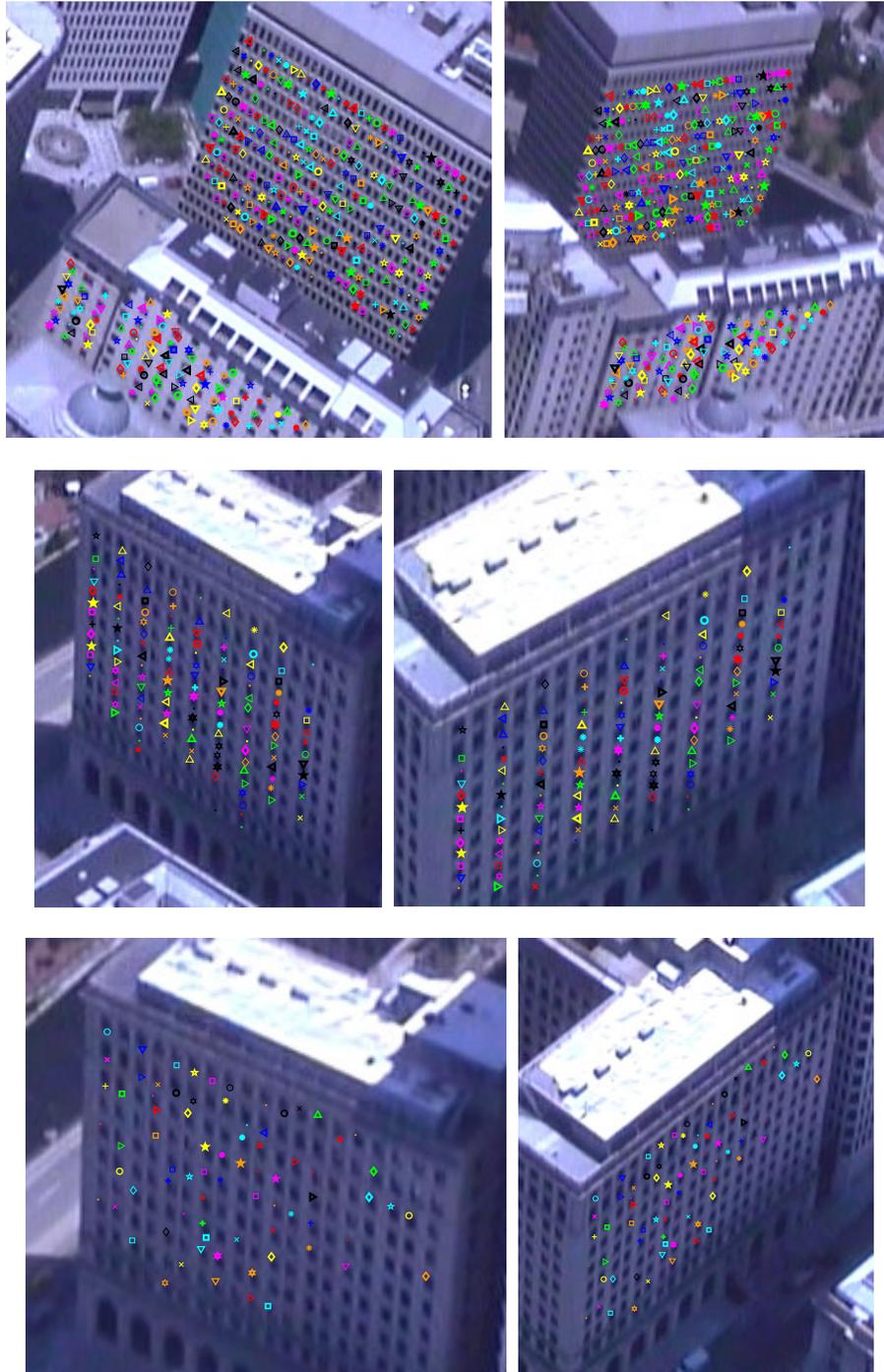


Figure 8.21: Additional results of the proposed wide-baseline matching of dense repetitive features. The top row shows robustness to multiple planar surfaces with distinct repeating elements. The middle row illustrate robustness for periodically missing grid lines that are undetected due to uncommon failure of the lattice line detector. The bottom row demonstrates wide-baseline matching under a significant number of missing grid matches.

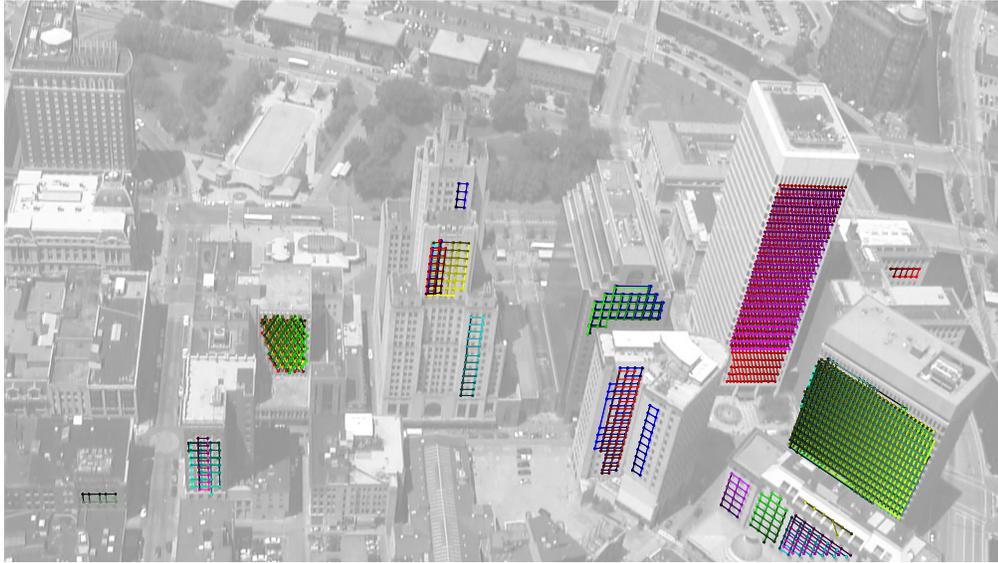


Figure 8.22: Multiple estimated lattices on a single image. A single facade may have multiple lattices because windows are seen as small blobs and blobs may generate one or more corners depending on the size of the blob, *e.g.*, a corner at the top and other at the bottom of the window. Each distinct repeating corner defines a lattice.

planar lattices when these exist.

Figures 8.19 to 8.21 and 8.23 show qualitative results demonstrating the robustness of the proposed dense repeating feature matching algorithm in the “downtown46” dataset (see figure 8.18). Figure 8.19 illustrates the method performing on large grids, on small grids, under drastic occlusion, under oblique views, and also a failure case in figure 8.19f. Figures 8.20 and 8.21 show results of the lattice tracking for successive frames achieving wide-baseline matching and its robustness to missing matches and multiple nearby planar surfaces. In general, several lattices are detected in a single image, as in figure 8.22, and are matched on a second image, as shown in figure 8.23.

Tables 8.15 and 8.16 show quantitative results for two aerial urban datasets, denoted the “capitol26” and the “downtown46” datasets. The “downtown46” scene contains many planar facades with dense repeating features, whereas the “capitol26” has planar lattice features that are not dense enough to require the use of the planar lattice model for match disambiguation. The density of repeating features is represented by a threshold on the number of similar features in the vicinity of a given feature. Repetition in the “capitol26” dataset is gracefully handled by the disambiguation of chapter 4.

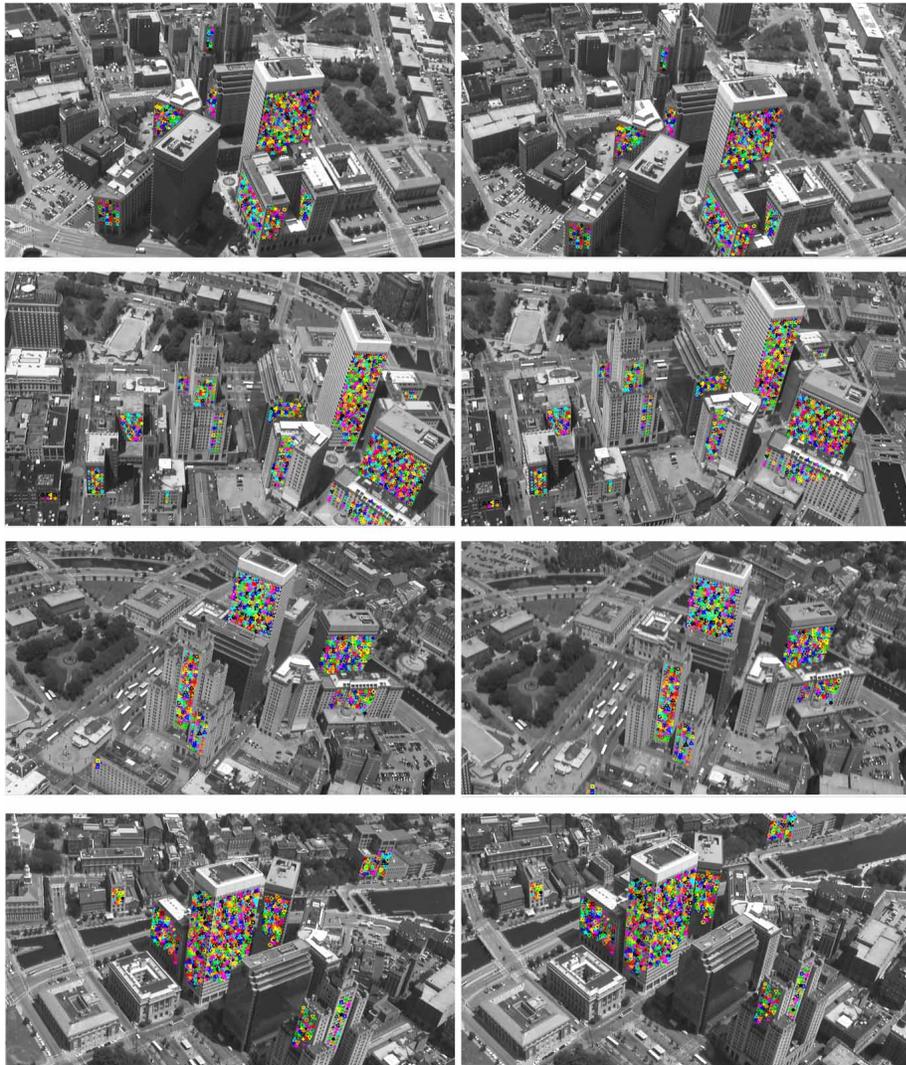


Figure 8.23: Several multiple-lattice correspondences from different angles.

Matching method	MSE	Number of 3-d points	Number of 2-d matches	MSE ratio w.r.t. SIFT
SIFT	0.349	26809	41690	1.00
NccD	0.131	54013	94416	0.38
NccD+NccDRF	0.131	54013	94416	0.38
NccD+NccDRF (NccD matches only)	0.131	54013	94416	0.38
NccD+NccDRF (NccDRF matches only)	N/A	0	0	N/A

Table 8.15: Mean squared reprojection error (MSE) for “capitol26” scene.

Matching method	MSE	Number of 3-d points	Number of 2-d matches	MSE ratio w.r.t. SIFT
SIFT	0.305	48963	76280	1.00
NccD	0.161	134686	214026	0.53
NccD+NccDRF	0.138	161872	279277	0.45
NccD+NccDRF (NccD matches only)	0.168	121747	199859	0.55
NccD+NccDRF (NccDRF matches only)	0.065	40125	79418	0.21

Table 8.16: Mean squared reprojection error (MSE) for “downtown46” scene.

As desired, NccDRF detects no dense planar lattice features in “capitol26” dataset, as reported in table 8.15. Conversely, the “downtown46” dataset presents many dense lattice features, *e.g.* as seen in figure 8.23.

Tables 8.15 and 8.16 have five rows of results for the following methods: (1) SIFT matching, (2) the NccD method of chapter 4, (3) the NccD followed by the NccDRF of chapter 5, (4) the same as before restricting the statistics to the matches found using the NccD method (ignoring the ones from NccDRF), and (5) similarly restricting to the dense feature matches found on planar facades by the NccDRF method (ignoring the ones from NccD). The columns show results for the reprojection error of equation (3.15), the number of 3-d points associated to matched features on the entire scene, and the number of 2-d features matched in the images of scene. Note, when using NccDRF following NccD, NccDRF will discard some sparse feature matches found by the NccD on regions it detects to be of planar lattices of dense repeating features and replace them by its own matches, which are more reliable. This replacement causes small drops in the number of features found in the fourth row compared to the second row. Also note that the number of 3-d points is usually higher than the number of 2-d matches because a single 3-d point may be seen in multiple images.

