Close-range Machine Vision for Gridded Surface Measurement

CLOSE-RANGE MACHINE VISION FOR GRIDDED SURFACE MEASUREMENT

BY

MICHAEL KINSNER, B.Eng.Mgmt.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Doctor of Philosophy (2011) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Close-range Machine Vision for Gridded Surface Measurement
AUTHOR:	Michael Kinsner B.Eng.Mgmt., (Computer Engineering and Management) McMaster University, Hamilton,Canada
SUPERVISOR:	Dr. David Capson and Dr. Allan Spence
NUMBER OF PAGES:	xx, 199

To my parents, for their guidance,

support, and love.

Abstract

Accurate measurement of surface grids through imaging enables a variety of applications. One important example can be found in automotive manufacturing, where deformed sheet metal surface strains must be validated in safety critical regions, and rapidly measured to correct process variations. This thesis advances machine vision techniques in the context of close-range surface imaging and measurement. Sheet metal surface strain analysis provides a motivating application, but the contributions may be directly transferred to a variety of other machine vision applications where reliable, accurate measurements are required in adverse imaging conditions.

Close-range imaging in practical environments presents a number of challenges, primarily relating to depth of field blur and the regional field of view. This thesis contributes to three major components required for close-range optically-based surface measurement. First, an approach for grid line intersection measurement in the presence of significant and varying depth-of-field blur is considered, with a solution based on scale-space ridge extraction. An architecture for acceleration of the computationally intensive algorithm is then developed, and implemented using state of the art graphics (GPU) hardware. Acceleration to camera video frame rates is achieved.

The second contribution is a novel approach for interframe motion tracking of uniform gridded surfaces. The algorithm exploits topological structure of the imaged grid

pattern, thereby reducing dimensionality of the interframe tracking problem. Intrinsic fiducial measurement is proposed to avoid the need for explicit feature detectors that locate fiducials in the presence of varying size and blur. Close-range interframe tracking is demonstrated, and statistics are presented on the registration objective function.

Finally, an approach is considered for camera and hand-eye calibration of a monocular camera mounted to the tool point of a coordinate measuring machine (CMM). Pre-processing algorithms are contributed to prepare close-range gridded image data for the calibration process. Ideal model coordinate points are coherently assigned to detected grid features across video sequences, and grid approximation is performed for highly blurred image frames where reliable features have not been extracted.

The contributions of this thesis make significant progress toward enabling video frame rate, close-range, computer vision-based sheet metal surface strain analysis, and other applications where challenging image conditions impede measurement.

Acknowledgements

I would like to thank my supervisors, Dr. David Capson and Dr. Allan Spence, for their extensive guidance, support, knowledge, and availability during this research project. The many meetings and discussions taken out of their busy schedules have proven invaluable to both the direction and implementation of this work.

I would also like to thank the members of my supervisory committee, Dr. Nicola Nicolici and Dr. Shahram Shirani for their comments and support. Thanks go out to my fellow graduate students, both past and present, in the Computer Vision and Design/Manufacturing Systems laboratories.

Financial support is gratefully acknowledged from the Natural Sciences and Engineering Research Council of Canada (NSERC), Shared Hierarchical Academic Research Computing Network (SHARCNET), and Ontario Graduate Scholarships (OGS).

I would like to thank Coral and the Browne family for their support and kindness over the years, and for putting up with my unusual work hours. Finally, I would like to thank my parents and family for their unending support and guidance though my studies and also throughout life. I wouldn't be where I am without them.

Contents

List of Tables

List of Figures

List of Acronyms

3D	Three Dimensional
CAD	Computer Aided Design
CCD	Charge Coupled Device
CGA	Circle Grid Analysis
CMM	Coordinate Measuring Machine
CMOS	Complimentary Metal Oxide Semiconductor
CoC	Circle of Confusion
CUDA	Compute Unified Device Architecture
DLT	Direct Linear Transform
DOF	Depth of Field
FEA	Finite Element Analysis
FLD	Forming Limit Diagram
FLOPS	Floating Point Operations Per Second
FMA	Fused Multiply-Add
FOV	Field of View
GPGPU	General Purpose Graphics Processing Unit
GPIO	General Purpose Input Output
GPU	Graphics Processing Unit
HPC	High Performance Computing
LM	Levenberg-Marquardt
POV-Ray	Persistence of Vision Raytracer
RAC	Radial Alignment Constraint
RANSAC	Random Sample Concensus
SAD	Sum of Absolute Differences
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SM	Streaming Multiprocessor
SSD	Sum of Squared Differences
SVD	Singular Value Decomposition

List of Mathematical Symbols

α	Horizontal camera imaging scale factor (pinhole camera model)
α_n	Camera angle from surface normal in simulation experiments
α_v	Camera vergence angle in simulation experiments
β	Vertical camera imaging scale factor (pinhole camera model)
β_n	Target plane rotation angle in simulation experiments
γ	Camera imaging skew factor between axes (pinhole camera model)
ε_1	Major surface strain
ε_2	Minor surface strain
ε_3	Thickness strain
$\lambda_{1,2}$	Eigenvalues of a 2×2 matrix
λ	Scaling factor in camera calibration
ν	Calibration grid aspect ratio
κ	Calibration grid scaling factor
σ_i^k	Tracking metric ratio for connected component i in frame k
$\sigma_{(x,y)}^k$	Tracking metric ratio for connected component at (x, y) in the topological structure (frame k)
θ	Rotation angle between image and parabola coordinates
$\mathbf{A}$	Camera intrinsic calibration matrix in Zhang calibration
$\mathbf{B}$	Parameter matrix in Zhang calibration
$\mathbf{b}$	Parameter vector in Zhang calibration
C_θ	$cos(\theta)$
c	Circle of confusion diameter
c_i	Parabola coefficients
D_1,D_2	Grid principal direction vectors in model coordinate assignment
d	Lens diameter
d_c	Close DOF distance limit
d_f	Far DOF distance limit
d_k	Distance metric for k-means clustering

$\mathbf{E}$	Essential matrix from epipolar geometry
F	Linear mapping matrix for strain calculation
f	Camera focal length
$\mathbf{F}$	Fundamental matrix from epipolar geometry
$\mathbf{H}$	Homography matrix
$\mathbf{h_k}$	k-th column of homography matrix
I_j	Image coordinates of a detected feature during model assignment
$\mathbf{K}$	Camera intrinsic calibration matrix
k_1	First radial distortion coefficient
k_2	Second radial distortion coefficient
$L(x, y; t)$	Scale-space representation at pixel (x,y), scale level t
$\tilde{\mathbf{M}}$	Homogeneous world point in Zhang calibration
$\tilde{\mathbf{m}}$	Homogeneous image point in Zhang calibration
M_i	Model grid coordinate assigned to an image feature
N	Camera aperture F-stop number
n_{thresh}	Number of active pixels in thresholded connected component
n_{filled}	Number of active pixels in filled connected component
P_i	Parabola function
$\mathbf{R}$	Rotation matrix from world to camera coordinate system
r	Computational unrolling factor
r_I	Horizontal image coordinate of distortion centre
r_J	Vertical image coordinate of distortion centre
$\mathbf{r_i}$	i-th column of rotation matrix
S_θ	$sin(\theta)$
s	Camera lens to object distance
s_c	Arbitrary scaling factor in camera calibration development
s_e	Morphological structuring element
t	Scale-space scale parameter (filter variance)
$\mathbf{t}$	Translation vector from world to camera coordinate system
u_0	Camera imaging principal point in horizontal pixels
u_c	Horizontal centre of radial distortion in pixels
u_d	Distorted horizontal image coordinate

u_u	Undistorted horizontal image coordinate
u	Image coordinate in horizontal pixels
$\mathbf{V_{ij}}$	Constraint term in Zhang calibration
V	Camera lens to image plane distance
V_C	Close lens to focus point distance
V_F	Far lens to focus point distance
v_0	Camera imaging principal point in vertical pixels
v_c	Vertical centre of radial distortion in pixels
v_d	Distorted vertical image coordinate
v_u	Undistorted vertical image coordinate
v	Image coordinate in vertical pixels
$\tilde{\mathbf{X}}$	World point in homogeneous coordinates
$\tilde{\mathbf{x}}$	Image point in homogeneous coordinates
X_c	Parabola vertex in horizontal image coordinates
x_c	Parabola vertex in horizontal local coordinates
x_p	Local horizontal basis for parabola fitting
Y_c	Parabola vertex in vertical image coordinates
y_c	Parabola vertex in vertical local coordinates
y_p	Local vertical basis for parabola fitting

Chapter 1

Introduction

The content of this thesis pertains to close-range machine vision, with algorithms and techniques that enable surface measurement in important real world applications including, for example, automotive manufacturing, aerospace manufacturing, and others where precision and processing speed are imperative. This chapter begins by describing one motivating application for the work to set the context for the machine vision algorithms that are subsequently developed. The chapter then summarizes the thesis objectives and its contributions, and provides an overview of the remaining chapters.

1.1 Motivation for this Work

Automotive manufacturing is a major industry throughout the world, and is a significant driver of the economy in many geographic regions. A trend apparent in the automotive industry has been a move towards lighter weight vehicles, motivated by the need to reduce emissions to meet environmental standards, and the desire to provide longer run times from battery equipped hybrid and electric cars. Significant research is required and currently being devoted to the challenges imposed by light weight vehicle structures.

Sheet metal stamping processes are a major activity in automotive production, and are used to form components of the exterior body from flat sheet metal stock. The desire for lighter weight assemblies has reduced the mechanical redundancy in vehicle

structures, tightening the acceptable limits on forming stresses during production.

When sheet metal is stamped into the desired shape within a die, the part must have the correct three dimensional geometry for assembly, and must also adhere to the material "forming limits" (described in Chapter 2) to ensure strength. If stressed beyond the forming limits, parts may wrinkle, causing complications during assembly and leading to rust from water and salt retention. Overstress may also manifest as material thinning, leading to in-service failures, and a negative impact on crash worthiness of the vehicle. In extreme cases, overstress leads to fracture and tearing of the material, although these failures can often be detected through inspection. With low weight design increasing dependence on the body as a critical element of structural integrity, sheet metal strain analysis in industrial environments has become significant from a safety standpoint.

In the context of this thesis, sheet metal surface strain measurement is required in three specific instances. The first is industrial production, where adjustments are occasionally required to bring a manufacturing process or production line back under control. Such a situation occurs, for example, when a new batch of metal is introduced into production. Slight variations in properties between batches may cause the stamping dies to produce parts that are out of geometric tolerance. Adjustments made to the stamping process must correct the output geometry without exceeding test strain limits for the batch of metal, requiring strain to be measured during iterative adjustment (or to validate finite element based corrections).

A second need for surface strain measurement is part design. Computer Aided Design (CAD) systems for mechanical engineering can model and predict surface strains in metal parts to be produced, but feedback from test samples of the metal to be used is often required to fine tune the Finite Element Analysis (FEA) modelling. Such feedback is especially important for safety critical parts that are formed with significant strains, such as in aerospace design.

A third need for surface strain measurement is in forming research, where measurement is required to validate strain models, especially when new materials are being developed. Figures 1.1(a) and 1.1(b) show a square sheet metal sample being deformed to measure its material properties. The specialty forming machine is designed to measure metal batch properties prior to manufacturing.

(a) (b)

Figure 1.1: (a) Flat sheet metal sample before deformation; (b) Sheet metal sample deformed into a dome to measure material properties.

Three dimensional reconstruction of part surfaces is significant during design and production of a die that will stretch sheet metal into a formed part. As pressure is removed from the die, elastic deformation in the sheet metal releases and the part changes shape, a process known as "springback". Measurement of part geometry is required during die testing to ensure that the final product is within required geometric tolerance.

The typical approach to measure surface strain during production is to place markers, such as ovals or line grids, on the metal surface before deformation. The location (or shape) of these markers is either known or measured. After deformation the markers are measured, and by monitoring the relative movement and/or distortion, stretching or compression of the metal can be inferred.

1.2 Existing Approaches

1.2.1 Circle Grid Analysis

A technique commonly used to measure forming surface strains is known as Circle Grid Analysis (CGA), and involves placing circles in a regular grid pattern on the flat sheet metal blank (before deformation). The circles are typically placed on the metal though etching or printing. As the metal is stamped into shape and stretches, the

circles are deformed into ovals. A technician then compares the shape and size of the distorted ovals with a reference sheet, which gives the strain values associated with the observed deformation. This technique is common because it is simple, but the results are dependent on the skill and care of the technician performing oval matching, and significant time is required to manually compare and register the features. Grid spacing tends to be relatively large for practical measurement, leading to a coarse spatial resolution in the strain values. Even with an automated or photogrammetric approach to measuring the deformation, the resolution remains relatively coarse because of the oval size needed for reliable detection. This technique is useful during lab analysis of small sheet metal samples to determine batch properties, but is not practical for online analysis of failed production parts. As a further disadvantage, CGA does not allow non-proportional deformation to be detected [145].

1.2.2 Laser Scanner Approaches

Laser scanner systems on the market produce three dimensional models of parts, but because sheet metal surfaces are typically featureless, strain information cannot be inferred. Markers can be placed on the metal prior to deformation, but laser scanners are typically designed to generate model surface geometry only, and not to detect coloured or intensity-based features. Figure 1.2 shows a gantry CMM (Coordinate Measurement Machine) with an attached laser scanner measuring the surface geometry of a sample dome. Past work to perform CGA analysis using intensity data returned by an experimental laser scanner was described in a Ph.D. thesis by Chan [16, 17, 18].