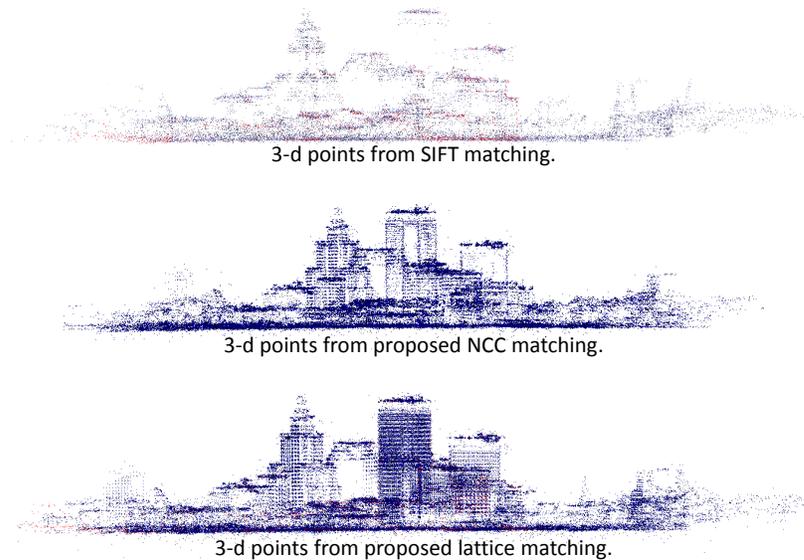


Figure 8.24: Comparison of 3-d points returned by bundle adjustment on the “downtown46” scene. SIFT matching, at the top, returns fewer points and does not include the dense repetitive features from windows in building facades, as opposed to the proposed methods, NccD at the middle and NccD+NccDRF at the bottom, which detect repeating features and others that SIFT matching does not disambiguate. SIFT features are concentrated mostly only on the ground and at the roofs of the buildings. NccDRF matches a large number of *dense* repeating features, as seen in the bottom, while NccD can disambiguate a few of them, as seen in the middle.

Discussion. The experiments show that the reprojection error of the reconstructed features of a scene diminishes when using the proposed methods for matching than when using SIFT matching. For the “capitol26” scene, the proposed NccD method attains an average reprojection error that is only 38% of the SIFT matching error. For the “downtown46” scene, where dense planar-lattice features are found, the NccD presents approximately half of the reprojection error of SIFT matching, while the features found using the NccDRF method present only nearly a fifth of the error. Note, the numbers of associated feature pairs of the proposed matching methods are always higher than SIFT matching. Similarly, the number of estimated 3-d points of the proposed methods tend to be higher, especially when augmenting matches with the NccDRF method that include the windows in building facades, as shown in figure 8.24.

8.4 Dense surface estimation

The proposed pipeline of chapter 6 includes a multiple view stereo (MVS) method for 3-d point estimation and another method for surface estimation. These steps are described in chapter 1: a point cloud estimation stage (using MVS) and a meshing stage (surface fitting). At the MVS stage, a dense set of 3-d points is recovered from multiple calibrated images (section 6.4). A watertight triangular mesh is fit to the point cloud using surface priors via energy minimization techniques (section 6.7). In this section, qualitative and quantitative results are discussed for evaluating the proposed methods. Reconstructions in the form of point clouds and in the form of surface meshes are compared to state-of-the-art methods. The datasets and ground truth data used for these experiments are discussed next.

8.4.1 Evaluation datasets.

The evaluation of the proposed dense point-cloud reconstruction of multiple view stereo (chapter 6) and its associated surface estimation (section 6.7) are performed using the evaluation setup of [58, 57]. Jensen *et al.* [58] are extending the stereo evaluation benchmark of Seitz *et al.* [102, 101] by providing ground truth reconstructions and accurate cameras for a larger variety of scenes. Both evaluation setups use cameras mounted on a robot arm in order to measure camera pose with high accuracy.

In [101], a laser stripe scanner was used to measure ground truth with accuracy of 0.05 mm to 0.2 mm and two scenes are provided containing well-textured diffuse-reflectance 3-d objects that are observed from 317 camera locations located on a one-meter radius hemisphere around the objects. The image resolution of the CCD camera is 640×480 pixels and objects are lit by three fixed external spotlights. Three standard image datasets for the two scenes are defined: one with all 317 images, denoted *full*; other with a subset whose size is 47 images, denoted the *ring*; and the last with 16 images, denoted *sparse*. The smaller subsets are used to test algorithms performance on sparser image data and their cameras are distributed in a circle.

In [57], binary stripe encoding structured light scans are employed for estimating ground truth and providing a wider collection of 80 scenes with a broad range of objects found in real-world applications each one captured under different illumination conditions. The objects of [57] have

varying reflectance, texture, and geometric properties, and include fabric, groceries, fruit, sculptures, stuffed animals, metal *etc.* (see figure 8.25 for examples). Every object is captured from 49 or 64 different viewpoints at a distance of 50 cm or 65 cm. Note, the 317 images from [101] cover a full hemisphere around the objects, whereas the 49 or 64 cameras from [57] correspond to the views from the top and from one side of the object (see figure 8.26). The structure light scans display an average error estimate with standard deviation of 0.14 mm, meaning that 68.2% of its measurements are accurate up to 0.14 mm and 95.4% up to 0.28 mm. Thus, the scans are physical measurements with very high accuracy (see figure 8.27 for an example) and are considered ground truth for the purpose of evaluating MVS. Image resolutions of [57] are 1200×1600 pixels and, at each viewpoint, images are captured under varying illumination by strobing 18 LEDs in groups, providing eight illumination conditions per viewpoint (see figure 8.28).

Although the innovative evaluation benchmark of [101] strongly influenced the developments of MVS, the datasets from [57] are chosen to evaluate accuracy of the proposed reconstructions algorithms due to their higher diversity of objects, object geometry, materials, illuminations and higher image resolution. Another seminal evaluation benchmark is the one from Strecha *et al.* [112] that targets ground level outdoor scenes, and, similarly to [101], has a limited number of scenes with mostly nonspecular surfaces. Even though [101] and [112] made tremendous contributions in advancing the quality of multiple view stereo, they only answer the question of which algorithms best work for those few scenes, whereas [57] was designed to point out in which scenes algorithms work best and which ones they fail.

In addition to a quantitative evaluation using twelve scenes from [58], reconstructions from real-world aerial scenes and indoor environments are also provided for qualitative analysis in section 8.4.3.

8.4.2 Quantitative evaluation.

Among the 80 scenes provided by Jensen *et al.* [57, 58], twelve representative ones are selected to evaluate the performance of the proposed methods as a measure of its accuracy, completeness and runtime.

The evaluation of algorithms in [57] is similar to the one in [101]. In [57], there are masks near the

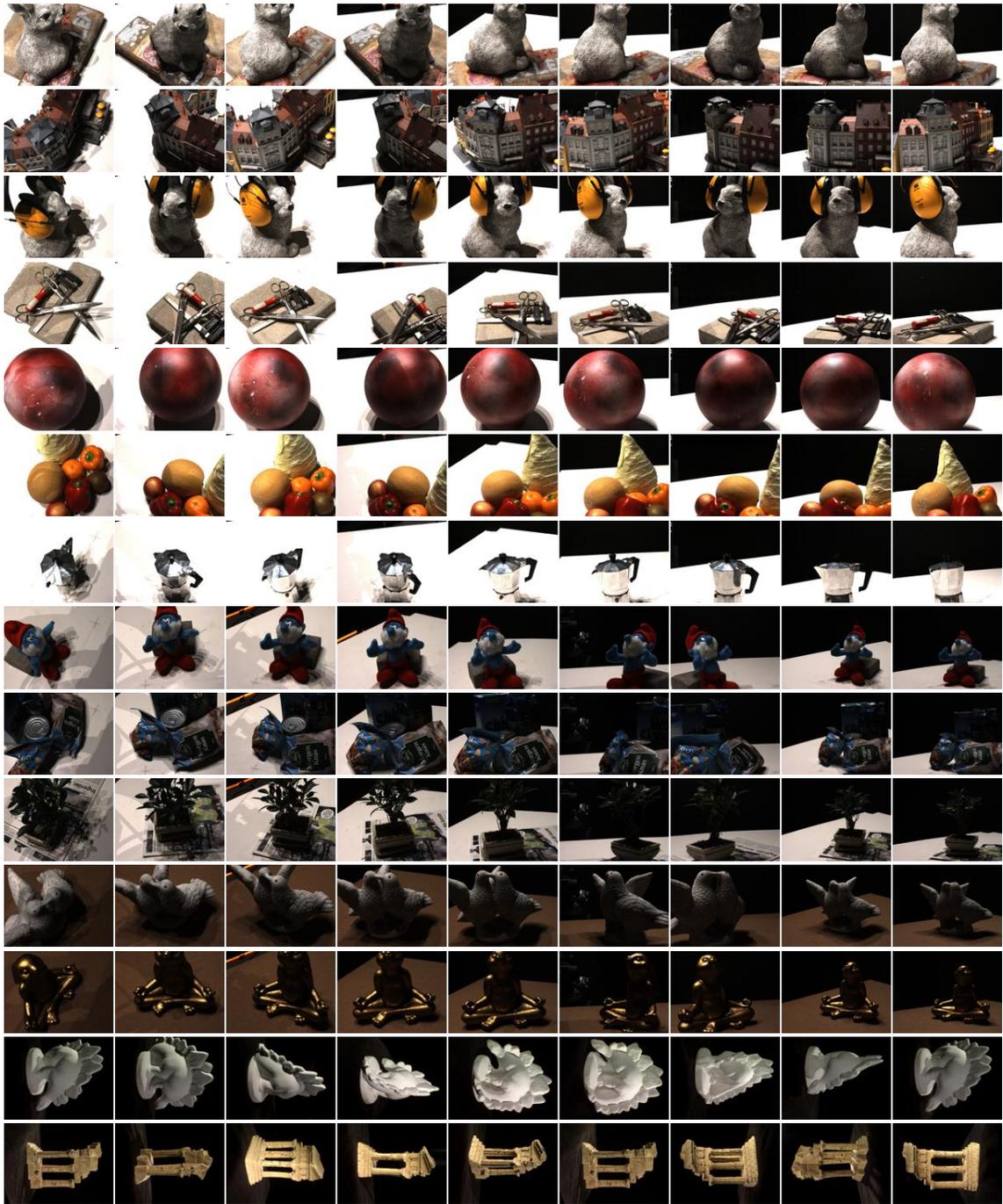


Figure 8.25: Datasets used for reconstruction evaluation. From top to bottom, the first twelve are from [58] and are denoted “bunny49”, “houses49”, “headphones49”, “tools49”, “ball49”, “groceries49”, “brewer49”, “smurf63”, “packages64”, “plant64”, “birds64” and “statuette64”. The last two are from [102] and are denoted “dinoRing48” and “templeRing47”.

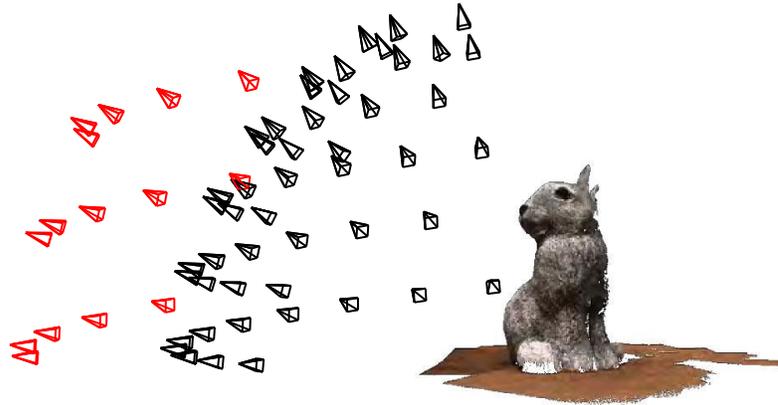


Figure 8.26: Camera positions for multiple view stereo reconstruction evaluation of [58] (image from [58]). Cameras in black are 50 cm away from the object, while cameras in red are 65 cm away.

ground truth data and only points in the masks are used for measuring the accuracy and completeness of the reconstructions. Accuracy is measured as the distances from stereo reconstruction points to the ground truth, while completeness represents the distances from the ground truth to the stereo reconstructions. Distances are computed from every point of a set to its nearest neighbor on the other set. The mean and median value of all distances below 20 mm are provided yielding the mean and median accuracy, and, the mean and median completeness. Distances above 20 mm are discarded as resulting from outliers.

The evaluation strives to provide unbiased measurements by dealing with different point densities, with outliers, with points below the supporting table and unreliable reconstructions on the unseen side of the objects opposing the cameras. Thus, additional data that may bias the evaluation is intentionally ignored from accuracy and completeness measurements (see [58] for more details). For instance, since the objects are only seen from one side, as in figure 8.26, and reconstructions on the unobserved side are not meaningful, only points that are within a certain distance to the ground truth are considered in error measurements. The evaluations are performed both in 3-d point clouds as well as in surface meshes. For surfaces, points are sampled in mesh faces and the problem is converted back a 3-d point comparison (see [58] for details).

Figures 8.29 and 8.30 show reconstruction comparisons between the proposed method and the ones of Tola *et al.* [116], and, Furukawa and Ponce [41], which are included in the evaluation of [58]



Figure 8.27: Highly dense ground truth point cloud of the “tools49” scene. The observed objects have significant specularities.

as representative algorithms from the feature expansion and depth-map fusion families. The figures show, in their first columns, an image of the dataset in which evaluations are performed. In the second columns, performances of accuracy and completeness of reconstructed points (Pts) and meshed surfaces (Sur), are provided for each dataset scene. The error is measured both as mean and median (smaller is better). The abbreviation “Tol” is for Tola *et al.* [116], “Fur” is for Furukawa and Ponce [41] and “Pro” is for the proposed method (chapter 6). The surfaces of the alternative methods are computed using Poisson reconstruction [61]. The third columns display curves for accuracy evaluations (higher is better). The graphs show, for a given distance d , how much of a reconstructed model falls within a distance d of the ground truth data. The fourth columns present completeness curves, which for a given distance d , show how much of the ground truth falls within d of the model (higher is better). The associated overall performance for all twelve datasets shown in figures 8.29 and 8.30 is presented in figure 8.31.



Figure 8.28: Images of the “smurf63” and “statuette64” datasets that are taken with eight distinct illumination conditions from a fixed viewpoint.

Discussion. As seen in figures 8.29 and 8.30 and in figure 8.31, the performance of the proposed MVS method for *point* reconstruction is comparable to state-of-the-art methods. This discussion is based on the curve plots since they show more information than the bar plots with a mean and a median. In fact, the medians are simply abscissas in the curve graphs associated to 50% ordinates. According to the curves, the proposed *point* reconstruction accuracy is in general better than the method of Furukawa and Ponce [41] and its completeness is better than the method of Tola *et al.* [116]. There is a trade-off between accuracy and completeness among the evaluated methods. Normally, the more accurate methods are less complete and vice-versa, as already noted in [58]. Note, the proposed method achieves the best completeness for the “tools49” dataset, which contains challenging specular metallic objects like a scissors.

Regarding overall *surface* estimation accuracy and completeness, the proposed method is outperformed by the others, according to the curves. Nevertheless this does not necessarily means that subjective surface quality is worse, as shown in the example of figure 8.32. In addition, the proposed method achieves superior surface accuracy when considering error thresholds of more than 1 or 2 mm, as seen in “houses49”, “headphones49” and other datasets. The proposed surface estimation performance is degraded because of limited resolution. The memory required to construct the regular grid of voxels and the 3-d graph limits the surface resolution to a coarse level compared to the estimated point cloud and to the ground truth. Low surface resolution degrades measurements of both accuracy and completeness. If the meshing stage is replaced by Poisson reconstruction, as used by the alternative methods, the proposed surface performance evaluation is statistically similar to the alternative methods (see figure 8.33).

The evaluation dataset of Jensen *et al.* [58] provides many different illumination conditions, as

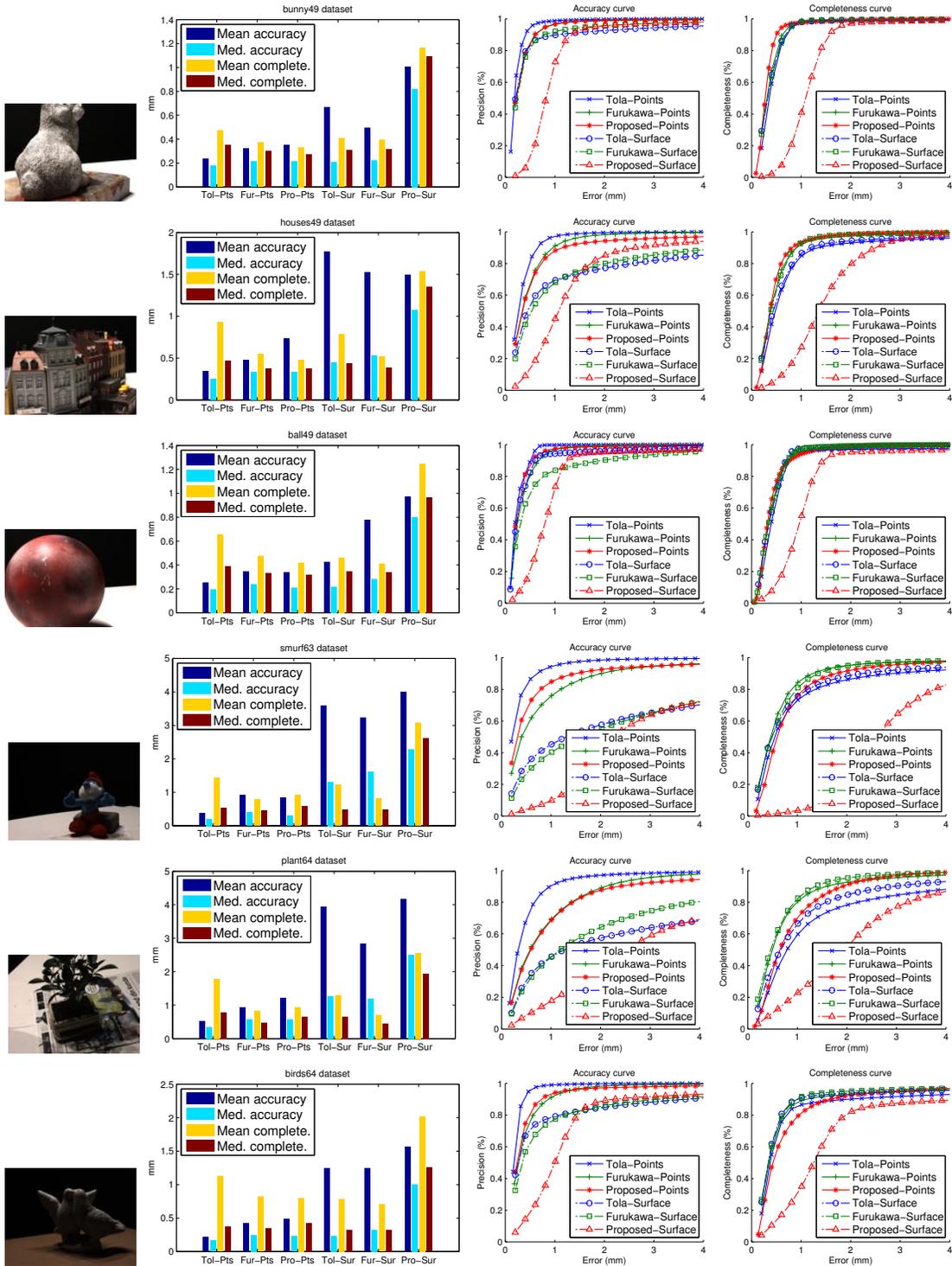


Figure 8.29: Reconstruction evaluations for scenes with well textured objects (see section 8.4.2 for details and definitions of abbreviations).

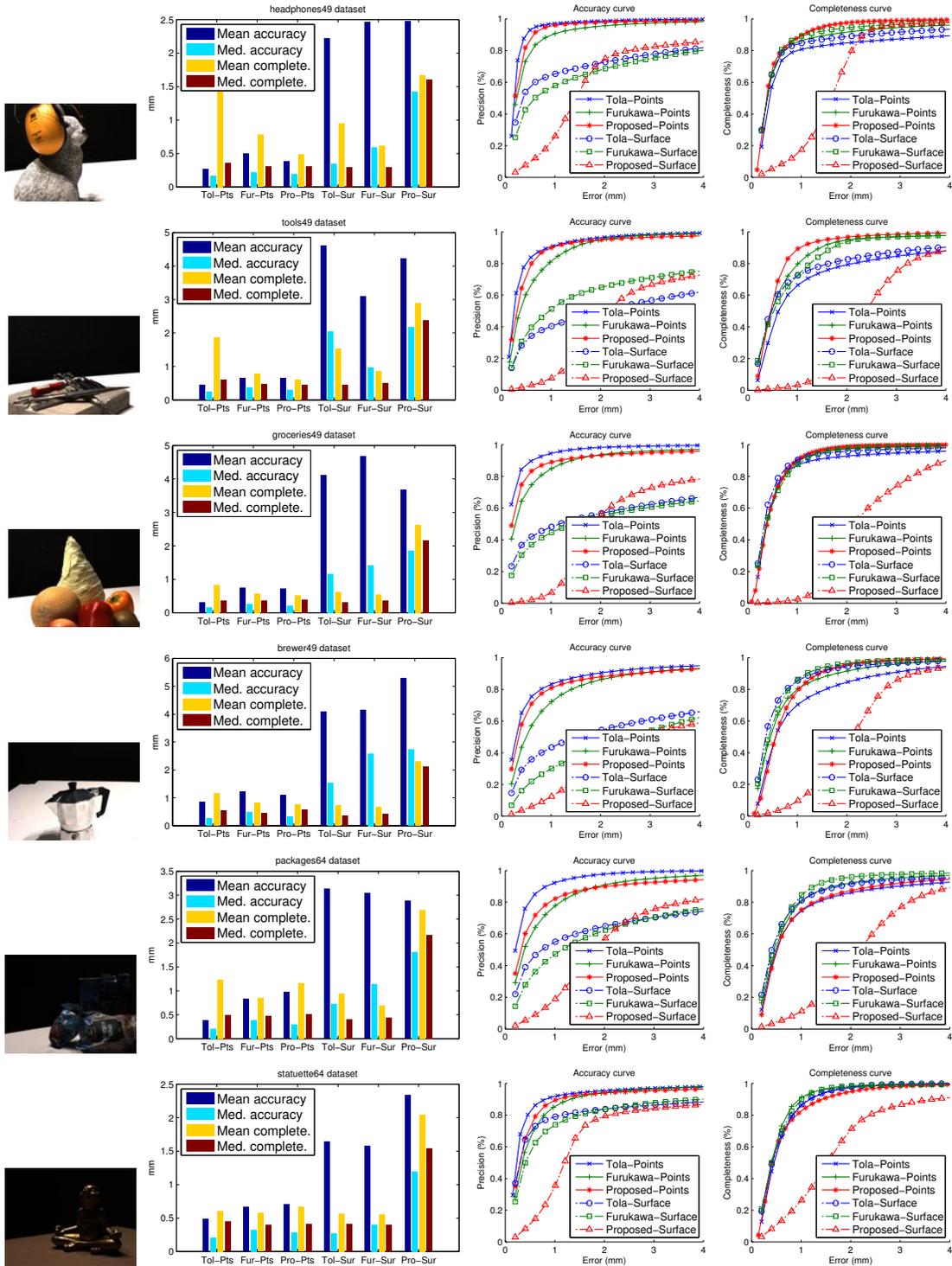


Figure 8.30: Reconstruction evaluations for scenes containing textureless or highly specular objects (see section 8.4.2 for details and definitions of abbreviations).

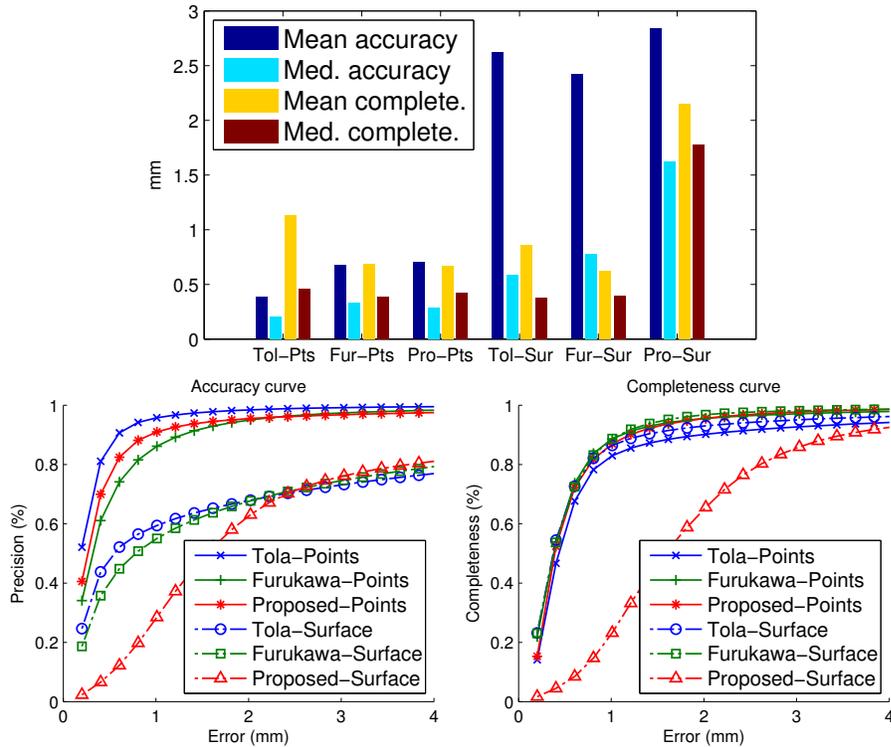


Figure 8.31: Overall reconstruction performance comparisons for all twelve evaluated scenes shown in figures 8.29 and 8.30 (see section 8.4.2 for discussion).

shown in figure 8.28, that are exploited here to show the robustness of the proposed method to varying illumination. Figure 8.34 presents results comparing reconstruction evaluations for a dataset with fixed illumination and the same dataset with images from the same viewpoints, but with alternating illumination from frame to frame. Images are selected such that illumination alternates between eight lighting conditions and only repeats after eight images. Since the MVS method uses four neighboring views per reference image, all images in a group have distinct illumination. The results in figure 8.34 show that the reconstructions are not affected by lighting variations.