Mobile and hand-held laser scanners often use rigid markers such as reflective dot stickers placed manually on the part to simplify tracking of the laser scanner motion, thereby simplifying point cloud registration. Although the markers are rigid on the surface and therefore move during material stretching, increasing the density for strain measurement is not practical.

In the situation where surface intensity information is provided by a laser scanner, the data becomes similar to that of a camera, but with three-dimensional point information embedded. Extracting surface features from the intensity data requires computer vision-like techniques, but on a non-uniform point grid as opposed to a

Figure 1.2: Gantry CMM-mounted laser scanner measuring surface geometry of metal dome.

uniform image sensor.

1.2.3 Gantry CMM Optical Approach

A Master's thesis by Mitchell [106] explored the idea of using stereoscopic cameras mounted to a gantry CMM to measure gridded points on a deformed sheet metal surface [52, 147]. Using position and pose information from the CMM at multiple viewing locations and angles, a three dimensional model of the part could be generated. The thesis proved the proposed concept, but is impractical for use in an industrial environment for two reasons. First, a gantry CMM is an expensive and immobile device, and has a fixed size work area, typically much smaller than parts produced during automotive sheet metal stamping processes. Second, the CMM must be programmed or manually driven to place the stereoscopic cameras at positions around the part such that the entire surface is reconstructed. The cameras must be properly focussed, so manual control of the CMM would typically be required. Either programming or manually driving the gantry device is time consuming, and leads to impracticality in real environments for online process correction.

Gantry CMM-mounted cameras were also considered for three dimensional part reconstruction by Aguilar et al. [2], but marker measurement for surface deformation was not considered.

1.2.4 Commercial Systems

Two commercial optical strain measurement systems are available from companies in Germany. They are compared with the approach taken by this thesis at the end of Section 2.2.4. Key features of the systems are summarized in Table 1.1.

GOM mbH

A commercial offering by GOM mbH provides an optical approach for strain determination through their ARGUS system [49]. Dots or a random speckle patterns are placed on a part surface prior to deformation, and are imaged on the deformed part using a single camera. Multiple snapshots are taken with the camera, and uniquely coded registration cubes may be placed around the object [49] to simplify multiple-view point correspondence. Deformation of the dot or speckle pattern is interpreted as strain information, which may be overlaid on a CAD model of the part. Dot spacing is on the order of 1-5mm [137].

Correlation of speckle pattern regions, as available through the GOM system, is described by Vacher et al. [168]. An application of these patterns for video imaging during material property testing is presented in [64].

ViALUX GmbH

A second commercial offering is the AutoGrid system, produced by ViALUX GmbH. It uses four CCD cameras mounted to a frame, which simultaneously image a part, and line-based grids are used to provide surface features. The system applied to generation of forming limit diagrams is described in [77]. Two models of the AutoGrid system are available, one for stationary mounting on a tripod, and the other designed to be held in position by an operator.

Table 1.1: Comparison of commercial optical strain analysis systems.

	GOM ARGUS	ViALUX AutoGrid
Tracked surface feature	Dots or speckle pattern	Line-based grid
Number of cameras	1	4
Advertised strain accuracy	0.01%	0.2%

1.3 Current Limitations

Optical imaging is one of the few ways that surface deformation information can be obtained from parts in an industrial environment. Current optical approaches for strain measurement suffer from some limitations. The commercial offerings attempt to address some of the issues, as does this thesis. The primary limitations of current systems include:

- Measurement equipment designed for small part sizes, and not automotive scale parts
- Motorized scanning systems require path programming
- Insufficient capability in current systems to scan large parts with high accuracy. Large parts are typically imaged from greater distances.
- Lighter weight materials require finer strain measurement grids to assume homogeneous deformation in strain computation.

1.4 Thesis Objectives

The objective of this thesis is to develop an architecture, algorithms, and techniques that move toward the goal of a high-speed, high accuracy 3D measurement system. Challenges are addressed relating to hand-guided free form scanning of surfaces at close range, where the camera field of view is small compared with the part surface.

While sheet metal surface strain analysis is a motivating application, such analysis critically relies on the accurate collection of large quantities of 3D surface measurement data - for some production environments, collection of this data may be

challenged by adverse imaging conditions and the requirements for fast measurement speed.

This thesis is therefore restricted to addressing the important issues related to the capture of surface measurement data. However, an overarching goal is to provide an approach that may be useful in practice, meaning that convenient and fast data capture performance is desirable. The algorithms that are developed (and the techniques for their computational acceleration) can be extended to other applications, and are designed to be scalable to finer mesh sizes and larger strain gradients, etc. Extension to different measurement approaches is possible, including recovery of 3D surface information using projected structured light [124, 182, 54]. It should be noted, however, that strain measurement systems specifically require affixed grids or other targets so that stretching due to deformation may be detected.

1.5 Contributions of this Thesis

The significant contributions described by this thesis include:

1. System design and work flow for a manually manipulated CMM-based monocular vision system for accurate 3D surface measurement

2. Approach for accurate sub-pixel measurement of grid line intersections in the presence of depth-of-field blur

3. GPGPU-based acceleration of the scale-space ridge extraction algorithm

4. Topologically-based algorithm for efficient and robust interframe motion tracking of surface grids across long video sequences in the presence of depth-of-field blur

5. Approach to camera calibration across close-range video sequences without unique grid identifiers (coded fiducials)

6. Algorithms to robustly assign model coordinates across large data sequences, in presence of significantly varying depth-of-field blur

1.6 Thesis Organization

This thesis is organized into seven chapters (including this introductory material of Chapter 1) as follows:

In Chapter 2, background information is provided on multiple view triangulation and its application to sheet metal strain analysis. Close-range imaging with narrow depth of field is described, and the overall architecture and data flows of the new proposed measurement system are presented.

Chapter 3 considers approaches for measurement of the imaged grid lines using conventional feature detectors. Background on scale-space feature extraction is then provided in brief, followed by description of a ridge detection approach capable of measuring grid lines in the close-range images. Computational intensity of the ridge measurement algorithm is examined, and CPU suitability for video-rate implementation considered.

Chapter 4 provides an introduction to graphics processors in the context of general purpose computation, and then describes design of the ridge detection algorithm for acceleration using state-of-the-art GPGPU hardware. Performance results are presented from experimental implementation of the accelerated algorithm. An algorithm to filter ridge data and interpolate grid line intersections through polynomial fitting is described, and an accuracy validation study is presented using synthetic data to verify correct operation of grid measurement in the presence of depth-of-field blur. Experimental results using real video sequences are also provided for comparison.

Chapter 5 presents an innovative algorithm developed for topological interframe motion tracking of gridded sheet metal samples. The algorithm provides accurate tracking of grid motion over large areas, and across multiple video frames, that is critical for proper calibration and subsequent strain measurement. Topological grid structure is used to reduce the dimensionality of the projective interframe tracking problem. Results are presented from the algorithm for typical video sequences, and statistics are provided on the objective function demonstrating the robustness of the approach.

Chapter 6 considers calibration of the system. State-of-the art techniques are briefly reviewed, and an approach for camera and hand-eye calibration of the new

system is described. This chapter then contributes image data pre-processing algorithms required for application of the close-range video data to Zhang-based calibration algorithms. New pre-processing algorithms are proposed for coherent assignment of model coordinates to imaged grid features across a video sequence, as well as for performing grid approximations in highly blurred image frames where reliable features have not been extracted.

The thesis concludes in Chapter 7.

Chapter 2

Background and System Design

This chapter describes the overall system design and resultant data flows developed to meet the thesis objectives. An overview of surface strain analysis and three dimensional reconstruction techniques are presented first, followed by an analysis of camera-to-target imaging distance to obtain acceptable measurement accuracy. The chapter concludes with a description of the system design including camera synchronization and data flows.

2.1 Surface Strain Analysis

2.1.1 Strain Computation

Stamping sheet metal between the surfaces of a die causes the metal to stretch or compress, leading to strains in the material. These strains must be within the limits of the material to ensure product strength. There are three primary strains of interest, two of which can be calculated from the motion of surface features during deformation. The surface strains are denoted ε_1 and ε_2, which are respectively called the major and minor strains. A third strain, ε_3, is the thickness strain, and can be computed from ε_1 and ε_2. A technique pioneered by Sowerby et al. [146] may be used to compute the surface strains from feature point motion as described next.

Deformation of the metal surface may be tracked by fixing rigid marks to the surface prior to forming, and then measuring the relative positions of the marks after

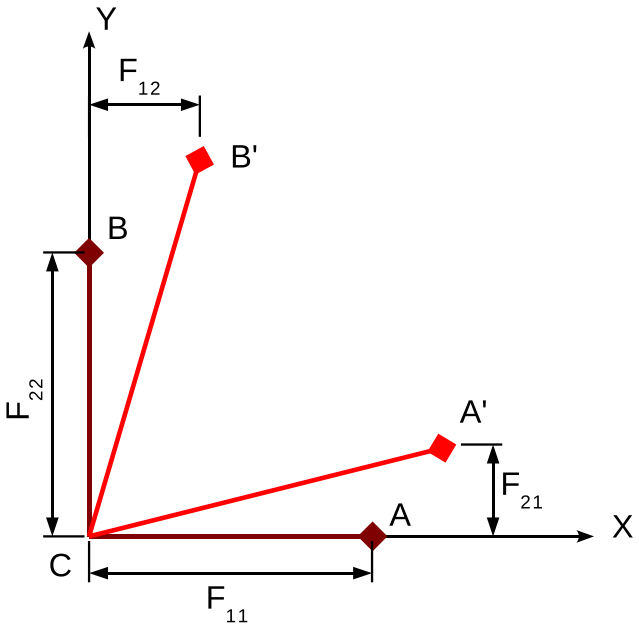


Figure 2.1: Surface point deformation by material stretching (adapted from [146]). Dark lines (to A,B) are the original point locations, and the bright lines (to A', B') show the point location after surface deformation (and stretching).

stamping. Position of the marks may either be measured prior to deformation, or a regular pattern such as a grid with known spacing may be used. Methods to rigidly place a pattern, such as a line-based grid, on the metal surface include ink-based screen printing, photo resist-based acid etching, or inkjet printing. The accuracy and repeatability of the marking process determines whether feature points need to be measured prior to stamping to ensure reliable strain results.

Figure 2.1 shows three points from a line-based grid pattern both before deformation (points A, B, C), and after (points A', B', C). Point C is chosen as the origin to measure relative displacements, so appears unchanged in the figure. Through the technique [146], a linear system is introduced to define the mapping of points across the deformation as:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{2.1}$$

The original point locations are denoted (X, Y), and the deformed locations (x, y).

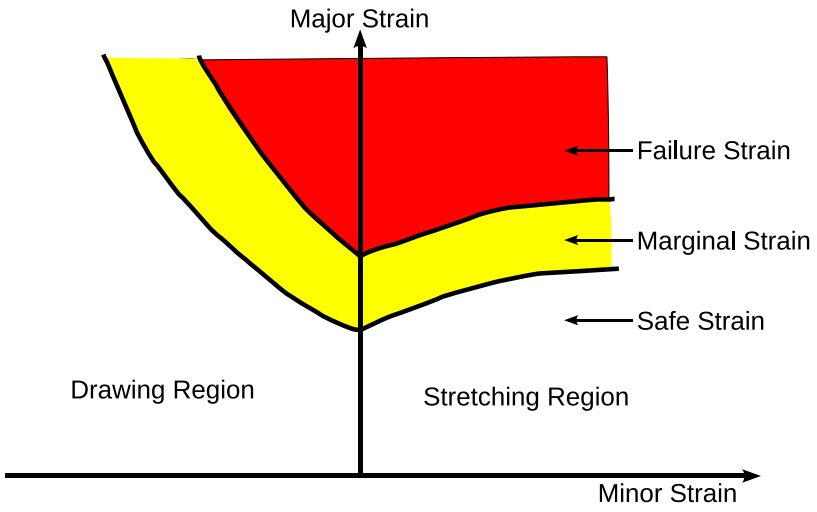


Figure 2.2: Sample forming limit diagram. Adapted from [147].

Such a linear mapping assumes a homogeneous deformation, and thus requires that the spacing of the measured points be small compared with the local strain gradient.

The transformation coefficients F are assumed to be constants for the region in which strain is being measured. By considering the undeformed line pairs AB and AC as the (not necessarily orthogonal) basis of the coordinate system, the coefficients F may be calculated from the displacements of points A and B in Figure 2.1. The coefficients F are assumed to be constant for the region, and therefore affect the deformations of both points A and B, leading to the system:

$$x' = X'F' \tag{2.2}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} X_1 & Y_1 & 0 & 0 \\ X_2 & Y_2 & 0 & 0 \\ 0 & 0 & X_1 & Y_1 \\ 0 & 0 & X_2 & Y_2 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \end{bmatrix} \tag{2.3}$$

Point A is deformed from location (X_1, Y_1) to (x_1, y_1), and point B is deformed from position (X_2, Y_2) to (x_2, y_2). The surface strain values may be computed from F according to the relationship $\varepsilon_{1,2} = ln(\lambda_{1,2})$, where $\lambda_{1,2}$ are the eigenvalues of F. Further details of the derivation are available from [146].

A third strain value, $\varepsilon_3 = -\varepsilon_1 - \varepsilon_2$, is the thickness strain.