Tables 8.17 and 8.18 present statistics and running times for the proposed methods. The provided running times showcase the speed of both the GPU point estimation and CPU surface estimation methods. A reference image and neighboring views are processed by the GPU in roughly 2 seconds (all overheads included) resulting in high-resolution depth maps. The images can be processed as fast as their acquisition process, *i.e.* in real-time. The CPU method takes slightly more time to return a

Multi-view stereo						
		#Images #rays #Points Runtime (s)				Total
		#Images	#rays per view	#Points per view	Runtime (s) per view	
Datasets	bunny49	49	532498	129131	1.8491	90.6059
	houses49	49	507836	99581	1.8478	90.5442
	headphones49	49	562974	53055	2.4411	119.6155
	tools49	49	438519	40581	1.4332	70.227
	ball49	49	577982	66164	2.3325	114.2909
	groceries49	49	477197	52700	1.524	74.6742
	brewer49	49	428750	6966	1.2706	62.2612
	smurf63	63	466983	17853	1.9387	122.1403
	packages64	64	534682	54293	2.3256	148.8406
	plant64	64	514736	34720	2.2059	141.179
	birds64	64	620122	61756	2.7574	176.4749
	statuette64	64	575242	69856	2.4488	156.7256
	dinoRing48	48	273446	60377	0.37027	17.773
	dinoRing48	48	98440	24981	0.13588	6.5221
	templeRing47	47	256894	115502	0.54388	25.5621
	templeRing47	47	77690	35046	0.15449	7.2611

Table 8.17: Statistics for proposed multiple view stereo pipeline. Runtimes are on GPU. Values that are defined *per view* are averaged from depth maps that are estimated per reference view (with four nearby images).

mesh from given point clouds and operates in near real-time speed. However, the mesh needs to be computed only once and computational complexity is a function of 3-d grid resolution and not the number of images. For instance, the “bunny49” dataset required roughly 90 seconds of processing time for 49 depth maps and about 24 seconds for fusing and meshing them. The GPU and CPU used in these experiments were the “HD 7970” and the “i7-950” defined in section 8.1.2.

In table 8.17, the datasets “dinoRing48” and “templeRing47” are processed at two different resolutions to show how resolution affects complexity. In order to put these GPU running times into perspective, comparison charts with running times of the *fastest* alternative methods of the evaluation benchmark of [101] are provided in figure 8.36, showing that the proposed method and its variants are among the fastest of all 74 methods listed there as of February 2015. In terms of speed, the proposed method is only behind the methods labeled “Merrell Stability” [79] and “Merrell Confidence” [79], and sometimes behind “Zach2” [137] and “Vu” [126]. However, the proposed method is more accurate than “Furukawa 3” [41] from result of the proposed evaluation (figure 8.31) and “Furukawa 3” is ranked as one of the best methods of the evaluation of [101], as shown in table 8.19.

		Surface extraction		
		#Voxels (millions)	Graph construction time (s)	Max-flow time (s)
Datasets	bunny49	12.7215	20.7268	3.2173
	houses49	12.444	18.5199	3.9587
	headphones49	17.8848	21.1265	1.8732
	tools49	10.0737	12.1259	1.5266
	ball49	14.5854	19.8275	2.3894
	groceries49	11.505	14.2052	1.5348
	brewer49	11.2464	11.6883	0.76381
	smurf63	12.3201	12.8766	1.1071
	packages64	13.554	16.4677	2.3387
	plant64	13.7115	16.5718	2.1847
	birds64	16.8237	23.9801	4.36
	statuette64	15.0792	20.3135	3.3579
	dinoRing48	27.0384	35.3328	14.8631
	dinoRing48	5.8404	8.5158	6.5816
	templeRing47	18.8976	39.3407	45.7419
	templeRing47	3.1416	5.8909	6.989

Table 8.18: Statistics of proposed surface extraction pipeline. Voxel grid resolution is provided as number of voxels. Graph construction time refers to the CPU time to embed the MVS data into a volumetric representation and build an associated 3-d graph, which is solved via graph cuts using a max-flow algorithm.

Table 8.19 also show that the fastest methods are not as accurate, showing up to 3X the accuracy error of “Furukawa 3”. In conclusion, the proposed method surpasses the accuracy of one of the most accurate alternative MVS methods and has speeds comparable to the fastest MVS methods, which present only moderate quality accuracy, therefore the proposed method is among the fastest *and* most accurate methods. Note, the accuracy of the proposed method is outperformed by the method of Tola *et al.* [116] (see figure 8.31). However, [116] is an efficient CPU method that is roughly 100X slower than the proposed MVS method running on a GPU.

8.4.3 Qualitative evaluation

Reconstruction of indoor and outdoor scenes are also provided for visual evaluation. Aerial scenes are included as the proposed solutions aim to perform well in uncontrolled environments. Figure 8.37 shows results of reconstruction from aerial scenes. Figures 8.38 and 8.39 present reconstructions in indoor environments. Successful recovering of thin geometry is demonstrated in figure 8.39. A partial

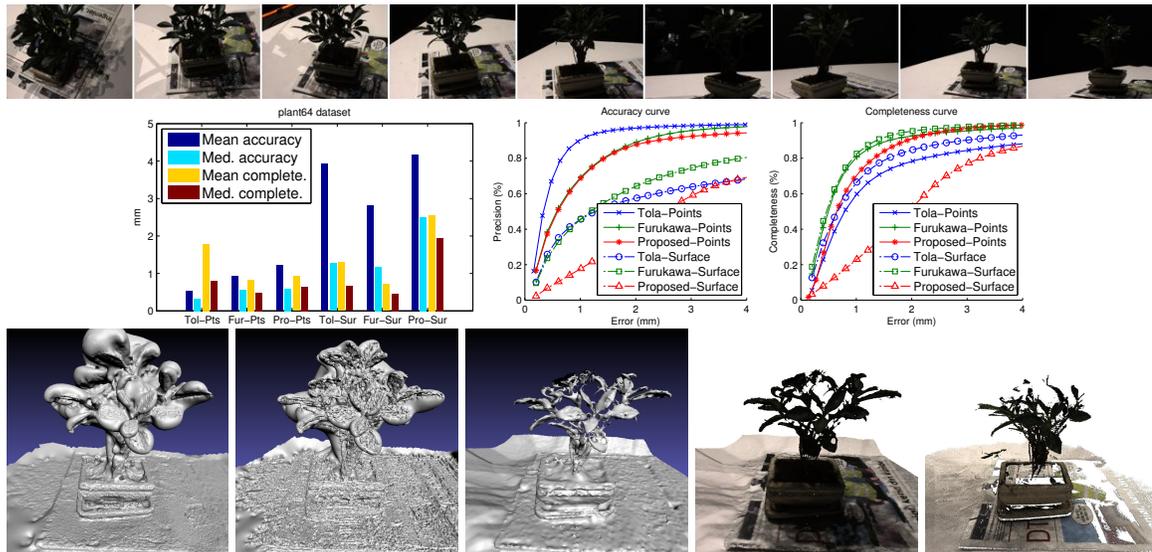


Figure 8.32: Surface reconstruction for thin geometry. The top row has images of the “plant64” dataset. Quantitative evaluation results are shown in the middle row. The bottom row displays reconstructions (from left to right): Tola *et al.* [116] with Poisson reconstruction, Furukawa and Ponce [41] with Poisson reconstruction, proposed (untextured), proposed (textured) and ground truth. The proposed *surface* reconstruction seems more representative of the thin geometry of the object than the alternative ones, despite quantitative evaluations results.

failure case is shown in figure 8.40. Reconstructions used in the quantitative evaluation section are displayed in figure 8.35.

Thin geometry. Thin geometry is commonly a problem for surface estimation algorithms as it defies smoothing priors. For instance, the proposed method fits a mesh as a minimal surface constrained to photo-consistent locations and a ballooning term. Thus, the model encourages smaller areas and larger volumes. However, thin structure, *e.g.* wire, is approximately a cylinder of zero radius and when the radius of a cylinder goes to zero, the volume converges to zero faster than the area. Hence, reconstruction methods tend to favor discarding such object completely as the cost reduction of its tiny volume does not compensate the cost of its area. Similarly, models that favor piecewise constant regions often see the discontinuity of a tiny cylinder in space as noise and discard it.

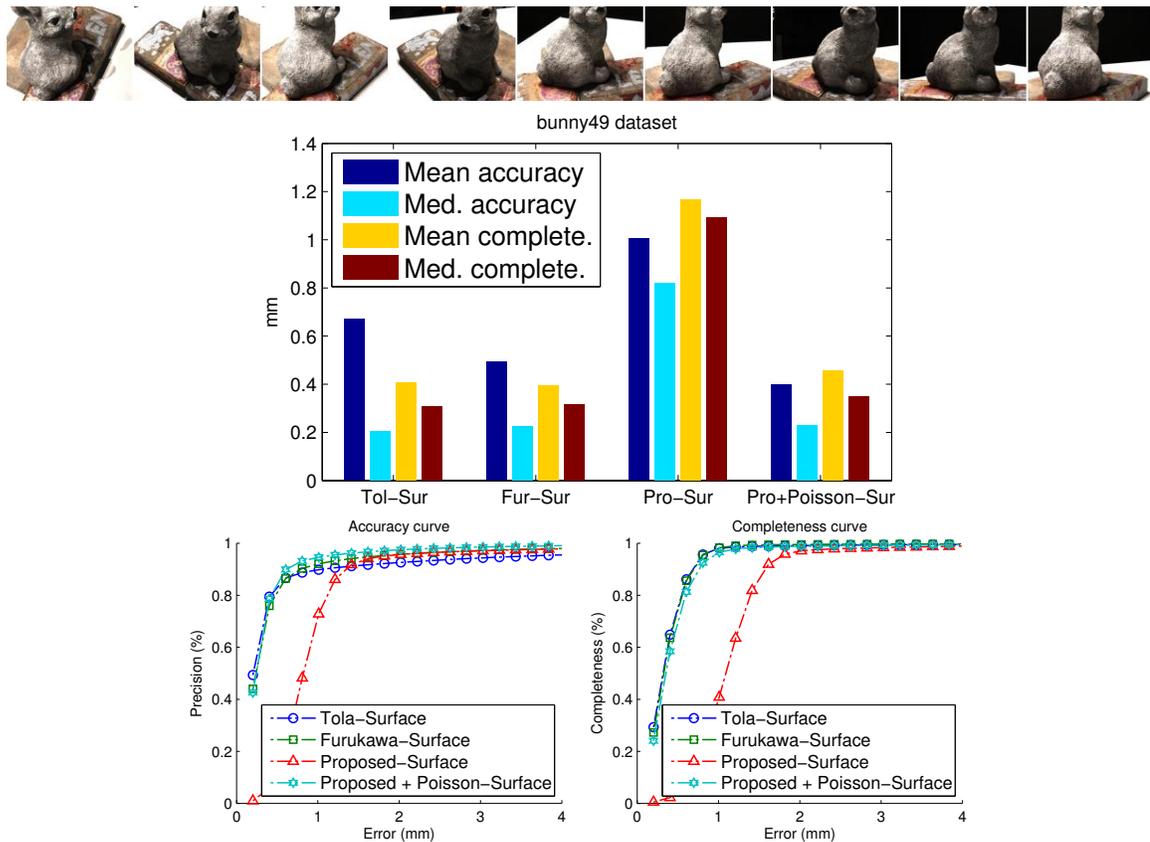


Figure 8.33: Performance evaluation comparison of the proposed MVS algorithm paired with the proposed surface estimation method and paired with Poisson reconstruction against alternative MVS methods that also use Poisson reconstruction. The experiment uses the “bunny49” dataset. When all methods use Poisson surface reconstruction, the proposed method is comparable to the alternative methods and slightly better in accuracy.

Discussion. The method performs well in both indoor and outdoor uncontrolled environments, as shown in figures 8.37 and 8.38, and can reconstruct thin geometry under appropriate model resolution (figure 8.39). The experiments show many successful examples of reconstructions from aerial views in figure 8.37. The failure case shown in figure 8.40 is representative of occasional failures that may happen near textureless surfaces and near the boundary of the volume of interest. Border regions are challenging since part of their neighborhood is unmodeled. Textureless regions are often not reconstructed by MVS, thus generating holes that the surface extraction must try to fill, however if such holes also happen near the border, meshing based on surface priors fail due to lack of information.