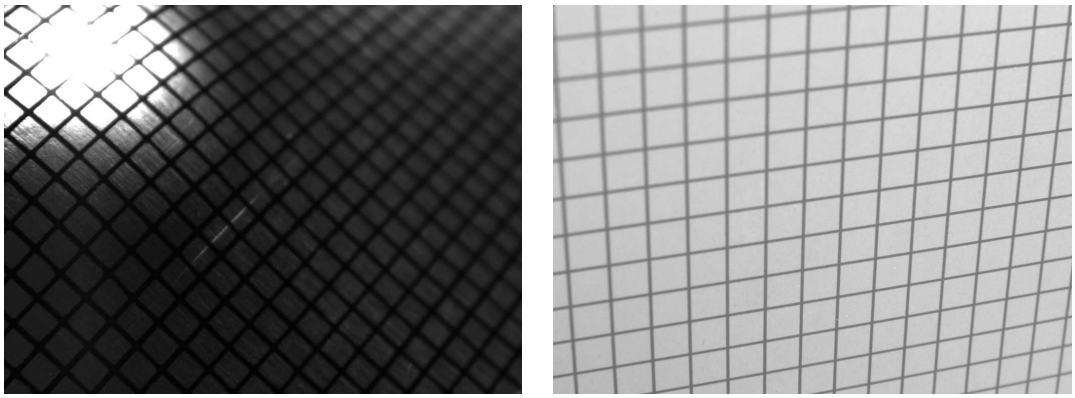
A sample Forming Limit Diagram (FLD) is shown in Figure 2.2, and is used to ensure that a part has not exceeded material strain limits during deformation. The limit curve in the FLD is often determined through progressive deformations and strain analysis of material samples from a specific batch, and is used to define safe deformations across a range of major and minor strains. Results from measurements across a part surface are typically compiled into a strain diagram, and then overlaid onto the FLD to check for values exceeding the safe limits. If the strain values are calculated at known three dimensional locations on the part surface, a thickness strain plot can be overlaid on the part geometry.

2.1.2 Surface Strain from Three-dimensional Reconstruction

The common manual approaches for strain computation require only two dimensional measurement of a part surface. Measuring the deformation of surface features is sufficient to estimate material stretching. Verification that part geometry is within tolerance, however, typically requires a three dimensional reconstruction. If the features used for 3D reconstruction are rigid on the metal surface throughout deformation, and if the feature geometry is either known or measured before stamping, a deformed reconstruction is sufficient to simultaneously validate part geometry and calculate surface strain values. With this observation, only 3D reconstruction based on fixed surface features need be considered, with the understanding that surface strain analysis follows directly from the reconstructed geometry data. The ability to simultaneously measure deformed sheet metal geometry and compute dense surface strain data is a strong advantage of vision-based measurement techniques.

2.1.3 Feature Grid Scaling

With the intended application of surface strain analysis, the size and density of printed or etched surface features affects the resolution and accuracy of the computed strain values. Standard strain computation methods are based on the assumption that each region bounded by neighbouring feature points (neighbours for a single strain value computation) undergo homogeneous deformation. During industrial manufacturing, the strain gradients are typically small enough over most of the surface that



(a) (b)

Figure 2.3: Sample images of gridded parts: (a) Deformed test dome with silkscreened grid (10 lines/inch); (b) Flat aluminum surface with photographically printed grid (10 lines/inch).

homogeneous deformation is considered reasonable [146]. As a consequence of this assumption, the frequency of the spatial measurements must be large enough that there is little to no strain gradient across each area defined by neighbouring features. Larger strains and gradients require finer grids, with under-sampling directly leading to errors in the computed strain.

Sample automotive parts examined exhibit obvious strain gradients based on the degree of change in surface curvature. Within this thesis, a line-based grid pattern of ten lines per inch is used as shown in Figure 2.3. This line spacing provides sufficient feature frequency for typical automotive part analysis, and enables finer strain analysis than CGA systems. To measure parts with larger strain gradients, the grid spacing can be further reduced without any changes to the approach or algorithms presented in the thesis. Adaptability of the algorithms to finer grids and strain gradients is a critical advantage of the proposed system, and allows the approach to evolve with expanding forming limits and new materials.

Grid patterns can be formed from a variety of geometric features. A line-based grid is adopted to allow for ease of production, and also to simplify accurate feature detection as described in Ch. 3. Lines can be etched or printed without difficulty, and provide dense data from which to interpolate intersections to a sub-pixel resolution. The density of data is superior to isolated features such as dots, in which sub-pixel

feature locations must be interpolated from a much smaller set of pixels.

A common pattern used in computer vision is the checkerboard grid, where alternate squares of a line grid are filled in. Such grid patterns are avoided in this work for two reasons. First, the pattern is more difficult to produce reliably on a large surface when the feature (square) size becomes small. More importantly, feature detection from this pattern is asymmetric, so in the presence of imaging conditions leading to blur, feature detectors may become biased.

2.2 Close-range Optical Computer Vision

Like the human eye, optical measurement techniques use light to infer information about the world. In the context of three-dimensional reconstruction, these techniques may be broken into three categories [100], determined by the fundamental property measured:

1. **Triangulation-based techniques:** Two or more views of a feature can allow three dimensional information to be calculated (triangulated) from the two dimensional image data. Photogrammetric techniques are of interest in the context of this thesis, as opposed to active structured light or other triangulation methods. Photogrammetric triangulation is typically based on reflected intensity or colour information, as recorded by an imaging device.

2. **Interferometry:** Useful for target ranging and tracking, typically through laser interferometry [150].

3. **Time-of-flight measurement:** Principle behind laser scanners and Light Detection and Ranging (LIDAR) [172] systems.

Photogrammetry is the process of taking measurements from photographs, typically digital images. "Close-range" photogrammetry has been defined by [100] as applying to objects ranging from a few metres in size down to less than one metre, with accuracies better than 0.1mm. To reconstruct a stamped metal surface in three dimensions, a close-range photogrammetric approach is selected for the following reasons:

1. **Intensity markers:** Images provide colour or greyscale intensity information, making it possible to locate marker features on the sheet metal surface after deformation, a critical requirement for subsequent surface strain computation. Laser scanners, for example, paint surface points but do not identify the positions of intensity features on the surface.

2. **Cost:** Cameras are significantly less expensive than many alternatives such as laser scanner systems. Considering the need to locate discrete surface features, high quality or experimental laser scanners or other devices would typically be required.

3. **Non-contact:** Optical systems use light as the information carrier, which provides data sampling that is faster and denser than contact-based approaches. When physical contact is used to probe the dimensions of an object, additional fixturing is required to secure the part from displacement caused by the measurement process.

4. **Compute power:** An increasing availability of computer processing power and technology allows large volumes of data to be processed at realtime speeds, enabling an image-based solution to this class of application.

5. **Scalability:** Camera resolution may be readily modified based on the target application, allowing camera cost and compute power to be scaled according to the application needs.

6. **Operability:** An optical approach is easily understood by operators, who can naturally see the data that the system is capturing. Manual operation or intervention is easily trainable because human feedback mechanisms can be directly transferred to operation of the system.

7. **Versatility:** A manually driven optical approach removes the need for path planning or other task-specific programming, and can be quickly adapted to changing needs such as failures as encountered on a production line, for example.

CCD or CMOS cameras are typically used to capture data for optical computer vision. An obvious system parameter, that consequently drives selection of a camera

system and optics, is the intended distance between the camera and object to be measured. Given a grid density on the order of 10 lines/inch and the need to accurately detect intersections to a sub-pixel accuracy, a well posed approach is to maintain pixel size smaller than the imaged line width. An ideal grid line may be modelled as two step edges separated in distance by the line width. If imaged grid lines are smaller than the pixel size, both step edges may occur within the same pixel. Through the combination of feature detector characteristics, point spread functions of the image pixels, presence of blur and noise, non-linear lens distortions, and camera positioning error, a precise line position can become nearly inseparable from the various other effects. Even by fitting error minimizing functions to the grid lines across large regions of the image, accuracy in the estimated line intersection locations are affected by the low spatial signal to noise ratio.

To better pose the initial data collection and feature extraction problem, an imaged grid line width larger than the sensor pixel size is desired. In these imaging conditions, the effect of image capture and feature extraction error is reduced relative to the grid size. An analysis of surface strain error relative to camera target distance is provided in Section 2.2.4.

2.2.1 Review of Pinhole Camera Imaging Model

When the three-dimensional world is imaged using a typical camera, light from the outside world is projected onto the camera image plane. Although modern cameras and lens systems are complex to model precisely, simplifications and standard models allow the geometry of imaging systems to be easily understood, and form the basis of many multiple view vision techniques. The standard pinhole camera model [60] is briefly reviewed in this section. A description of many alternative camera models is provided in Sturm et al. [154].

Figure 2.4 shows a pinhole camera imaging a world point X_i. A pinhole (very small hole) at the origin of the coordinate system is the only opening through which light can pass from the $+z$ to the $-z$ halfspaces, and is in reality the camera aperture. For a light ray from point X_i to meet the image plane (at point x_i), it must pass through the camera coordinate system origin. Figure 2.4 depicts the image plane on the $+Z$ side of the origin, a convention known as a virtual image plane, which is used

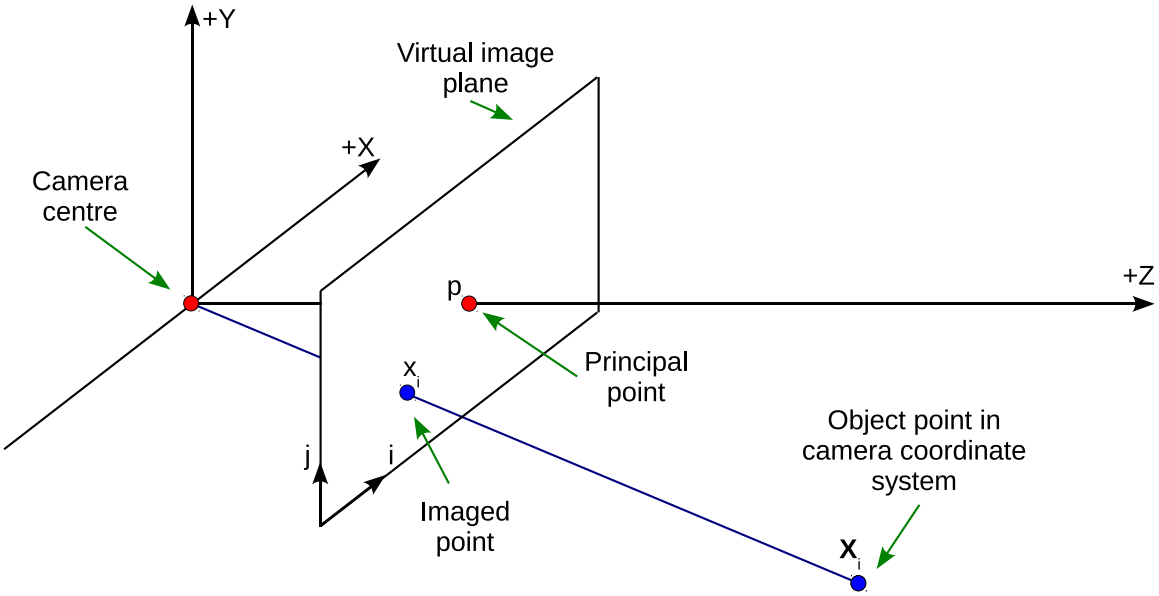


Figure 2.4: Three dimensional view of pinhole camera imaging a world point X.

to simplify analysis by avoiding projective image reversal.

To capture the internal or intrinsic parameters of the pinhole camera, a matrix K is defined as:

$$\mathbf{K} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

where u_0 and v_0 are principal point coordinates, α and β are the scaling factors, and γ defines skew between axes. The principal point coordinate offset allows the pixel origin of the image plane to be moved to a corner rather than the centre, preventing negative pixel coordinates, and following the convention used in many vision algorithms. The scaling factors account for rectangular pixel geometries, the camera focal length, and provide a scaling between the world coordinate and pixel sizes. The skew factor γ allows the image sensor pixel locations to be skewed during the manufacturing process, and is zero for most modern cameras designed with rectilinear pixels.

In Figure 2.4, the world point X_i is defined in the local camera coordinate system. In reality, world points are defined in a world coordinate system which is related to the camera system through a rigid transformation $[\mathbf{R} \quad \mathbf{t}]$. $\mathbf{R}$ is a rotation matrix between

the two systems, and $\mathbf{t}$ is the offset. The transformation captures the camera position and pose in the world frame, and provides the external or extrinsic parameters of the camera. Defining points using homogeneous coordinates, a world point $\tilde{M}$ is projected to an image point $\tilde{m}$ using the projective imaging equation:

$$\tilde{\mathbf{m}} = \mathbf{K}\,[\mathbf{R} \quad \mathbf{t}]\,\tilde{\mathbf{M}} \tag{2.5}$$

In real camera systems, a lens is used to gather more light than a pinhole can provide. Lenses produce distortions and aberrations (discussed further in Chapter 6), but a more significant affect in this work is the associated focal depth of field.