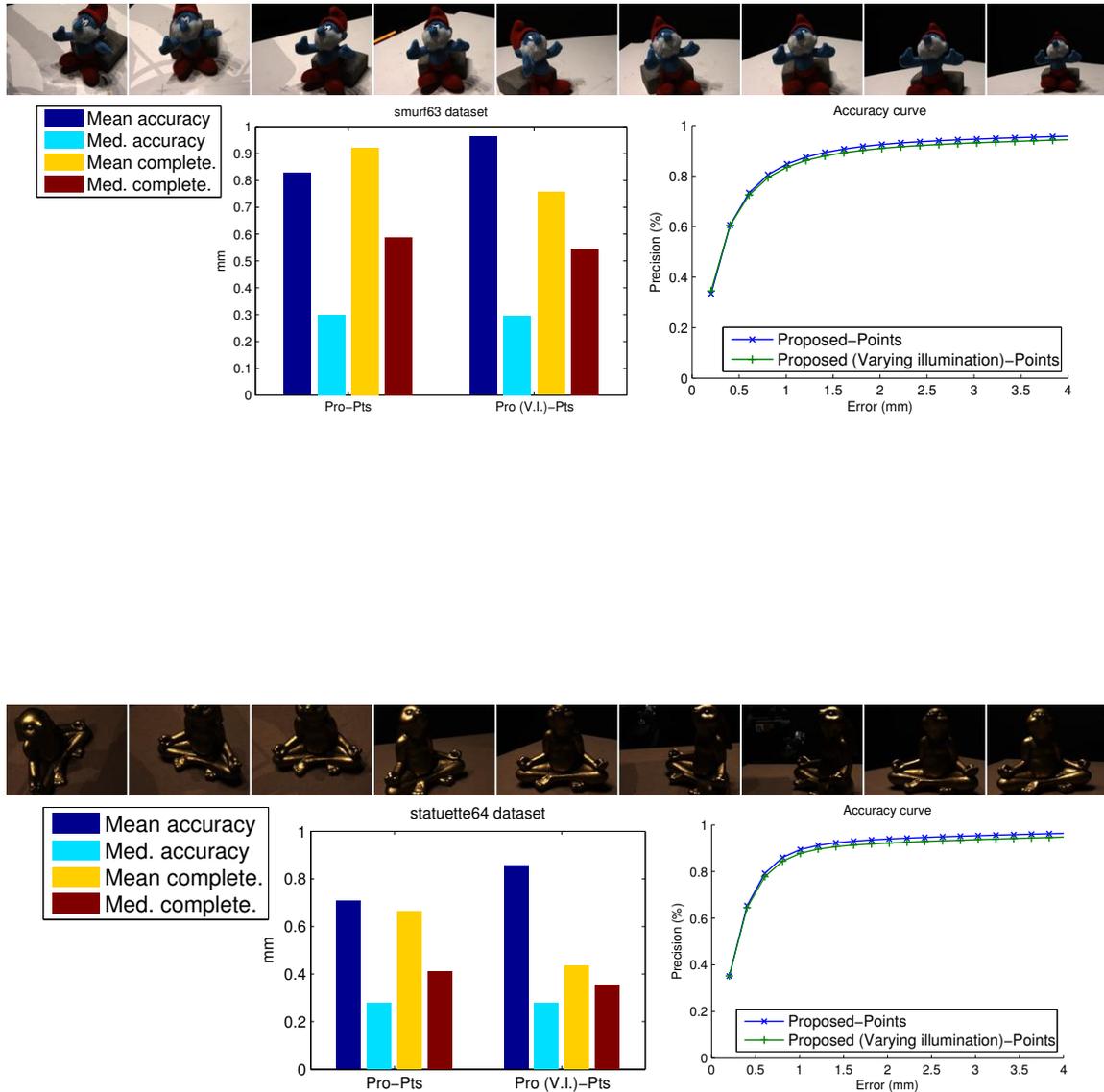


Figure 8.34: Robustness of proposed method to varying illumination (V.I.). Images from the “smurf63” and “statuette64” datasets were provided to the proposed MVS algorithm in two forms: with fixed illumination and with illumination alternating between eight different conditions. No set of nearby views had two images with the same lighting. The performances are statistically the same even though the statuette is a specular object. Moreover, the completeness (yellow and red bars) actually improved (smaller is better) when forcing illumination changes on these experiments due to statistical variabilities of point estimation.

8.5 Conclusions

The proposed GPU implementation of NCC achieves a remarkable speedup with respect to alternative optimized CPU and GPU algorithms. Moreover, the reported speedup of the proposed implementation is much higher than speedups reported by prior work on accelerating NCC for various tasks.

A solution is presented to the camera pose estimation problem using NCC and results show that its estimation accuracy is better than is the accuracy of SIFT matching, a standard technique for this purpose. In addition, NCC is successfully applied to the detection, matching and tracking of dense repeating features. The ambiguity of the problem is resolved assuming a planar model and estimation accuracy of matching feature locations found in this way is also much higher than is SIFT matching.

The evaluations of the proposed multiple view stereo method for point reconstruction indicate that the proposed approach is among the fastest MVS methods and also among the most accurate, and its completeness is comparable to the state-of-the-art methods. The proposed surface estimation method generates visually pleasing results, but it is outperformed by alternative approaches due to its limited resolution. However, using a point cloud from the proposed MVS method with Poisson surface reconstruction yields a meshing method that is also as fast and as accurate as the fastest and the most accurate state-of-the-art pipelines, respectively. These results were assembled based on comprehensive evaluations performed in twelve different datasets containing a variety of objects.

The proposed surface reconstruction method works well in uncontrolled environments such as aerial scenes and produces photo-realistic renderings of the reconstructed scene. The approach satisfactorily models thin geometry and flat surfaces even though it assumes no known prior information about surface shape. This chapter also shows many successful surface reconstructions in indoor environments from a variety of objects including stuffed animals, scissors and a plant.

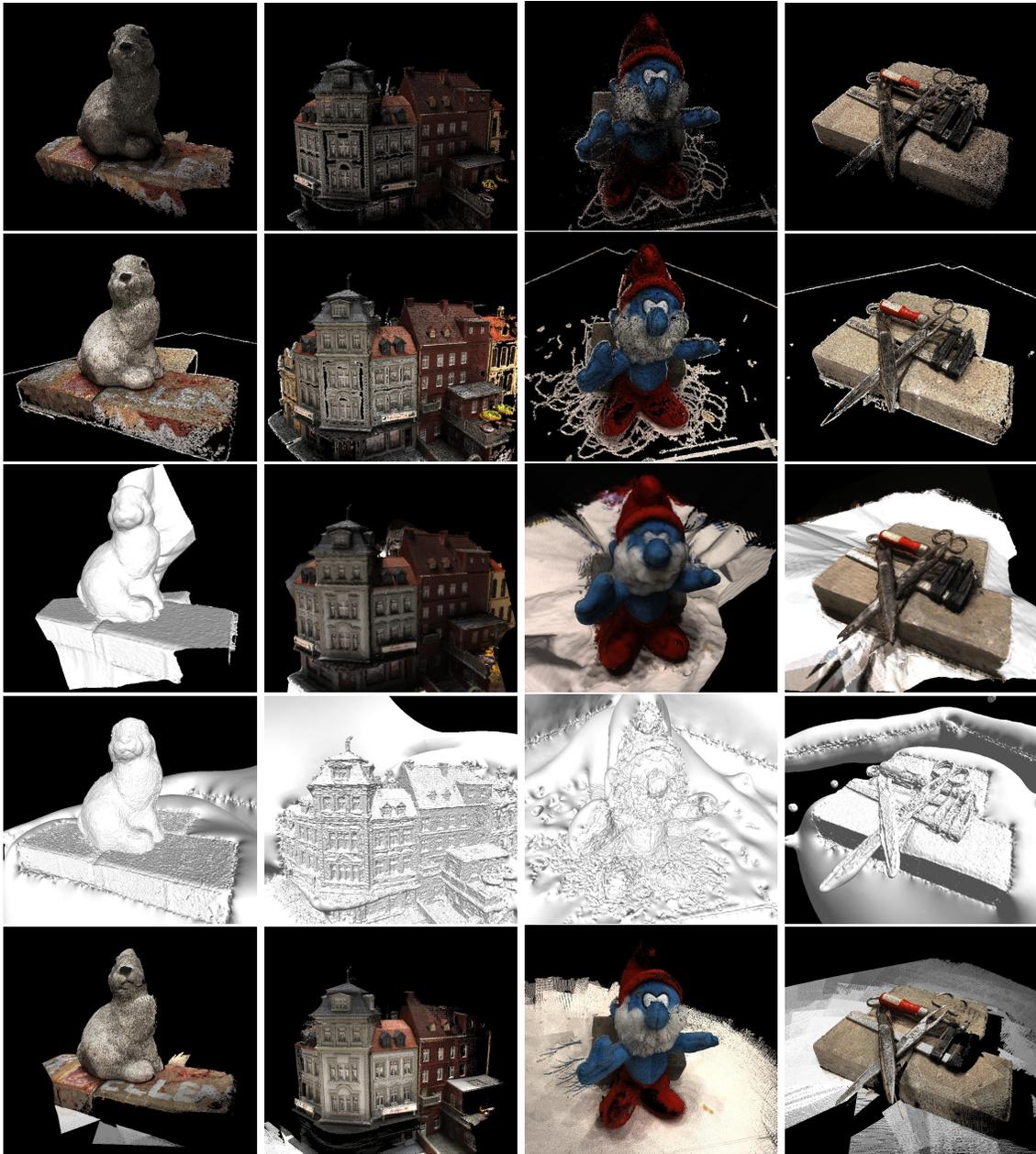


Figure 8.35: Reconstruction of objects from datasets used in evaluations in section 8.4.2. The first row shows the proposed method’s point reconstructions. The second row displays points from Furukawa and Ponce [41], which are visually more complete, but less accurate than the proposed method, in accordance with evaluation measurements. The third row displays surfaces from the proposed method. The fourth row illustrates surfaces recovered from Furukawa and Ponce. The last row has the ground truth.

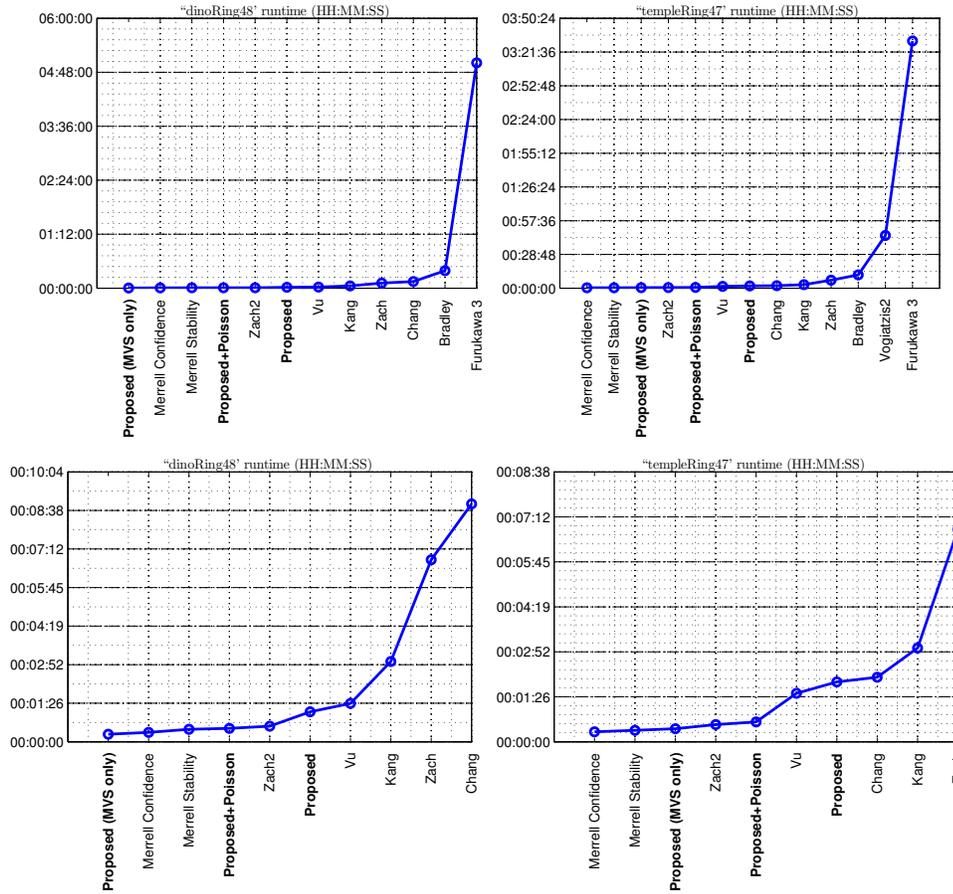


Figure 8.36: Running times of several multiple view reconstruction algorithms for the “dinoRing48” and “templeRing47” datasets. The alternative methods running times and their labels are taken from [101]. *Top*: CPU and GPU methods. *Bottom*: detail including only the GPU methods, which are the fastest among all currently listed methods (74 methods as of February 2015). “Furukawa 3” is [41], “Vogiatzis2” refers to [124], “Zach2” is [137], “Vu” refers to [126], “Chang” is [25], the “Merrell” methods are [79], “Proposed” refers to the pipeline of MVS and surface estimation of chapter 6, “Proposed+Poisson” is the proposed MVS method with Poisson surface reconstruction, and “Proposed (MVS only)” is the proposed MVS method (point estimation) without a surface fit.

		dinoRing48		templeRing47	
		Rank	Accuracy (mm)	Rank	Accuracy (mm)
Methods	Furukawa 3	1	0.28	5	0.47
	Zach2	30	0.51	17	0.56
	Vu	32	0.53	2	0.45
	Merrell Stability	44	0.73	42	0.76
	Merrell Confidence	47	0.84	47	0.83

Table 8.19: Accuracy results obtained from the evaluation of [101] for Furukawa’s method [41] and for the fastest GPU methods as of February 2015. Results were obtained using an accuracy threshold of 90%. The proposed method has not been evaluated in [101].