2.2.2 Depth of Field

Depth of field (DOF) refers to the range of distances that an object may move along the camera axis and still be considered in focus. To better define what is meant by "in focus", a photography term known as the acceptable circle of confusion (CoC) may be used. Figure 2.5 illustrates the DOF concept. Rays emanating from the "ideal focus point" point meet at a single point on the camera image plane, and are therefore in focus. The remaining points (shown in red and blue) have focal distances offset from the image plane, and so appear on the image sensor as diffuse. The diffuse region on the image plane is by definition the circle of confusion. For a fixed CoC size, bounding distances can be defined in the object space for which diffuseness is no larger than the CoC. The range of distances in object space that meet a specific CoC size is defined as the DOF.

The CoC size that is considered acceptable depends strictly on the application. Considering human perception, as long as the CoC is smaller than or on the order of one image sensor pixel size, the image appears sharp [100]. Assuming a thin lens without aberrations, and objects near the optical axis, the bounding CoC object distances may be calculated using similar triangles and the geometries illustrated in Figures 2.6(a) and 2.6(b):

$$\frac{\frac{c}{2}}{\frac{d}{2}} = \frac{V_C - V}{V_C} \tag{2.6}$$

$$\frac{\frac{c}{2}}{\frac{d}{2}} = \frac{V - V_F}{V_F} \tag{2.7}$$

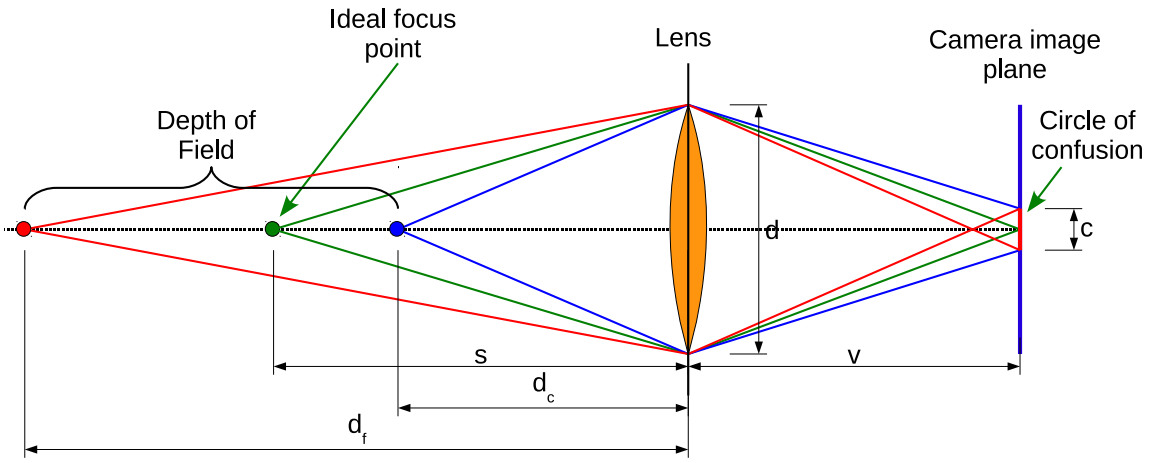


Figure 2.5: Depth of field illustration. The ideal focus point (shown in green) is in focus on the image plane. The remaining two points (shown in red and blue) are out of focus, and the size of this defocus on the image plane forms the circle of confusion. The range of image target distances that produce a small circle of confusion (1 pixel in this case) define the depth of field (DOF) for the specified aperture setting.

Considering the standard aperture F-stop equation of:

$$N = f/d \tag{2.8}$$

where N is the F-stop number, f is the focal length and d is the circular lens diameter at the given aperture, Equations 2.6 and 2.7 may be rewritten:

$$\frac{Nc}{f} = \frac{V_C - V}{V_C} \tag{2.9}$$

$$V_C = \frac{V}{1 - \frac{Nc}{f}} \tag{2.10}$$

$$\frac{Nc}{f} = \frac{V - V_F}{V_F} \tag{2.11}$$

$$V_F = \frac{V}{\frac{Nc}{f} + 1} \tag{2.12}$$

To relate the distances V_C and V_F to the world space, the thin lens equation can be applied:

$$\frac{1}{s} + \frac{1}{V} = \frac{1}{f} \tag{2.13}$$

where s is the lens to object distance, V is the lens to image plane distance, and f is the focal length of the system, defined as the distance from lens to focal point when collimated rays enter the lens.

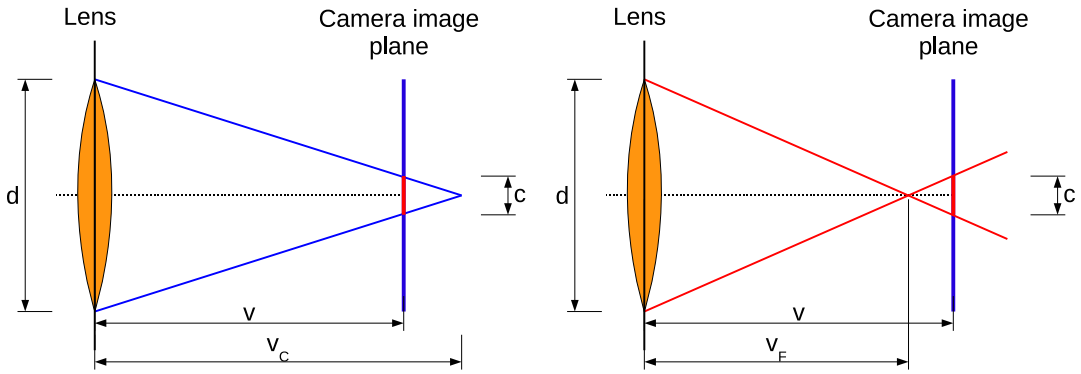


Figure 2.6: (a) Geometry for calculation of object space close-range bound (V_C) on CoC; (b) Geometry for calculation of object space far-range bound (V_F) on CoC.

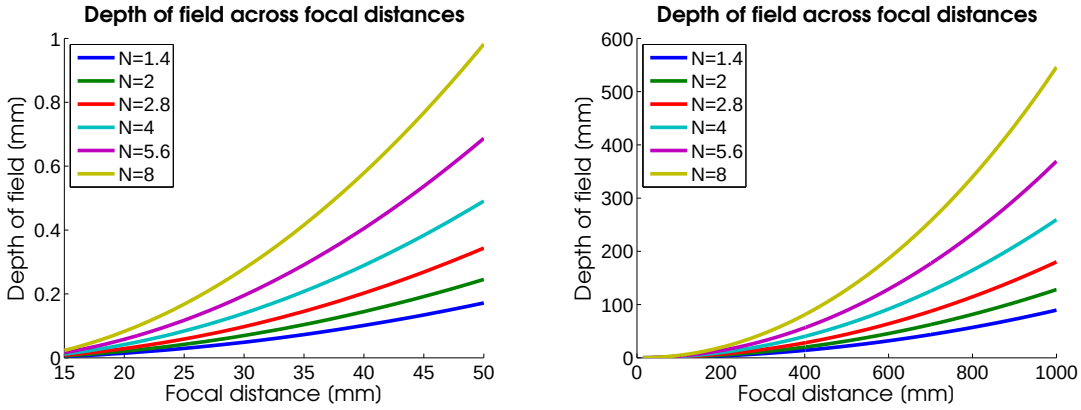
The DOF distance limits d_c and d_f, as shown in Figure 2.5, are then expressed as:

$$d_c = \frac{f^2 s}{f^2 + cN(s-f)} \qquad (2.14) \qquad\qquad d_f = \frac{f^2 s}{f^2 - cN(s-f)} \qquad (2.15)$$

The depth of field is therefore expressed as:

$$\begin{aligned} DOF &= d_f - d_c \\ &= \frac{2f^2 s(s-f)cN}{f^4 - c^2 N^2 (s-f)^2} \end{aligned} \qquad (2.16)$$

Figure 2.7 shows the DOF range across varying focal distances, at multiple aperture f-settings. In the figure, the camera focal length is 12mm, and the limiting circle of confusion size is set to the pixel size of a sample camera ($4.65\mu m$). The DOF is small at close range, meaning that deviation from a perpendicular angle between the camera and a flat surface quickly results in DOF blur. Parts that are curved beyond the DOF limits within the frame also encounter blur. Increasing the f-number (decreasing aperture size) increases the DOF, but results in less light collection and therefore longer exposure times. Longer exposures decrease the available frame rate, but more significantly increase the chance of motion blur when the camera is manually manipulated. There is a trade-off between aperture setting and DOF, which



(a) (b)

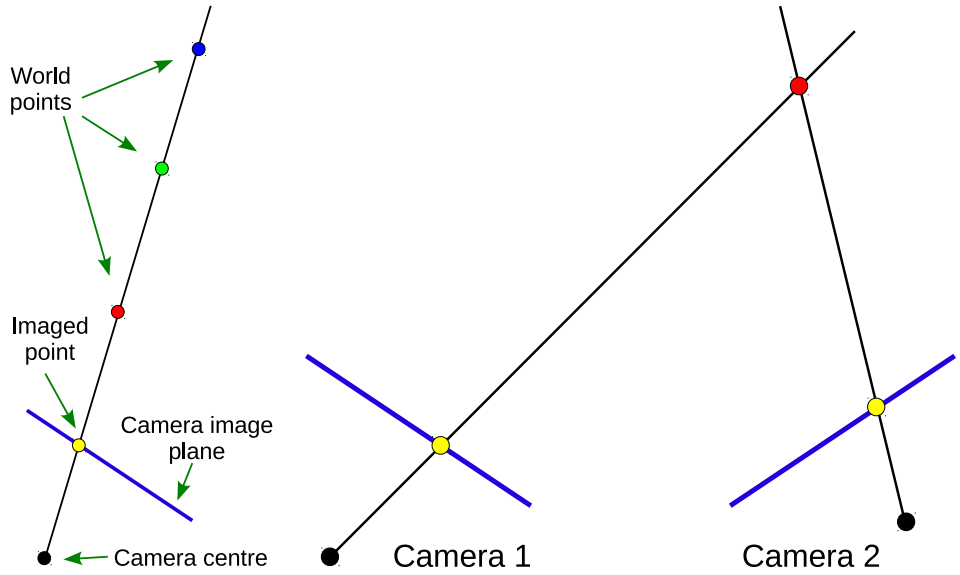
Figure 2.7: Variation in depth of field with varying focal distance and f-stop.

plays a role in the ability to focus the camera when oblique viewing angles are used for multiple view triangulation.

2.2.3 Multiple View Triangulation

When the three-dimensional world is projected onto a two-dimensional camera image plane, information is lost about the depth of the imaged point from the camera. As shown in Figure 2.8(a), points lying on a common ray from the camera centre will project to the same point on the image plane. Additional camera views from different positions may be used to determine the unique world point location (with appropriate camera positioning and calibration), as shown in Figure 2.8(b). From distinct camera positions viewing a single point in 3D space, assuming that the point can be exactly located in both images, then the point location can be reconstructed.

A variety of well known techniques are available to perform 3D reconstruction from multiple camera images. Good descriptions of reconstruction techniques are available in books by Hartley et al. [60] and Faugeras et al. [39]. Reconstruction of structure typically involves epipolar geometry, with imaged feature coordinates across images related by the essential matrix $\mathbf{E}$ when the camera calibration is known, or the fundamental matrix $\mathbf{F}$ in the case of uncalibrated cameras [61, 60]. From corresponding point locations across two or more images, a linear triangulation may be performed to



(a) (b)

Figure 2.8: (a) Points lying on a single ray from the camera centre. Distance along the ray cannot be determined from the single imaged point, although the ray direction can be ascertained; (b) Two camera views of a world point from different positions allow the point location to be triangulated.

locate the original point in 3D space using, for example, the Direct Linear Transform proposed by Abdel-Aziz et al. [1]. Iterative non-linear reconstruction techniques are well known and typically formulated as optimization problems, as described in [60]. In the presence of noisy image feature locations and camera parameters, bundle adjustment is used to refine not only the estimated 3D reconstruction points, but also the camera matrices to minimize error between expected and measured image feature locations [165, 98].

Presence of a grid (and therefore distinct conformal features) on the sheet metal surface avoids a substantial difficulty often encountered in three-dimensional reconstruction, that of point correspondence between images. Multiple view correspondence is a problem which is still actively researched and unsolved in many general imaging conditions [12].