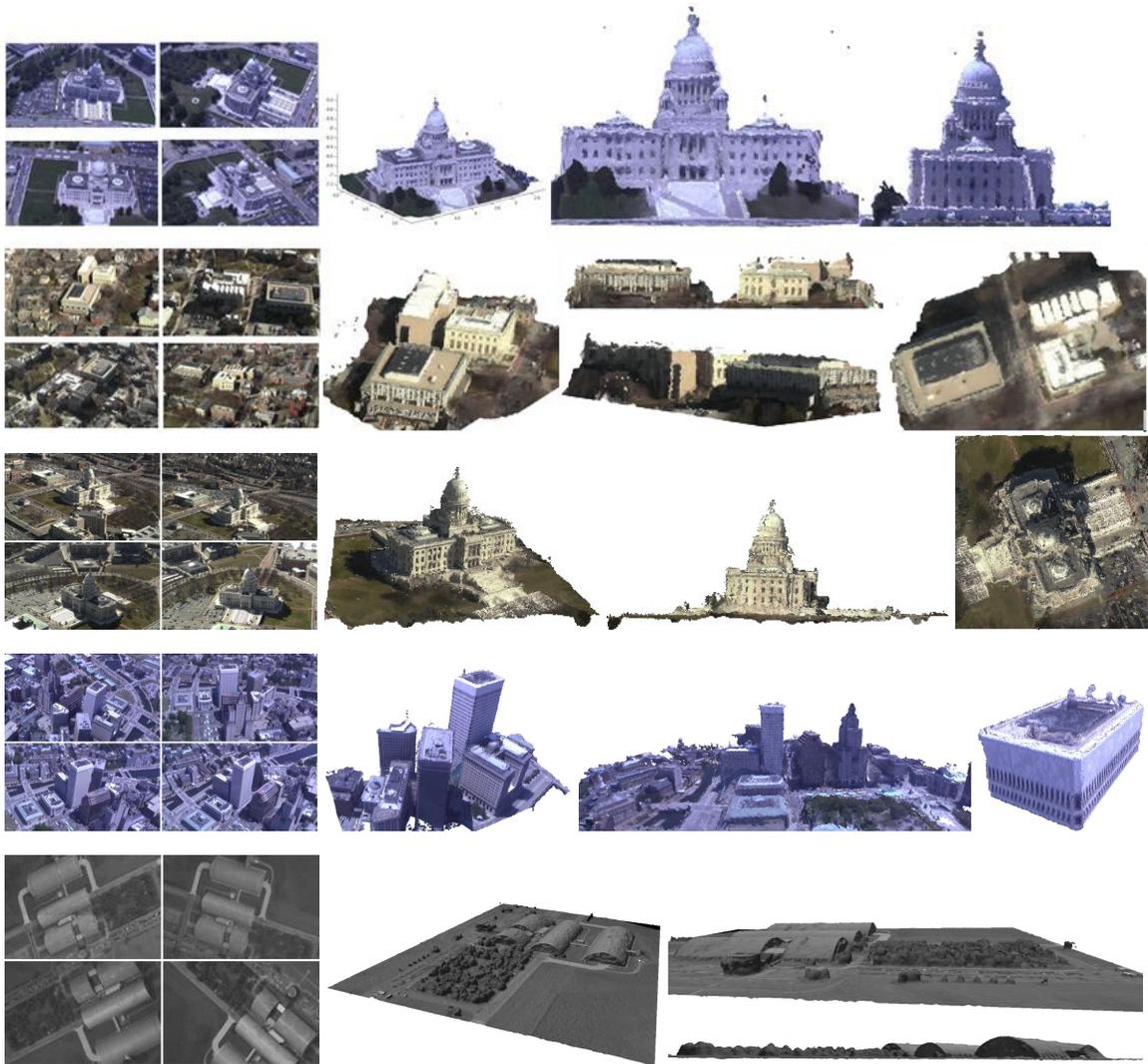


Figure 8.37: Surfaces estimated by proposed meshing algorithm running on aerial views. The mesh is painted with texture from the dataset images, shown in the left. *Top*: The Rhode Island state house building (“capitol26” dataset) showing successful renderings of the scene from an unobserved ground viewpoint and correct estimation of the building flat and curved surfaces, including one flag on the right side. *Below*: Reconstruction results on scenes containing a few buildings, streets and vegetation correctly recovered.

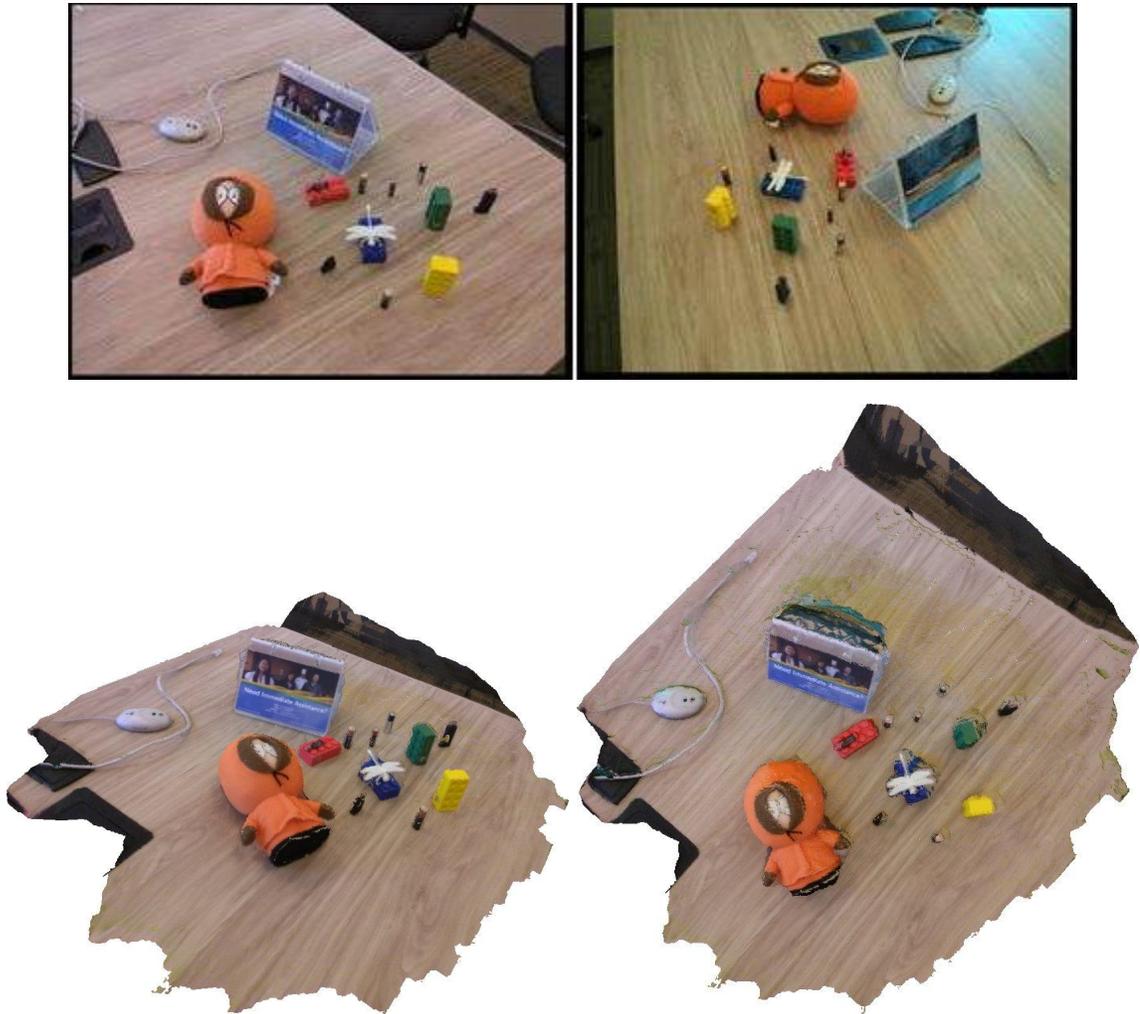


Figure 8.38: Reconstruction of an indoor environment (bottom) where radiometric variations are noticeable in the input images (top) of the “office18” dataset.

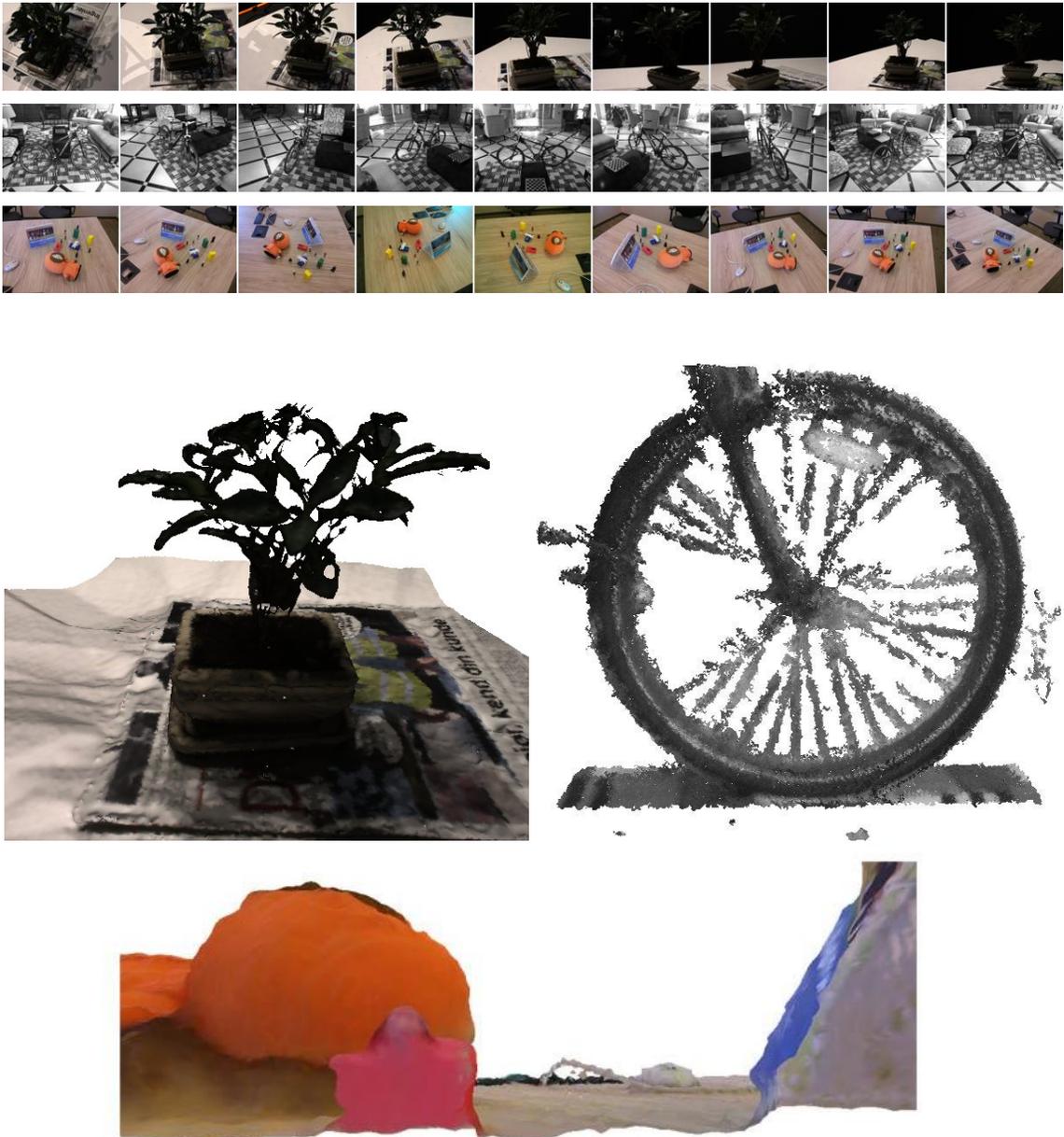


Figure 8.39: Reconstruction of thin geometry. The proposed surface estimation method handles thin geometry well if given appropriate model resolution. The images demonstrate the retrieving of plant leaves (“plant64” dataset), bicycle spokes (“bicycle34” dataset) and a network cable (“office18” dataset). The bicycle spokes are roughly 2 pixels wide in image resolution; the network cable is 4.

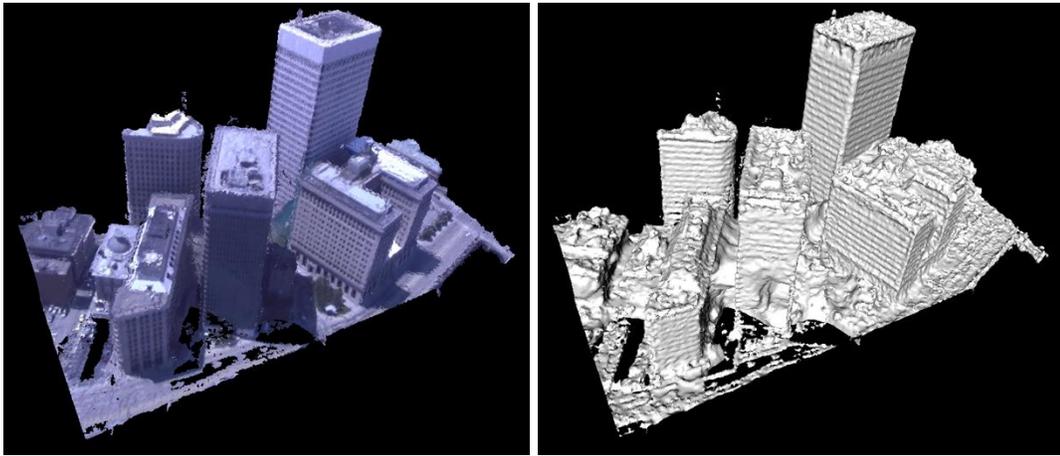


Figure 8.40: Example of failure regions on the “downtown46” dataset. Occasionally, the surface estimation leaves holes in the reconstruction, shown here as black spots. These regions are typically textureless regions, where MVS provides no point reconstruction, and are more often concentrated near volume boundaries. Fitting holes near borders is difficult since there is neighboring data only at one side. The algorithm then favors a hole instead of fitting a surface because textureless regions have low photo-consistency and therefore are of high cost.

Chapter 9

Conclusions and future work

This thesis revisited normalized cross-correlation for use in accurate camera pose estimation and accurate real-time multiple view stereo applications. The novel proposed solutions include a very fast GPU parallel implementation of normalized cross-correlation that is discussed in detail. Experiments show that the speedup of such implementation is phenomenally up to three orders of magnitude faster than alternative CPU methods and significantly faster than alternative GPU methods.

The proposed parallel normalized cross-correlation algorithm is successfully applied to feature matching for camera pose estimation and attains better location accuracy than SIFT matching, the most popular alternative method. It is hypothesized that the higher location accuracy of normalized cross-correlation matching w.r.t. SIFT is due to the way normalized cross-correlation compares the unprocessed appearance of an object directly. The SIFT descriptor incorporates a very large number of local visual fragments of the local features, which clearly builds its distinctiveness and invariance properties, but normalized cross-correlation does deliver superior accuracy in feature localization under different viewing conditions.

A new approach for matching dense repeating features also via normalized cross-correlation complements the proposed feature matching method and is able to establish remarkably accurate wide-baseline matches improving overall accuracy further. Dense repeating feature matching is a challenging task due to high ambiguity of the true match. The proposed method resolves ambiguities by assuming the features are on a plane, which support most common dense feature repetition found

in architectural scenes, such as windows of building facades.

In addition to the camera estimation application, the proposed normalized cross-correlation method is satisfactorily applied to real-time multiple view stereo to achieve real-time performance without sacrificing accuracy. The proposed pipeline runs entirely on a GPU and achieves accurate results outperforming most state-of-the-art methods in terms of accuracy *and* speed of geometry estimation. Moreover, a fast surface estimation framework is proposed to model arbitrary environments, from indoor to aerial imagery. Experiments show that this proposed procedure performs well in such environments successfully modeling surface shape, flat or curved, thin geometry and specular materials in a variety of conditions with good accuracy, good resolution and speeds that are better or comparable to alternative methods. Reconstructions are, in general, invariant to illumination conditions because the methods are based on normalized cross-correlation.

All the proposed pipelines are automatic and were tested on modern GPUs from different vendors. A comprehensive set of experiments demonstrates the claimed quality of the proposed reconstruction methods.

The parallel normalized cross-correlation utility proposed here is not limited to the proposed applications, feature matching for camera estimation and dense surface reconstruction. It is applicable in image registration, object recognition and general template matching. In order to extend applicability, future work will focus in generalizing the correlation implementation to handle a more diverse set of correlation kernels. Currently, the implementation supports square templates, Gaussian weighting and 8 bit/pixel grayscale images. Support for rotation- and scale-invariant correlation computation will be explored in the future. These potential enhancements deliver a matching process that is invariant to image rotations and adapts to the characteristic scale of interest points.

A second line of research to be explored is to fully parallelize all proposed application pipelines. For instance, there are many independent sequential CPU computations that could run concurrently processed in the proposed feature disambiguation pipelines. Moreover, the proposed surface estimation method has limited resolution due to high memory requirements of the volumetric representation with uniform grid. This approach can benefit from a more efficient volumetric representation, such as an octree, and from a GPU-accelerated graph cuts optimization solution to reach high-resolution

real-time performance.

Finally, the model-based disambiguation and matching of dense repeating features presented the most accurate localization of estimated correspondences between surveyed methods. Generalizing the approach for matching lattices on deformed surfaces and handling other primitive shapes besides planes is also useful in architectural scenes and will be investigated in the near future.

The results of this thesis indicate that normalized cross-correlation is powerful matching tool capable of providing very accurate results under a variety of conditions and at very high speeds thanks to the proposed parallel algorithms and to the massive processing power of modern GPUs. Furthermore, extensive experiments regarding the proposed applications demonstrate significant gains with respect to traditional alternative methods, in terms of estimation accuracy and speed.

Bibliography

- [1] Advanced Micro Devices. *AMD Accelerated Parallel Processing OpenCL Programming Guide*, July 2012. 152, 153, 156, 159
- [2] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *IEEE International Conference on Computer Vision (ICCV)*, pages 72–79, 2009. 18
- [3] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993. 146
- [4] Eduardo B. Almeida and David B. Cooper. Matching many identical features of planar urban facades using global regularity. In *International Conference on 3D Vision (3DV) Workshops*, December 2014. 25, 26, 82
- [5] Eduardo B. Almeida and C. W. Padgett. Automated 3D-terrain generation process from 2D aerial imagery. Technical report, Jet Propulsion Laboratory, California Institute of Technology, 2009. 19
- [6] Eduardo B. Almeida, C. W. Padgett, and D. B. Cooper. Real-time accurate surface reconstruction pipeline for vision guided planetary exploration using unmanned ground and aerial vehicles. Technical report, Jet Propulsion Laboratory, California Institute of Technology, 2012. 18

- [7] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*, pages 483–485. ACM, 1967. 166
- [8] Pavel Babenko and Mubarak Shah. MinGPU: a minimum GPU library for computer vision. *Journal of Real-Time Image Processing*, 3(4):255–268, 2008. 197
- [9] Henry Harlyn Baker. Depth from edge and intensity based stereo. Technical report, DTIC Document, 1982. 18
- [10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006. 14
- [11] Matthew Berger, Joshua A Levine, Luis Gustavo Nonato, Gabriel Taubin, and Claudio T Silva. A benchmark for surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(2):20, 2013. 20
- [12] P. Bhattacharya and M. Gavrilova. Improving RANSAC feature matching with local topological information. In *International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 17–23, June 2012. 25, 75
- [13] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Kernel descriptors for visual recognition. In *Advances in Neural Information Processing Systems*, pages 244–252, 2010. 14
- [14] J.-Y. Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/calib_doc. 40
- [15] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *IEEE International Conference on Computer Vision (ICCV)*, pages 26–33, Oct. 2003. 19
- [16] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004. 150

- [17] D. Bradley, T. Boubekeur, and W. Heidrich. Accurate multi-view reconstruction using robust binocular stereo and surface meshing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008. 129
- [18] Kai Briechele and Uwe D. Hanebeck. Template matching using fast normalized cross correlation. In *Aerospace/Defense Sensing, Simulation, and Controls*, pages 95–102. International Society for Optics and Photonics, 2001. 167
- [19] A. Broadhurst, T.W. Drummond, and R. Cipolla. A probabilistic framework for space carving. In *IEEE International Conference on Computer Vision (ICCV)*, pages 388–393, 2001. 18
- [20] F. Calakli and G. Taubin. SSD: Smooth signed distance surface reconstruction. *Computer Graphics Forum*, 30(7):1993–2002, 2011. 20
- [21] F. Calakli, A. O. Ulusoy, M. I. Restrepo, G. Taubin, and J. L. Mundy. High resolution surface reconstruction from multi-view aerial imagery. In *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pages 25–32, Oct. 2012. 18, 20, 22
- [22] Fatih Calakli and Gabriel Taubin. *SSD-C: Smooth Signed Distance Colored Surface Reconstruction*, pages 323–338. Springer London, 2012. 18, 20
- [23] Daniel Castaño-Díez, Dominik Moser, Andreas Schoenegger, Sabine Pruggnaller, and Achilleas S. Frangakis. Performance evaluation of image processing algorithms on the GPU. *Journal of Structural Biology*, 164(1):153–160, 2008. 197
- [24] Duygu Ceylan, Niloy J Mitra, Youyi Zheng, and Mark Pauly. Coupled structure-from-motion and 3D symmetry detection for urban facades. *ACM Transactions on Graphics (TOG)*, 33(1):2, 2014. 27, 28
- [25] Ju Yong Chang, Haesol Park, In Kyu Park, Kyoung Mu Lee, and Sang Uk Lee. GPU-friendly multi-view stereo reconstruction using surfel representation and graph cuts. *Computer Vision and Image Understanding*, 115(5):620–634, 2011. 16, 21, 22, 23, 24, 228

- [26] D. E. Crispell. *A Continuous Probabilistic Scene Model for Aerial Imagery*. PhD thesis, Brown University, May 2010. 18, 19
- [27] D. E. Crispell, J. Mundy, and Gabriel Taubin. A variable-resolution probabilistic three-dimensional model for change detection. *Geoscience and Remote Sensing, IEEE Transactions on*, 50(2):489–500, Feb. 2012. 19
- [28] Franklin C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, 1984. 165
- [29] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007. 13
- [30] F. Devernay and O. Faugeras. Computing differential properties of 3-D shapes from stereoscopic images without 3-D models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 208–213, Jun. 1994. 15
- [31] Petr Douthek, Jiri Matas, Michal Perdoch, and Ondrej Chum. Image matching and retrieval by repetitive patterns. In *International Conference on Pattern Recognition (ICPR)*, pages 3195–3198. IEEE, 2010. 28
- [32] Ye Duan, Liu Yang, Hong Qin, and Dimitris Samaras. Shape reconstruction from 3D and 2D data using PDE-based deformable surfaces. In *European Conference on Computer Vision*, Lecture Notes in Computer Science, pages 238–251. Springer Berlin Heidelberg, 2004. 19
- [33] Carlos Hernández Esteban and Francis Schmitt. Silhouette and stereo fusion for 3D object modeling. *Computer Vision and Image Understanding*, 96(3):367–392, 2004. 19
- [34] Ricardo Fabbri and Benjamin B. Kimia. High-order differential geometry of curves for multiview reconstruction and matching. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Lecture Notes in Computer Science, pages 645–660. Springer, 2005. 24
- [35] Olivier Faugeras and Renaud Keriven. Complete dense stereovision using level set methods. In *European Conference on Computer Vision (ECCV)*, pages 379–393. Springer, 1998. 18

- [36] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, 2005. 13
- [37] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70(1):41–54, 2006. 19
- [38] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 66, 84
- [39] Leila De Floriani and Paola Magillo. Algorithms for visibility computation on terrains: a survey. *Environment and Planning B: Planning and Design*, 30(5):709–728, 2003. 139
- [40] L. R. Ford and Delbert Ray Fulkerson. *Flows in networks*. Princeton University Press, 1962. 146
- [41] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, Aug. 2010. 23, 215, 216, 217, 221, 223, 227, 228
- [42] Yasutaka Furukawa and Jean Ponce. Carved visual hulls for image-based modeling. In *European Conference on Computer Vision*, volume 3951 of *Lecture Notes in Computer Science*, pages 564–577. Springer Berlin Heidelberg, 2006. 19
- [43] General purpose GPU programming (GPGPU) website. <http://www.gpgpu.org>. 152
- [44] A. Gallagher. Jigsaw puzzles with pieces of unknown orientation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 67
- [45] J. Mario Gallegos, J. R. Villalobos, G. Carrillo, and S. D. Cabrera. Reduced-dimension and wavelet processing of SMD images for real-time inspection. In *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation*, pages 30–36, 1996. 167
- [46] Riccardo Gherardi, Michela Farenzena, and Andrea Fusiello. Improving the efficiency of hierarchical structure-and-motion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1594–1600, June 2010. 18

- [47] Arturo Gil, Oscar Reinoso, O. Martinez Mozos, Cyrill Stachniss, and Wolfram Burgard. Improving data association in vision-based SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2076–2081, 2006. 27
- [48] M. Goesele, B. Curless, and S. M. Seitz. Multi-view stereo revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2402–2409, 2006. 16, 22, 23, 24, 126, 133, 138
- [49] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988. 146
- [50] Chris Gregg and Kim Hazelwood. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 134–144, 2011. 196
- [51] Dirk Hähnel, Sebastian Thrun, Ben Wegbreit, and Wolfram Burgard. Towards lazy data association in SLAM. In *Robotics Research*, pages 421–431. Springer, 2005. 27
- [52] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988. 14, 15, 17, 50
- [53] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. 24, 27, 32, 34, 41, 66, 72, 73, 77, 78, 178
- [54] Yong Seok Heo, Kyoung Mu Lee, and Sang Uk Lee. Robust stereo matching using adaptive normalized cross-correlation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4):807–822, 2011. 14, 15, 38
- [55] Tim Idzenga, Evghenii Gaburov, Willem Vermin, Jan Menssen, and C. De Korte. Fast 2-D ultrasound strain imaging: The benefits of using a GPU. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 61(1):207–213, 2014. 197
- [56] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon.

- KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, New York, NY, USA, 2011. 20
- [57] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. DTU robot image data sets, MVS data set – 2014. <http://roboimagedata.compute.dtu.dk>. 20, 212, 213
- [58] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 406–413. IEEE, 2014. 20, 21, 212, 213, 214, 215, 217
- [59] Nianjuan Jiang, Ping Tan, and Loong-Fah Cheong. Seeing double without confusion: Structure-from-motion in highly ambiguous scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1458–1465, 2012. 13, 27
- [60] H. Kawasaki, R. Furukawa, R. Sagawa, and Y. Yagi. Dynamic scene shape reconstruction using a single structured light pattern. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008. 30
- [61] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing (SGP)*, pages 61–70, 2006. 18, 20, 216
- [62] R. Kimmel, Cuiping Zhang, A. M. Bronstein, and M. M. Bronstein. Are MSER features really interesting? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2316–2320, Nov. 2011. 14
- [63] V. Kolmogorov and R. Zabini. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, Feb. 2004. 18, 147
- [64] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts. In *European Conference on Computer Vision (ECCV)*, pages 82–96. Springer, 2002. 146
- [65] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38:199–218, 2000. 18

- [66] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, Delaunay triangulation and graph cuts. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, Oct. 2007. 19
- [67] Patrick Labatut, Renaud Keriven, and Jean-Philippe Pons. A GPU implementation of level set multiview stereo. In *Computational Science–ICCS 2006*, pages 212–219. Springer, 2006. 21, 22
- [68] D.-S. Lee. Effective Gaussian mixture learning for video background subtraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):827–832, May 2005. 96
- [69] J. P. Lewis. Fast normalized cross-correlation. In *Vision Interface*, 1995. 165, 167, 182
- [70] Wen-Chieh Lin and Yanxi Liu. A lattice-based MRF model for dynamic near-regular texture tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):777–792, 2007. 29
- [71] Jingchen Liu and Yanxi Liu. Local regularity-driven city-scale facade detection from aerial images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 28, 29
- [72] Shubao Liu and D. B. Cooper. Ray Markov Random Fields for image-based 3D modeling: Model and efficient inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1530–1537, June 2010. 19
- [73] Shubao Liu and D. B. Cooper. Statistical inverse ray tracing for image-based 3D modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):2074–2088, Oct. 2014. 18, 19
- [74] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004. 6, 13, 14, 15, 27, 50, 55, 67
- [75] Peter J. Lu, Hidekazu Oki, Catherine A. Frey, Gregory E. Chamitoff, Leroy Chiao, Edward M. Fincke, C. Michael Foale, Sandra H. Magnus, Jr. McArthur, William S., Daniel M. Tani, Peggy A. Whitson, Jeffrey N. Williams, William V. Meyer, Ronald J. Sicker, Brion J. Au, Mark

- Christiansen, Andrew B. Schofield, and David A. Weitz. Orders-of-magnitude performance increases in GPU-accelerated correlation of images from the international space station. *Journal of Real-Time Image Processing*, 5(3):179–193, 2010. [14](#), [172](#), [197](#)
- [76] Q-T Luong and Olivier D. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices. *International Journal of computer vision*, 22(3):261–289, 1997. [40](#)
- [77] Anđelo Martinović, Markus Mathias, Julien Weissenberg, and Luc Van Gool. A three-layered approach to facade parsing. In *European Conference on Computer Vision (ECCV)*, pages 416–429. Springer, 2012. [27](#), [28](#)
- [78] Jiri Matas, Ondrej Chum, Martin Urban, and Tomáš Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767, 2004. [14](#)
- [79] Paul Merrell, Amir Akbarzadeh, Liang Wang, Philippos Mordohai, J-M Frahm, Ruigang Yang, David Nistér, and Marc Pollefeys. Real-time visibility-based fusion of depth maps. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007. [221](#), [228](#)
- [80] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, Oct. 2005. [14](#), [16](#), [17](#)
- [81] Andrew Miller, Vishal Jain, and Joseph L. Mundy. Real-time rendering and dynamic updating of 3-d volumetric data. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-4*, pages 8:1–8:8, New York, NY, USA, 2011. [19](#), [21](#), [22](#)
- [82] Hossein Mobahi, Zihan Zhou, Allen Y. Yang, and Yi Ma. Holistic 3D reconstruction of urban structures from low-rank textures. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 593–600, 2011. [28](#)
- [83] Pierre Moreels and Pietro Perona. Evaluation of features detectors and descriptors based on 3D objects. *International Journal of Computer Vision*, 73(3):263–284, 2007. [14](#), [16](#), [17](#)

- [84] O. Moslah, A. Valles-Such, V. Guitteny, S. Couvet, and S. Philipp-Foliguet. Accelerated multi-view stereo using parallel processing capabilities of the GPUs. In *3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video*, pages 1–4, May 2009. 21, 22
- [85] Patrick Mücke, Ronny Klowsky, and Michael Goesele. Surface reconstruction from multi-resolution sample points. In *VMV*, pages 105–112. Citeseer, 2011. 19
- [86] Richard A. Newcombe, S. J. Lovegrove, and A. J. Davison. DTAM: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2320–2327, Nov. 2011. 18
- [87] NVIDIA Corporation. *CUDA C Best Practices Guide*. DG-05603-001 v6.5, August 2014. 152, 156
- [88] NVIDIA Corporation. *CUDA C Programming Guide*. PG-02829-001 v6.5, August 2014. 152
- [89] Stanley J. Osher and Ronald P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002. 18
- [90] Minwoo Park, Kyle Brocklehurst, Robert T. Collins, and Yanxi Liu. Deformed lattice detection in real-world images using mean-shift belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1804–1816, 2009. 29
- [91] Harald Penz, Ivan Bajla, Konrad Mayer, and Werner Krattenthaler. High-speed template matching with point correlation in image pyramids. In *Industrial Lasers and Inspection (EUROPTO Series)*, pages 85–94. International Society for Optics and Photonics, 1999. 167
- [92] Julien Pilet, Vincent Lepetit, and Pascal Fua. Real-time nonrigid surface detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 822–828, 2005. 29
- [93] Thomas Pollard and Joseph L. Mundy. Change detection in a 3-d world. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–6, June 2007. 19
- [94] Thomas B. Pollard. *Comprehensive three dimensional change detection using volumetric appearance modeling*. PhD thesis, Brown University, May 2009. 19

- [95] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision*, 78(2-3):143–167, 2008. 23
- [96] P. Wayne Power and Johann A. Schoonees. Understanding background mixture models for foreground segmentation. In *Proceedings Image and Vision Computing New Zealand*, volume 2002, pages 267–271, November 2002. 96
- [97] Ananth Ranganathan, Emanuele Menegatti, and Frank Dellaert. Bayesian inference in the space of topological maps. *IEEE Transactions on Robotics*, 22(1):92–107, 2006. 27
- [98] Richard Roberts, Sudipta N. Sinha, Richard Szeliski, and Drew Steedly. Structure from motion for scenes with large duplicate structures. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3137–3144, 2011. 13, 27, 28
- [99] David P. Rodgers. Improvements in multiprocessor system design. *SIGARCH Comput. Archit. News*, 13(3):225–231, Jun. 1985. 166
- [100] Grant Schindler, Panchapagesan Krishnamurthy, Roberto Lublinerman, Yanxi Liu, and Frank Dellaert. Detecting and matching repeated patterns for automatic geo-tagging in urban environments. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7, 2008. 29
- [101] Steven M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. Middlebury multi-view benchmark. <http://vision.middlebury.edu/mview>. 22, 212, 213, 221, 228
- [102] Steven M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 519–528, 2006. 19, 20, 21, 22, 212, 214
- [103] Rajvi Shah, Aditya Deshpande, and P. J. Narayanan. Geometry-aware feature matching for structure from motion applications. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, January 2015. 24, 25

- [104] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994. 14, 15, 50
- [105] Noah Snavely. Bundler: Structure from motion (SfM) for unordered image collections. <http://phototour.cs.washington.edu/bundler>. 13, 18, 40
- [106] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring image collections in 3D. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 2006. 13, 18, 25
- [107] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 345–352, 2000. 18, 19
- [108] Kilho Son, Eduardo B. Almeida, and David B. Cooper. Axially symmetric 3D pots configuration system using axis of symmetry and break curve. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 257–264, 2013. 25
- [109] Hauke Strasdat, J.M.M. Montiel, and Andrew J. Davison. Scale drift-aware large scale monocular slam. In *Robotics: Science and Systems*. The MIT Press, 2010. 13
- [110] C. Strecha. *Multi-view stereo as an inverse inference problem*. PhD thesis, Katholieke Universiteit Leuven, May 2007. 19
- [111] C. Strecha, T. Tuytelaars, and L. van Gool. Dense matching of multiple wide-baseline views. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1194–1201, Oct. 2003. 19
- [112] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008. 20, 213
- [113] Olivier Teboul, Loic Simon, Panagiotis Koutsourakis, and Nikos Paragios. Segmentation of building facades using procedural shape priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3105–3112, 2010. 27, 28
- [114] The OpenCV Library. <http://code.opencv.org>. 152, 167, 182

- [115] Engin Tola, V. Lepetit, and P. Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, May 2010. [14](#), [15](#)
- [116] Engin Tola, Christoph Strecha, and Pascal Fua. Efficient large-scale multi-view stereo for ultra high-resolution image sets. *Machine Vision and Applications*, 23(5):903–920, 2012. [215](#), [216](#), [217](#), [222](#), [223](#)
- [117] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1992. [72](#)
- [118] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment — a modern synthesis. In *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer Berlin Heidelberg, 2000. [44](#), [77](#)
- [119] Du-Ming Tsai and Chien-Ta Lin. Fast normalized cross correlation for defect detection. *Pattern Recognition Letters*, 24(15):2625–2631, 2003. [14](#), [165](#)
- [120] Du-Ming Tsai and Ya-Hui Tsai. Rotation-invariant pattern matching with color ring-projection. *Pattern Recognition*, 35(1):131–141, 2002. [167](#)
- [121] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008. [16](#), [17](#)
- [122] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979. [13](#), [27](#)
- [123] A. O. Ulusoy, F. Calakli, and Gabriel Taubin. Robust one-shot 3D scanning using loopy belief propagation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 15–22, June 2010. [30](#)
- [124] G. Vogiatzis, C. Hernandez, P.H.S. Torr, and R. Cipolla. Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2241–2246, Dec. 2007. [16](#), [19](#), [22](#), [23](#), [24](#), [126](#), [127](#), [132](#), [133](#), [138](#), [145](#), [147](#), [148](#), [149](#), [228](#)

- [125] George Vogiatzis and Carlos Hernández. Video-based, real-time multi-view stereo. *Image and Vision Computing*, 29(7):434–441, 2011. 21, 23, 24
- [126] H-H. Vu, R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1430–1437, June 2009. 16, 21, 22, 221, 228
- [127] Xiaotao Wang and Xingbo Wang. FPGA based parallel architectures for normalized cross-correlation. In *International Conference on Information Science and Engineering (ICISE)*, pages 225–229, Dec. 2009. 14, 197
- [128] Kyle Wilson and Noah Snavely. Network principles for SfM: Disambiguating repeated structures with local context. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. 27
- [129] Changchang Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). <http://cs.unc.edu/~ccwu/siftgpu/>. 18
- [130] Changchang Wu. VisualSfM: A visual structure from motion system. <http://ccwu.me/vsfm/>. 18, 40
- [131] Changchang Wu. Towards linear-time incremental structure from motion. In *International Conference on 3D Vision (3DV)*, pages 127–134, 2013. 13
- [132] Changchang Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore bundle adjustment. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3057–3064, June 2011. 13, 18, 40, 43, 44
- [133] Changchang Wu, J-M Frahm, and Marc Pollefeys. Repetition-based dense single-view reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3113–3120, 2011. 28
- [134] Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Pacific Conference on Computer Graphics and Applications*, pages 225–234, 2002. 23

- [135] Jae-Chern Yoo and Tae Hee Han. Fast normalized cross-correlation. *Circuits, systems and signal processing*, 28(6):819–843, 2009. 167
- [136] Guoshen Yu and J-M Morel. A fully affine invariant image comparison method. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1597–1600, 2009. 14
- [137] Christopher Zach. Fast and high quality fusion of depth maps. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, 2008. 221, 228
- [138] Christopher Zach, Arnold Irschara, and Horst Bischof. What can missing correspondences tell us about 3D structure and motion? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008. 27
- [139] Feng Zhao, Qingming Huang, and Wen Gao. Image matching by normalized cross-correlation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2006. 14
- [140] Yong Zhao and Gabriel Taubin. Real-time stereo on GPGPU using progressive multi-resolution adaptive windows. *Image and Vision Computing*, 29(6):420–432, 2011. 21
- [141] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and vision computing*, 21(11):977–1000, 2003. 13