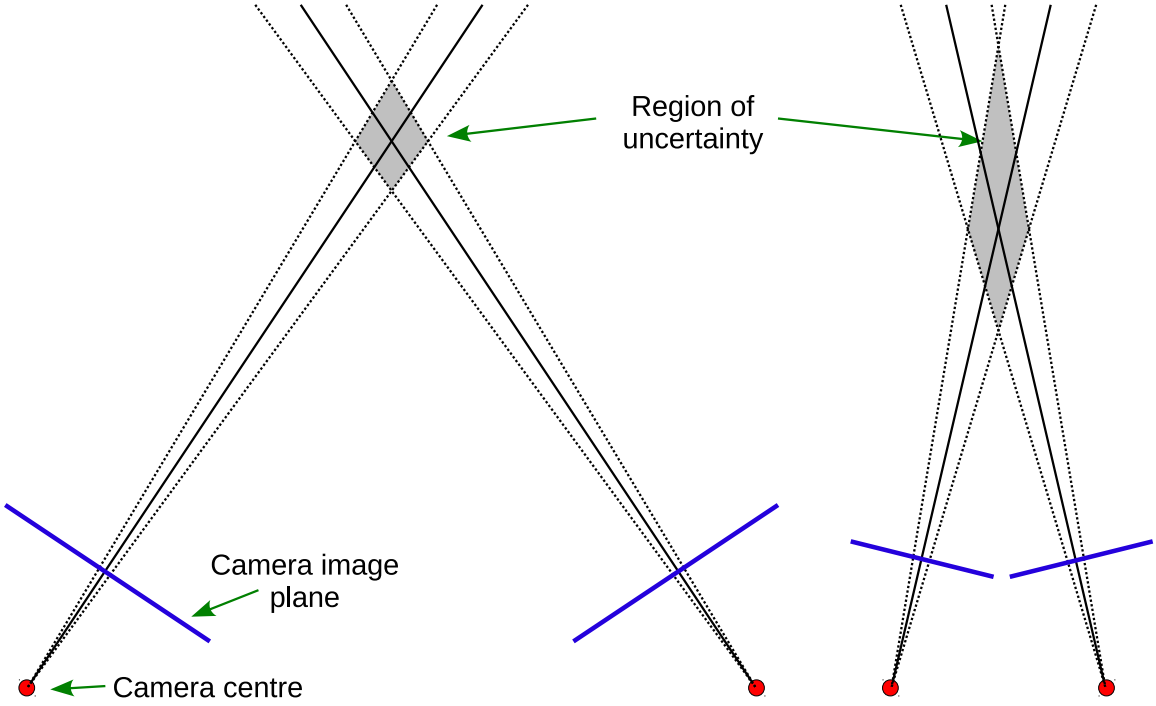


Figure 2.9: Triangulation error region given image point measurement uncertainty. A narrow baseline increases the triangulated depth uncertainty.

2.2.4 Surface Strain Error and Imaging Distance

Error or uncertainty in measured image coordinates leads to error in the triangulated intersection, as illustrated by Figure 2.9. The size and shape of the error region depends on the magnitude of measurement uncertainty, the angle between camera principal axes, and the camera positions (Figure 2.9 shows symmetric camera positioning).

The distance between camera and target affects system accuracy in terms of sensitivity to feature extraction and camera positioning errors, and also through increased computing requirements to process numerous small field-of-view images at close range. Optimization techniques such as bundle adjustment attempt to minimize errors in camera matrices and reconstructed position while considering all data simultaneously, but reducing initial error sources leads to a more robust measurement system. In order to analyze the effect of image measurement and positioning errors

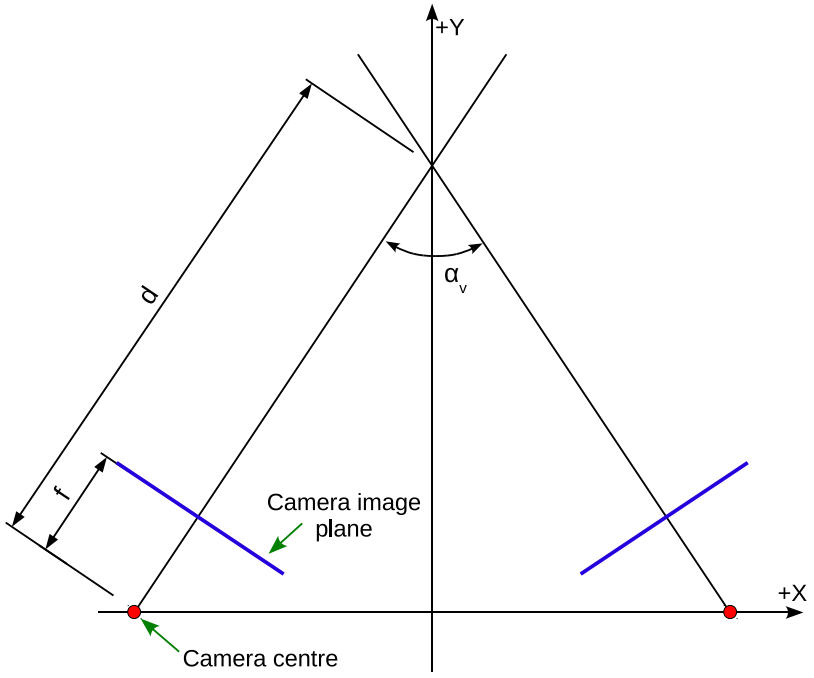


Figure 2.10: Geometry of camera triangulation simulation.

in the system relative to imaging distance, triangulation error is considered.

A two-dimensional model of a camera triangulation system is shown in Figure 2.10. The measured feature locations in the images are chosen as the source of error for analysis, which have a similar effect to camera positioning and pose errors when considering point triangulation. Camera position and pose errors can be analyzed separately, but to illustrate the reasoning behind the close-range vision approach, pixel measurement error is sufficient.

A Monte Carlo analysis is performed by adding error to the detected pixel locations in each of the two camera image planes in Figure 2.10. An erroneous point is triangulated, and the distance from this point to the ideal (error free) point is recorded.

The error level in detected feature location is selected from a normal distribution with $\sigma = \frac{1}{10}$ pixels. The error is intended to represent both image measurement and camera position/pose errors, and in real-world systems depends on the hardware and system used. The level of error selected is reasonable for current systems, and is meant to be instructive. Experimental analyses of edge detection position error in

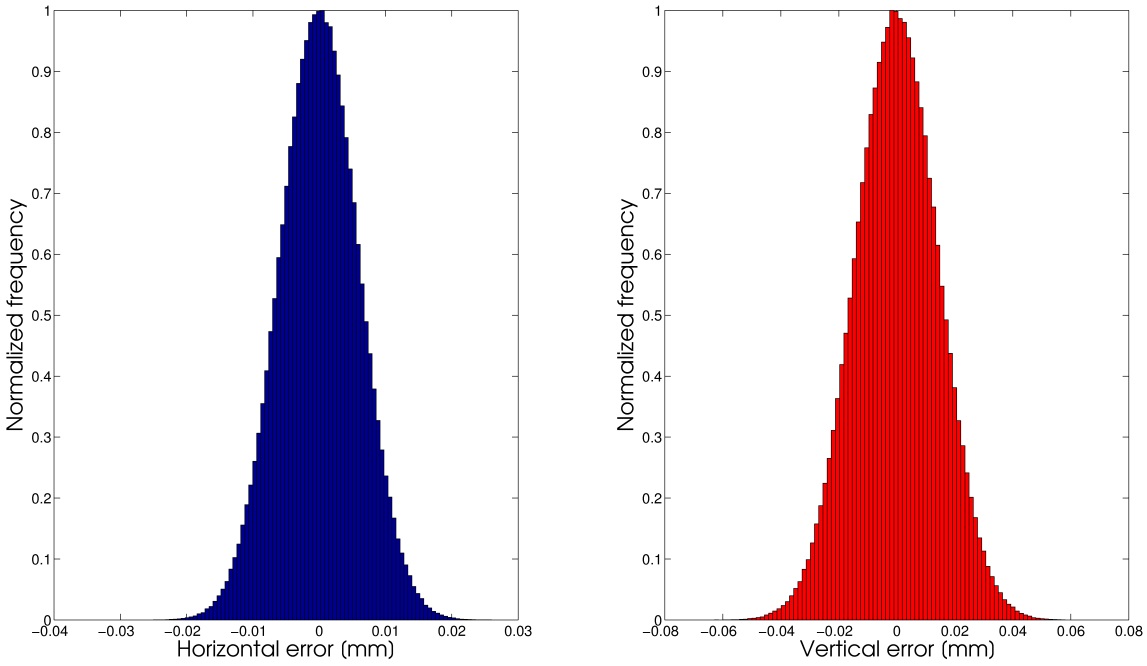


Figure 2.11: Histogram of horizontal ($\pm X$) and vertical ($\pm Y$) error distribution from 1 million simulation points. Image pixel errors were normally distributed with $\sigma = \frac{1}{10}$ pixels, camera vergence angle was $\alpha_v = 45°$, and camera imaging distance was 20 cm. The horizontal and vertical reconstruction errors have different standard deviations, caused by the vergence angle of the cameras.

the presence of noise and varying angle are available in the literature, and support the error level used in this analysis. Steger et al. [148] obtained edge position error with maximum standard deviation of $\approx \frac{1}{30}$ pixels, while Devernay et al. [30] measured edge position error with standard deviation $< \frac{1}{10}$ pixels. These error levels support the choice of $\frac{1}{10}$ pixel error standard deviation for simulation.

Figure 2.11 shows histograms of the reconstruction error from simulation of one million data points, where the camera imaging distance is set to 20 cm and the vergence angle $\alpha_v = 45°$. Horizontal error is along the X axis, and vertical error is in the Y axis direction, as defined by Figure 2.10. The vertical error has a larger standard deviation compared with horizontal error, which is a result of increased depth uncertainty caused by the non-orthogonal camera vergence angle (as expected from Figure 2.9).

Results from a sweep of camera to target distances are presented in Figures 2.12

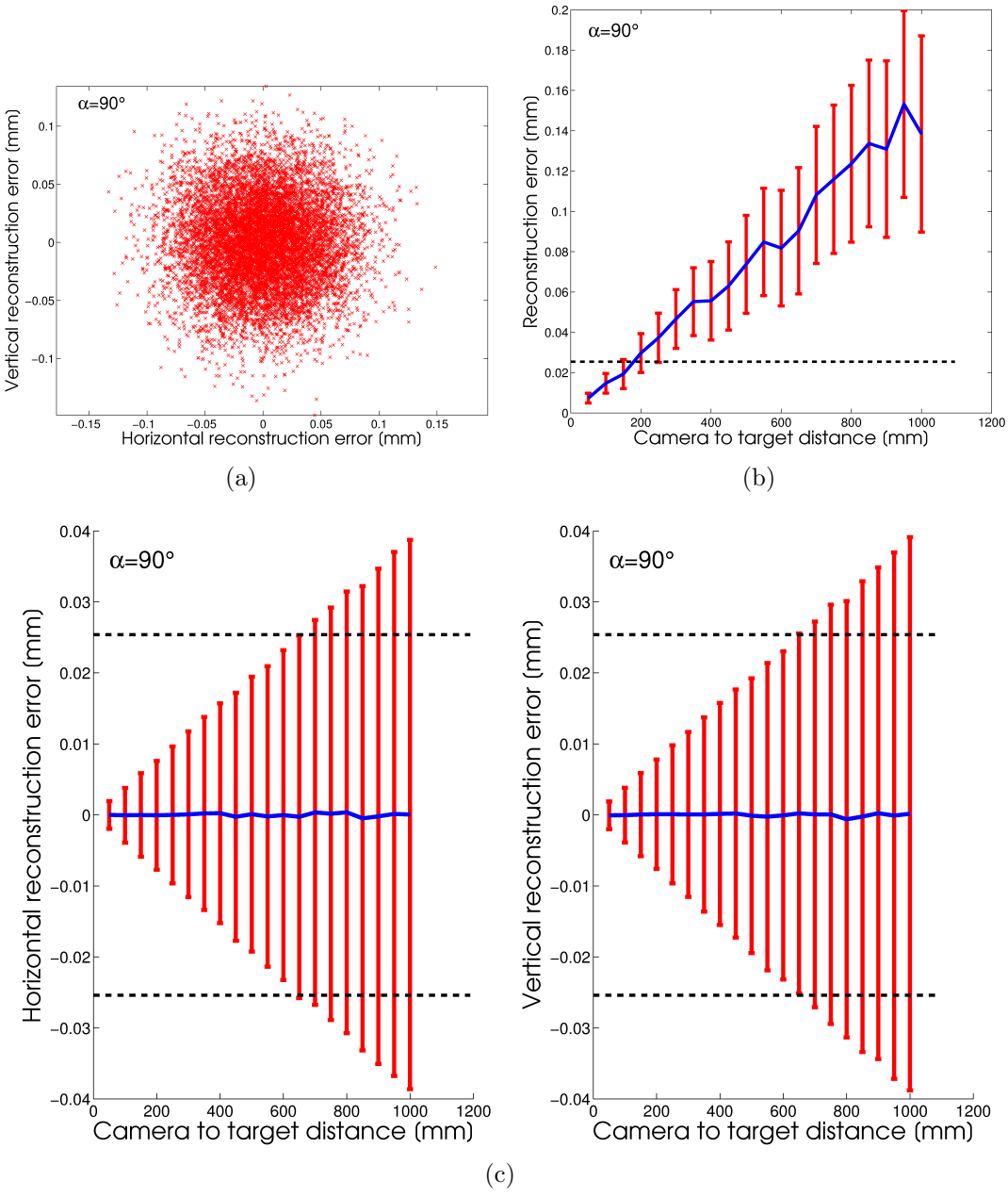


Figure 2.12: Monte Carlo simulation results from a sweep of camera distance from triangulation point, using 10000 trials per data point with the system geometry shown in Figure 2.10, and a camera vergence angle of $\alpha_v = 90°$. (a) Point cloud of triangulated points using camera to target distance of 1 m. (b) Magnitude of total error vector, with black horizontal line indicating 1% error margin for strain reconstruction. (c) Horizontal ($\pm X$) and vertical ($\pm Y$) error levels, with black horizontal line indicating 1% error margin for strain reconstruction.

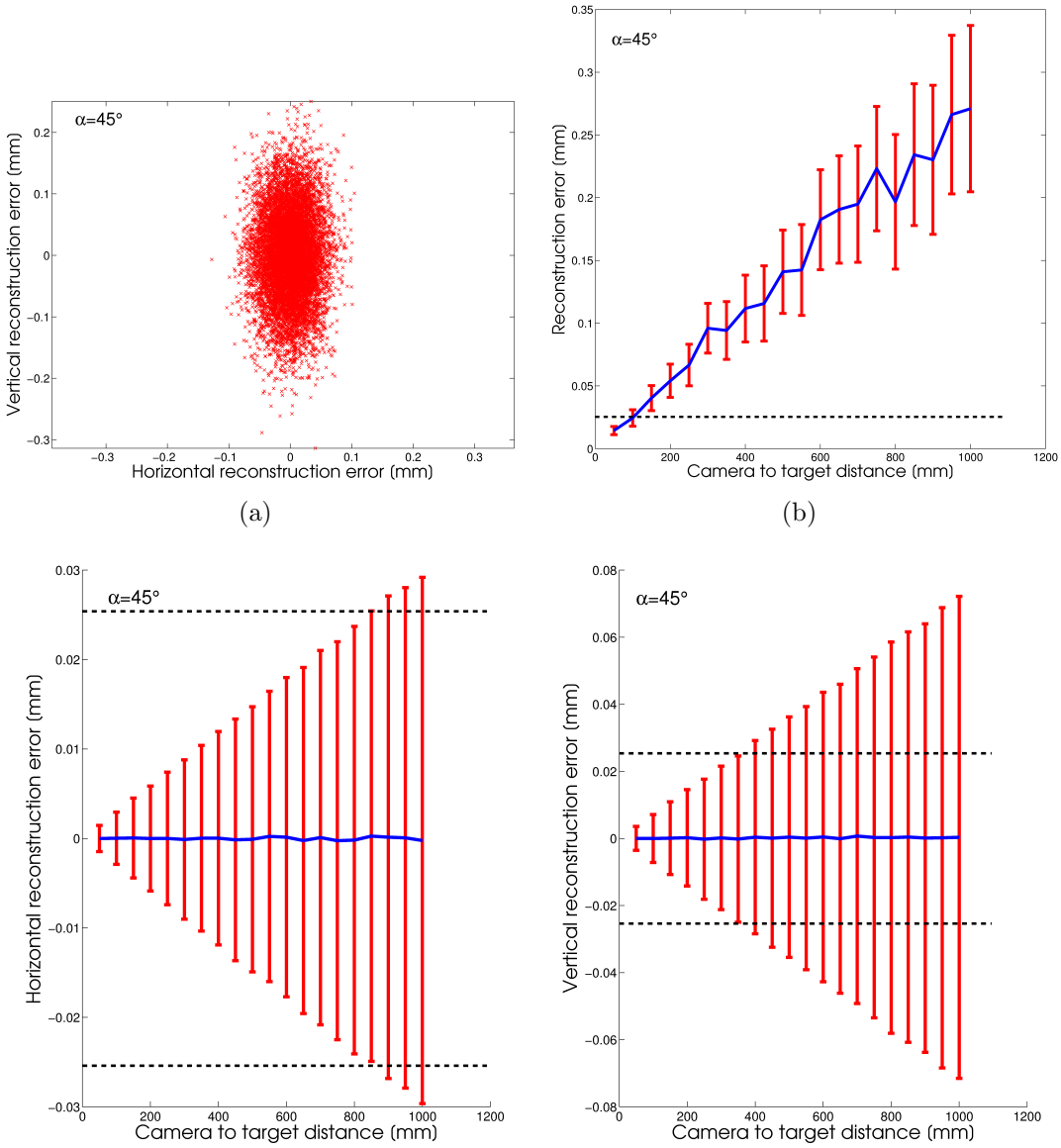
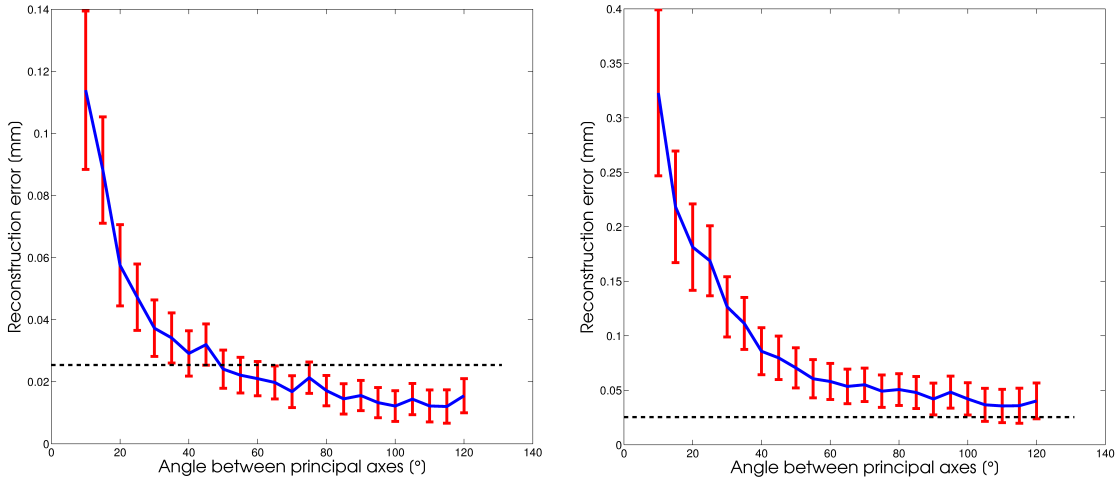


Figure 2.13: Monte Carlo simulation results from a sweep of camera distance from triangulation point, using 10000 trials per data point with the system geometry shown in Figure 2.10, and a camera vergence angle of $\alpha_v = 45°$. (a) Point cloud of triangulated points using camera to target distance of 1 m. (b) Magnitude of total error vector, with black horizontal line indicating 1% error margin for strain reconstruction. (c) Horizontal ($\pm X$) and vertical ($\pm Y$) error levels, with black horizontal line indicating 1% error margin for strain reconstruction.



(a) (b)

Figure 2.14: Monte Carlo simulation results from a sweep of camera vergence angle α_v, using 10000 trials per data point with the system geometry shown in Figure 2.10. (a) Magnitude of total error vector with camera to target distance of 10cm. Black horizontal line indicates 1% error margin for strain reconstruction. (b) Magnitude of total error vector with camera to target distance of 30cm. Black horizontal line indicates 1% error margin for strain reconstruction.

and 2.13, with the only difference between the two being the camera vergence angle. Horizontal dotted lines on the graphs indicate an error level of 1% of the grid spacing ($\approx 25\,\mu m$ at 10 grid lines/inch), and therefore approximately correspond to a strain error of 1%, which is the desired error threshold. Where a part surface is parallel to the camera plane, horizontal error is the prime contributor to strain errors. In reality the camera is imaging curved parts and is held at an angle to the part surface for triangulation, so vertical (depth) error is also significant. In terms of part dimensional analysis, error in both dimensions is equally important. As can be seen from the results, the error budget is quickly exceeded as the camera to target distance increases, especially when the vergence angle is smaller (such as $\alpha_v = 45°$ in Figure 2.13). Given the DOF constraints at close range, large camera vergence angles are not practical because most of the field of view becomes blurred. Vergence angles on the order of 45° are practical, as tested using real cameras.

A sweep of the camera vergence angle is shown in Figure 2.14 for two camera to target distances, 10 cm and 30 cm. As the camera moves further from the target,

extreme vergence angles are required to bring the error level near the 1% error margin (shown as block dotted lines), which are not practical given DOF constraints at close range.

Within this work, the image acquisition distance between camera and target is smaller than used by commercial systems. Imaging from a larger distance reduces the computational workload imposed by image feature extraction and interframe motion tracking across numerous frames, and allows unique feature points to be identified for grid registration across image frames. Larger imaging distances, however, increase the effect of measurement errors on the reconstruction results.

The two dimensional simulation provides an overview of the error margins present. In reality, each camera has two-dimensional image plane error, and the reconstruction contains three-dimensional error. The increased dimensionality causes error magnitudes in reconstruction to become larger, while additional views (more than two) of a feature point reduce the error. The worst case number of views to enable triangulation is two, but a well scanned part should contain additional views of each surface grid intersection. The level of error reduction through multiple views depends on relative camera positions and final optimizations, so controlling error through imaging distance is important to ensure accurate reconstruction. The simulations presented in this section reinforce the idea that close-range imaging is a critical factor for reconstruction within acceptable error levels.

Close-range imaging, on the order of 10-20 cm from a part surface, introduces two key difficulties in the data acquisition process.

1. **DOF blur:** Close-range imaging leads to a narrow DOF. The combination of human camera manipulation, curved part surfaces, and the requirement for a baseline distance between camera views invariably leads to DOF blur. Processing algorithms must be capable of operating in the presence of blur.

2. **Data density and computation load:** Close range leads to a camera field of view that sees only a portion of most automotive parts. The camera must therefore be moved to image the entire surface, leading to video sequences in which motion must be tracked to correlate grid points between views. This is in contrast to commercial approaches that take few images from a distance, where global identifiers are used to correlate feature points, and where image

processing occurs on a small set of data. Computation is further complicated by the presence of DOF blur in the images, which may necessitate more complex image processing approaches.

In contrast to the commercial optical strain analysis systems described in Section 1.2.4, the approach taken by this thesis involves imaging at a closer range. This approach is designed to better pose the data used for reconstruction, and to make the system scalable to much stronger strain gradients through imaging of a finer grid. The close-range imaging conditions introduce challenges not experienced by the commercial systems, such as DOF blur and inability to capture the entire part within a single image. These challenges are addressed throughout the remainder of this thesis. Although the algorithms are intended for the challenges of close-range imaging, they operate without modification at larger distances such as those used by commercial systems, where close-range effects are not significant.

2.3 Monocular Vision using Coordinate Measuring Machine

To control error levels, a typical imaging distance of less than 20 cm is selected for data acquisition (although much larger distances are supported by the algorithms without modification). At such distances the camera field of view does not encompass a complete part surface, so must be moved across the surface to image not just the completed part, but each grid line intersection from at least two views. To enable subsequent triangulation, a method is required to measure the camera position and pose at the instant that each frame is captured. An articulated arm Coordinate Measuring Machine (CMM) is selected to provide this measurement. The articulated arm is passive, so a human operator must physically move the camera while the CMM simultaneously measures the motion. Manual manipulation of the camera is chosen to eliminate path programming required for a robotic device, and to simplify mobility in an industrial environment.

Given that a human must manipulate the camera, a single camera mounted to the CMM is used. Stereo or trinocular vision systems are not practical for manual

manipulation at close range because of the narrow camera DOF. Given a required vergence angle between multiple view cameras, a human operator is required to maintain reasonable focus using feedback from, for example, real-time video output display. Maintaining focus on more than one camera is not practical for most human operators, so a monocular approach is used, obtaining temporal (as opposed to spatial) stereo by imaging each grid line intersection in multiple video frames from different positions. Further reasons for use of a single camera include reduced cost, simplified calibration, and mechanical robustness of the mount connecting the camera to the CMM tool point. A major consequence of temporal stereo is that images are taken at different times, so the target must be static across the video sequence. This assumption is safe for sheet metal parts (before or after forming), but the approach is not suitable to imaging of a part during an active deformation process.

2.3.1 CMM Details

Figure 2.15: Scanning automotive heat shield surface using single camera mounted to FARO arm.

Figure 2.16: Scanning automotive heat shield surface using single camera mounted to FARO arm.

The CMM used for this work is a Gold FaroArm, produced by FARO Technologies Inc. The command interface was provided to the research group by FARO, allowing custom software to command, configure, and receive data from the arm. Standard use of the FaroArm is for touch-probe based measurement, with a probe attached to the toolpoint. In typical operation, a technician manipulates the probe tip to touch the surface requiring measurement, and then presses a button to latch and store the position.

For this work the touch probe is removed from the FaroArm toolpoint, and replaced with a custom camera adapter. Latching of arm position data is synchronized with the camera shutter, as described in Section 2.3.4. Figures 2.15 and 2.16 show the camera-equipped FARO arm being used to scan a large truck exhaust system heat shield.

Tests performed on the Gold FaroArm in June 2005 obtained an accuracy standard deviation of 27 μm with a maximum measured error of 108 μm. Current generation FaroArm devices (FARO PowerGage) are accurate to $\approx 5\,\mu m (0.0002")$ [69].

Table 2.1: CMM Arm Specifications

Manufacturer	FARO Technologies Inc.
Model	Gold FaroArm
Single point accuracy (2 σ)	51 μm
Measurement radius	7 feet
External position latch trigger capability	Two auxiliary TTL trigger ports
Output	RS-232 serial

Table 2.2: Camera Specifications

Manufacturer	Point Grey Research Inc.
Model	Dragonfly
Signal and power interface	Firewire (IIDC-1394)
Resolution	1024×768
Pixel depth	8-bits/pixel
Pixel size	$4.85\,\mu m(H) \times 4.85\,\mu m(V)$
Image sensor	Sony ICX204 ($\frac{1}{3}$" HAD CCD)
Available frame rates	1.875, 3.75, 7.5, 15 fps
Shutter speeds	$\frac{1}{6000}$ s up to 60 s
Signal to noise ratio	> 50 dB at minimum gain
Dimensions	6.35 cm $\times$ 5.08 cm $\times$ 1.32 cm (without lens)

2.3.2 Camera Details

The camera used for this work is a Dragonfly from Point Grey Research, as detailed in Table 2.2 [132].

A 12 mm Cosmicar/Pentax lens with low distortion for metallurgy is attached to the camera. A 1 mm extension ring placed between the lens and camera CS-mount reduces the minimum object focus distance for close range imaging. The camera is allowed to perform automatic exposure control, so bright lighting in the vicinity of the camera helps to reduce the exposure time, which leads to less motion blur in the image.

The camera is monochrome to avoid complications in subpixel grid line detection when a colour filter pattern is placed over the CCD chip. Colour image sensors typically employ a primary colour mosaic filter, often in a Bayer pattern, to produce the apparent colour image. Neighbouring pixels consequently respond to different light wavelengths, which complicates accurate subpixel feature measurement. If a grid

pattern of specific colour is of interest, it suffices to place a filter on the monochrome camera to enhance grid contrast without adding non-uniform colour response to the sensor.

Considering the challenges of narrow DOF in close-range imaging, an obvious strategy is to use an auto-focus lens assembly. The problem with this approach is the changing camera calibration as the lens focus is modified. Calibration of automated lenses has been studied, for example in [175, 174]. For accurate metrology, a fixed parameter camera (fixed focus and aperture setting) is easier to calibrate, and provides confidence in the camera model used during triangulation and point reconstruction. Regardless of the constantly changing calibration, auto-focus lenses do not solve the DOF blur problem. Mean focus error may be minimized, but the DOF remains small at close range, resulting in blur when a part surface curves, or when the camera is held at an oblique angle to the surface.

2.3.3 Camera Trajectory Constraints

There are three primary constraints on the camera operator while scanning sheet metal surfaces with the proposed system.

1. **Focus:** The camera should be maintained at an imaging distance where most of the image appears in focus. Some outlier frames invariably occur where the entire image is DOF blurred as the operator corrects the camera position, but these instances should be minimized because they lead to frames where triangulation data is not obtained. Feedback to the operator is provided through display of video data from the camera.

2. **Multiple views:** To perform three-dimensional reconstruction of the grid line intersections, each intersection must be imaged from more than one camera position. Variety in the camera positions reduces error in the reconstruction, so it is up to the operator to image the entire surface from a variety of camera poses.

3. **Speed of motion:** Motion blur in images, caused by translation and/or rotation of the camera during a single frame integration time, should be avoided

because of increased uncertainty and error in the extracted line intersection locations. This form of blur is avoided by moving the camera smoothly and slowly (less than approximately one cm per second for the Dragonfly camera). Bright lighting near the imaged part decreases the shutter integration time, so reduces the tendency for motion blur.

A human operator cannot operate the system perfectly, so later stages of processing are used to judge the reliability of specific data frames, rejecting those that contain unacceptable error.

2.3.4 Camera Synchronization

The CMM-mounted camera is manipulated by a human operator, so is in nearly constant motion. The operator cannot freeze the camera motion for each image frame, so the camera sensor integration and CMM latch triggers must be synchronized. In this way, the CMM captures a position/pose data point simultaneously with the camera shutter being triggered to begin an integration.

Both the FARO arm and camera provide facilities for external triggering. As shown in Figure 2.17, a microcontroller is used to drive the simultaneous pulse signals to the CMM and camera. The camera is grounded through the computer FireWire bus, while the CMM is grounded to the wall power outlet. To avoid potential ground loop problems, optical isolators are placed between the microcontroller and external devices. Data from the camera (over FireWire) and from the CMM (serial port) are both buffered. To ensure that the data received from the two devices is correctly aligned temporally, a "go flag" is driven by the controlling computer. Prior to asserting the flag, the PC flushes all camera and CMM receive buffers. Custom software on the computer orchestrates buffer control, data reception, and also provides video feedback on the computer screen so that the operator can maintain camera focus. On startup, the software configures the CMM and camera to use the proper trigger and capture modes.

To simplify hardware interfacing, the "go flag" signal in Figure 2.17 is not implemented through a separate interface to the computer, but instead uses a General Purpose I/O (GPIO) pin on the camera. The GPIO level is controlled using commands sent through the FireWire bus.

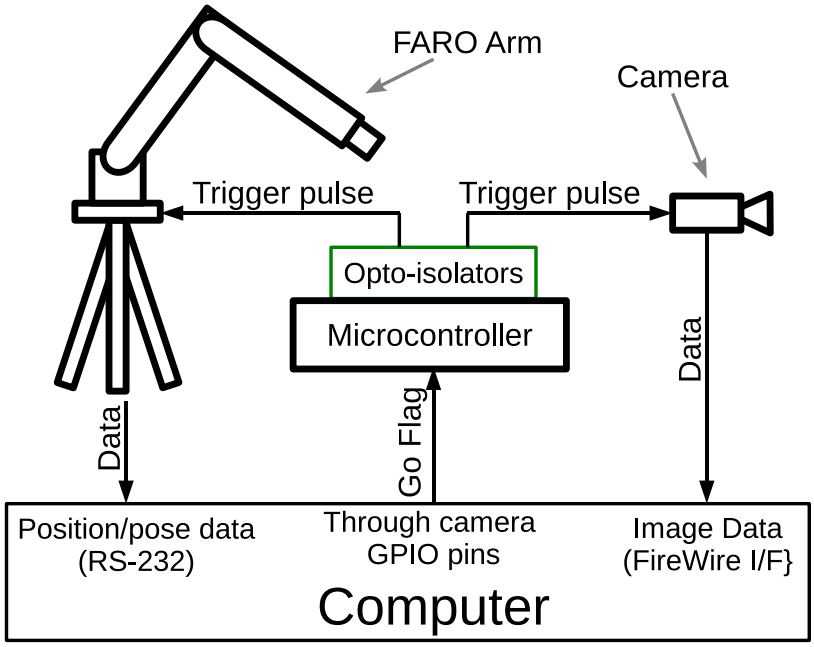


Figure 2.17: Synchronization and data flows between the controlling computer, FARO arm, and camera.

2.3.5 System Data Flows

Operation of the vision system is divided into two primary modes, with Figures 2.18 and 2.19 showing the primary processing stages in each of these. The first is system calibration, which involves scanning of a flat grid surface to perform both camera and hand-eye calibration, as described in Ch. 6. This mode is used when camera parameters are changed (such as focus or aperture), or when the camera mount to the FARO arm is adjusted. The second mode is used for part scanning and reconstruction, and is effectively a subset of the operations performed in the calibration phase.

The blocks shown in Figures 2.18 and 2.19 are described in the remaining chapters of this thesis. Ch. 3 and 4 cover the block labelled "GPU-accelerated grid intersection detection". Ch. 5 describes a grid motion tracking algorithm contained within the "Pre-processing for camera calibration" and "3D grid reconstruction" blocks. Ch. 6 describes the "Camera/hand-eye calibration" and a portion of the "Pre-processing for camera calibration" blocks.

Figure 2.18: Calibration data flow with high level processing steps (dotted box indicates GPU accelerated component).

Figure 2.19: Part processing data flow with high level processing steps (dotted box indicates GPU accelerated component).

2.4 Summary of this Chapter

This chapter has described an overall system design and resultant data flows developed to meet the thesis objectives. An overview of surface strain analysis and 3D reconstruction techniques was presented, followed by an analysis of the camera-to-target imaging distance required to obtain acceptable measurement accuracy. A description of the system design including camera synchronization and data flows was presented.

Chapter 3

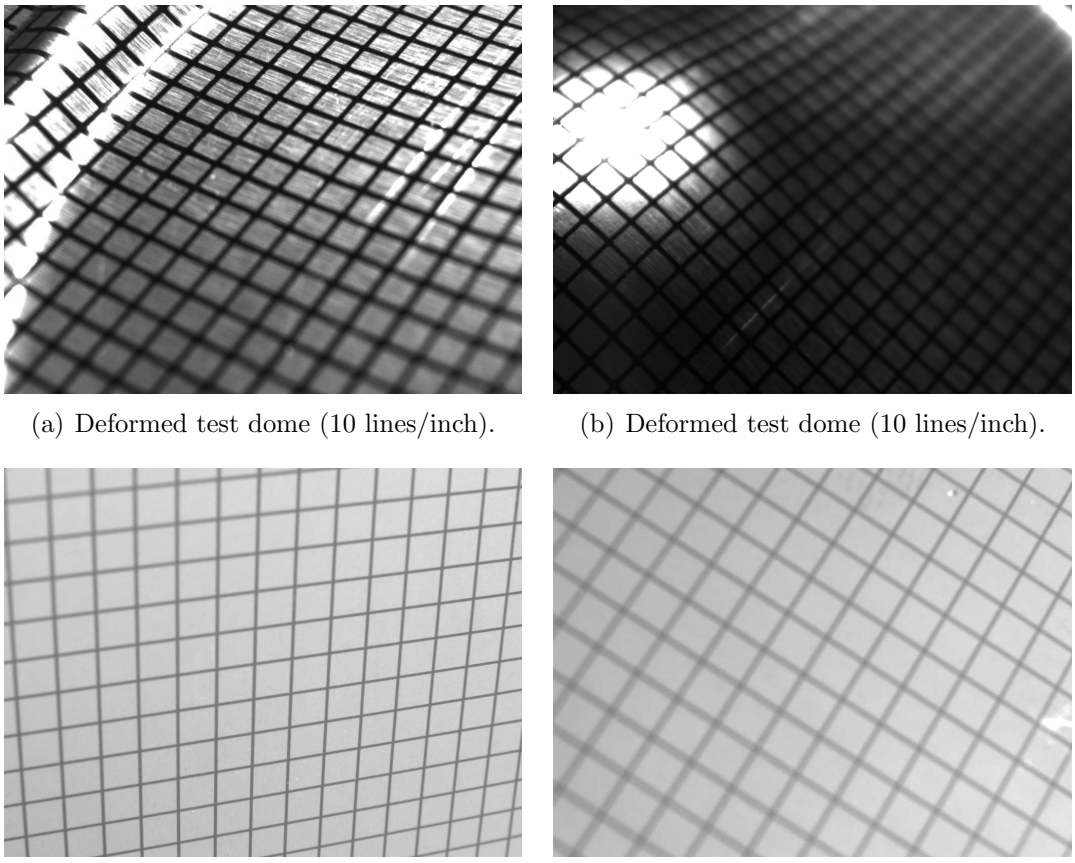
Grid Line Intersection Measurement

In the surface scanning and calibration data flows described in the previous chapter, a line-based grid forms the feature set to be measured from the close-range images (described by Section 2.1.3). Intersections of the grid lines are suited for extraction because they are well defined spatially, and form a regular structure from which the surface strain and part geometries can be reconstructed. Detection and sub-pixel measurement of grid line intersections in the presence of DOF blur is the topic of this chapter. Suitable classes of feature detectors are reviewed, followed by description of the scale-space ridge extraction and parabola fitting solution. Contributions of the chapter include: 1. Selection of scale-space ridge extraction for close-range DOF blurred imaging to enable multi-scale (varying degrees of blur) extraction of dense ridge data; and 2. Consideration of computational intensity of the scale-space approach in the context of CPU implementation.

Please note that some material in this chapter has been previously published in the Journal of Physics: Conference Series [74][1], Journal of Computer Aided Design and Applications [73], CCECE Conferences in 2007 [71] and 2008 [72], and other aspects have been submitted to the IEEE Transactions on Instrumentation and Measurement.

Figure 3.1 shows four sample images of gridded metal surfaces. In the context of feature detection for close range metallurgical imaging, five attributes of the imaging

[1]Published under licence in Journal of Physics: Conference Series by IOP Publishing Ltd.



(a) Deformed test dome (10 lines/inch).

(b) Deformed test dome (10 lines/inch).

(c) Calibration plane (10 lines/inch).

(d) Calibration plane (10 lines/inch).

Figure 3.1: Sample images of gridded parts: (a) and (b) are from a deformed metal dome sample, (c) and (d) are from a planar pattern. All images exhibit depth-of-field blur.

conditions are significant, as visible in the samples.

1. **Scale of features:** As distance changes between the camera and gridded surface, the size of grid lines and spacing between them changes relative to the image pixel size.

2. **Depth-of-field blur:** The combination of close-range imaging and human operation leads to inevitable depth-of-field blur, usually in portions of the image, but often across an entire frame. Feature detection should operate regardless of partial blur, and when excessive blur does exist, presence of the intersection should be detected to enable interframe tracking.

3. **Grid production method:** Multiple methods of grid production are available including photochemical printing or etching, and silkscreen printing. For adaptability to various applications, grid intersections should be detected without significant algorithm adjustment between grid production methods, colours, or line thicknesses.

4. **Lighting:** In practical industrial environments, uniform or non-specular lighting is difficult to guarantee. A robust feature detector must operate regardless of varying lighting conditions as shadows and light intensities around the metal surface change. Given the reflective nature of many sheet metal surfaces, specular reflections are expected from point source lighting, and should not be allowed to prevent feature extraction unless the image pixels are saturated (for example the left side of Figure 3.1(b)).

5. **Varying magnitude of curvature:** Grid line curvature depends on the magnitude of deformation in the sheet metal. Even with undeformed planar surfaces, lens distortions can lead to non-linear appearance of lines, so straight-line or fixed curvature approaches to detection are not suitable. Useful feature extraction must operate reliably regardless of unknown and continuously varying curvature in the imaged grid lines.

3.1 Existing Work

Measurement of line-grid intersections has only sparse coverage in the literature, especially when the lines are allowed to curve variably as in a metal deformation process. One published algorithm for line intersection measurement in nerve cell images [138] identifies arbitrarily curved line intersections, but requires perfectly skeletonized binary images of the lines. In close-range vision the challenge is to consistently detect lines regardless of scale or blur, so reliably generating a skeletonized form is the difficult part of the problem.

One form of grid that is considered extensively in the literature is a checkerboard (or chessboard) pattern, where alternating grid squares are filled in. This style of grid is often used as a target for camera calibration, with most techniques based on

corner detection (commonly the Harris corner detector [59] or a customized checkerboard specific operator) followed by line extraction through a technique such as the Hough transform [35]. A review of checkerboard pattern extraction and a Hough-based technique is provided by Escalera et al. [27]. Other related literature includes [19] with an intensity saddle-point operator, [78] with a sub-pixel refinement process for approximate checkerboard locations, and a multi-scale wavelet "X-corner" detector supporting incomplete checkerboard patterns [178]. Detection of squares with uniquely coded identification marks for multiple view correspondence is considered in [40], and results are presented indicating that extraction of the center of a square is more reliable than the corners in camera calibration applications. A saddle point detection scheme not requiring surface fitting is reported in [99].

Checkerboard patterns have received significant attention in the literature because of their popularity for camera calibration, but as described in Section 2.1.3, asymmetry makes them unsuitable for close-range vision applications. The results of Fiala et al. [40] demonstrate an expected reduction in measurement accuracy when detecting corners in the presence of focus-related blur, and reinforce the selection of line-based grids for surface strain application.

Deformed sheet metal surface grid extraction is explored by Tuzikov et al. [167], in which a morphological approach followed by skeletonization is used to extract the grid intersection information. The paper notably observes that for the image data tested, metallic reflections and grid damage from deformation result in a unimodal greyscale distribution. This observation is significant because it precludes many thresholding techniques that rely on bimodal pixel intensity distributions. Although the work summarizes a sheet metal grid extraction algorithm, the morphological approach does not extend to DOF blurred imaging with varying target range and scale.

3.2 Feature Detectors for Line-grid Measurement

In this section, approaches for grid line intersection measurement in the presence of depth of field blur are considered. The approaches are based on well known classes of feature detectors, and the merits of each in terms of robustness and stability are briefly considered.

3.2.1 Edge-based

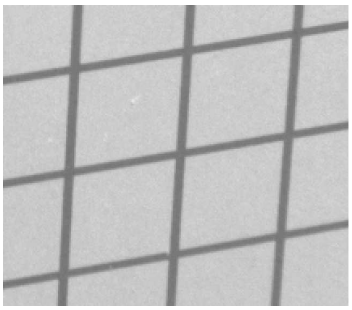
Edge detection operators can be used to detect the sides of grid lines in images. These operators typically define edges based on spatial gradients in the image intensity, with many modification and enhancements proposed throughout the literature. Early seminal work on edge detection was by Marr and Hildreth [105], combining a Gaussian smoothing operation with a spatial derivative through the Laplacian of Gaussian. Canny [14] described an edge detection operator that was designed to be "optimal" in terms of "good detection", "good localization", and single response to a single edge. Canny also observed an uncertainty principle involving a tradeoff between detection and localization of step edges in the presence of noise. Fleck [41] examined defects such as deformations and gaps produced by common edge detection operators.

Sub-pixel edge extraction accuracy experiments were performed by Devernay et al. [30] and Steger [148] in the presence of noise and varying line angle. Steger obtained edge position error with maximum standard deviation of $\approx \frac{1}{30}$ pixels, while Devernay et al. measured position error with standard deviation $< \frac{1}{10}$ pixels. An analysis of line and edge extraction from images using the Steger approach is presented in [149], with consideration of bias and lines of unequal polarity.

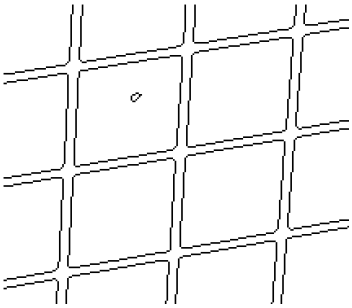
Figure 3.2 shows edge detection results from two sample image regions with different levels of DOF blur. Results from application of the Canny edge operator to the original image are shown, followed by the Canny operator applied after smoothing with two Gaussian filter variances ($\sigma = 2$ and $\sigma = 4$).

Four primary challenges emerge when considering edge detection for close-range grid line intersection measurement, as visible in Figure 3.2.

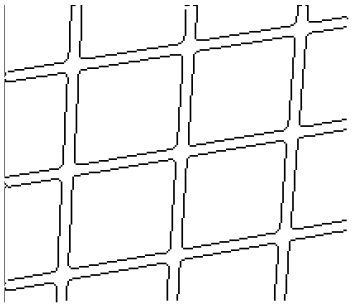
1. **Edge detection in varying blur:** The presence of blur causes even ideal step edges to become diffuse. Edge detectors are typically matched to a level of diffuseness through a low-pass filter or smoothing stage, which suppresses both high frequency noise and edges that are sharper than expected. With a continuously varying level of blur (within even a single image), the necessary filter parameters cannot be pre-determined or set as constant across an image. Diffuse blurred edges furthermore lead to loss of localization information, causing a shift in detected location for many standard edge operators. Multi-scale wavelet edge detection [55] and scale-space techniques can address this difficulty.



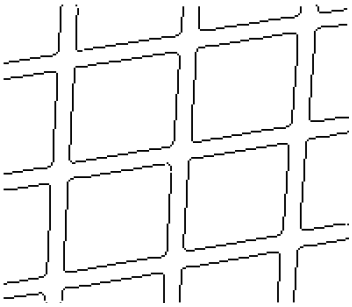
(a) Original image region 1.



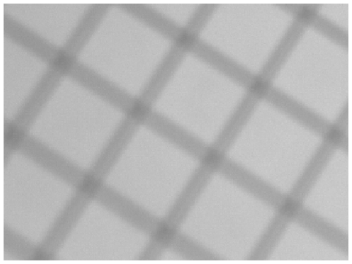
(b) Canny operator applied to image region 1.



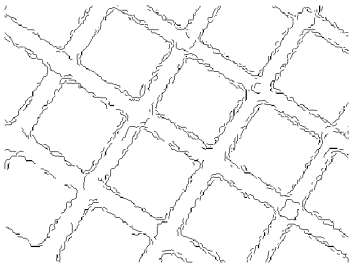
(c) Image 1 after Canny operator with Gaussian smoothing ($\sigma = 2$ pixels).



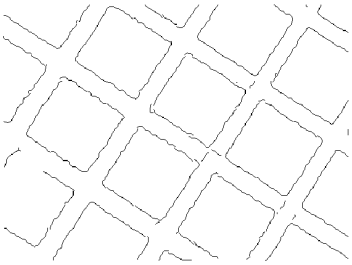
(d) Image 1 after Canny operator with Gaussian smoothing ($\sigma = 4$ pixels).



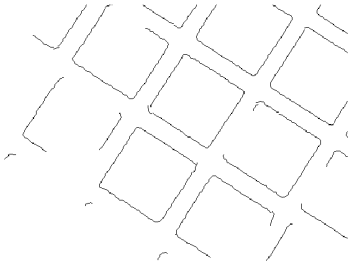
(e) Original image region 2.



(f) Canny operator applied to image region 2.



(g) Image 2 after Canny operator with Gaussian smoothing ($\sigma = 2$ pixels).



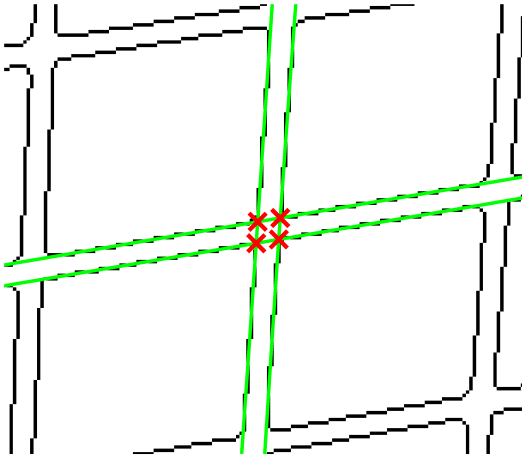
(h) Image 2 after Canny operator with Gaussian smoothing ($\sigma = 4$ pixels).

Figure 3.2: Canny edge detection results on two regions of typical camera images, showing examples of in focus and blurred imaging.

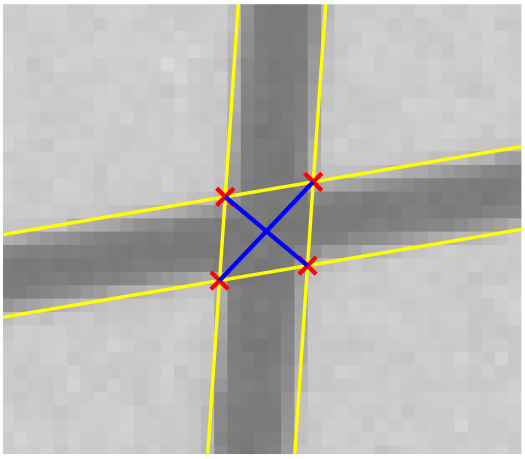
2. **Grid line intersections based on middle of line, not edges:** For a line-based grid, each line has two edges. Calculation of the intersection to subpixel accuracy requires an interpolation of the line centre. Figure 3.3 shows three such approaches. Parabolas may be fit to the edges, and intersections of these parabolas found to locate corners around the intersection. These corners can then be used to estimate the centre line intersection point, using techniques such as shown in Figures 3.3(b) and 3.3(c). The line centre may alternatively be interpolated from the edge parabolas and intersected, as shown in Figure 3.3(d), using a parabola averaging technique. A problem emerges in the bias introduced by many edge detection operators, which must either be removed or shown to be symmetric for any angle and diffusion of edge to be detected. Asymmetry in bias between the edges on either side of the line immediately leads to error or bias in the line centre estimation, and therefore intersection measurement error. Detection asymmetry has been shown between the two edges of a line in, for example, [148].

3. **Disconnected line segments:** Grid lines that intersect are coloured nearly identically on the metal surface, so an edge-based approach forms islands around the interiors of the grid lines (edges do not extend beyond a single grid unit, as they are interrupted by the next intersecting line). Edges belonging to a common line must be separated from the corresponding island, and associated with other edge segments belonging to the same line before the intersection can be measured. One solution, using a k-curvature approach to isolate edge fragments, was presented by Mitchell [106].

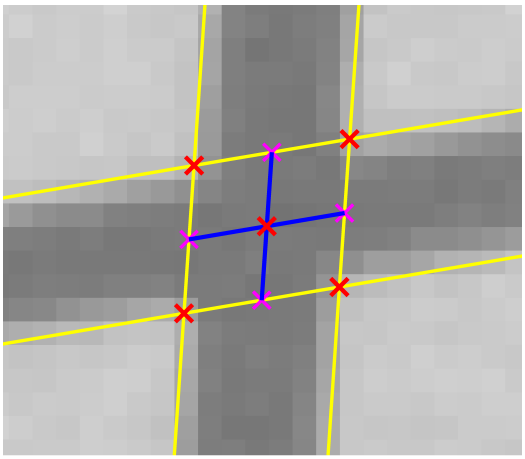
4. **Incomplete/noisy edges:** When noise or DOF blur are present in an image, the resultant edge response is typically broken with gaps and other inconsistencies appearing along the true edge. A decreased signal-to-noise ratio in significantly blurred image regions leads to increased noise detection, the result of which can be seen in Figure 3.2. To extract grid lines it becomes necessary to either fill the gaps, or to estimate the line information using techniques such as the Hough transform [35] in the case of straight lines.



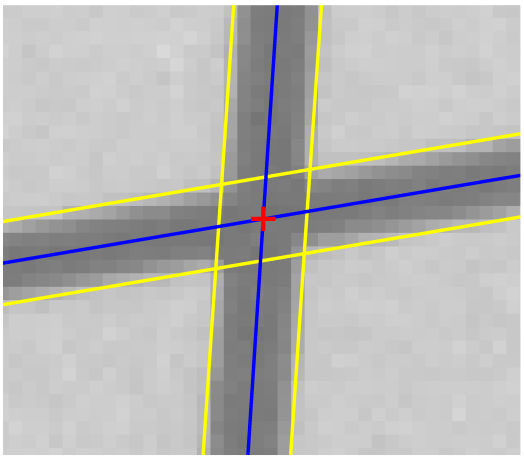
(a) Region of grid after application of Canny edge detector (Gaussian $\sigma = 2$). Light (green) lines are least-squares parabolas fit to the edge data on either side of the line intersection.



(b) Line intersection estimate from intersection of parabolas fit to line edges. Crossing of the (blue) lines is estimate of intersection centre.



(c) Line intersection estimate from intersection of parabolas fit to line edges. Midpoints of line segments between the parabola intersections approximate grid line centres.



(d) Midpoint line as average of parabolas fit to outer line edges. Intersection of the central lines estimates grid line intersection.

Figure 3.3: Three approaches for grid intersection estimation from parabolas fit to line edges. Canny edge detector used for this example (preprocessing Gaussian $\sigma = 2$).

Edge detection is an indirect approach to grid line intersection measurement, requiring interpolation of line centres from edge data. Coupled with the loss of localization and continuity in blurred imaging conditions, and the potential for asymmetric bias in complimentary edges, other approaches for grid line measurement are considered.

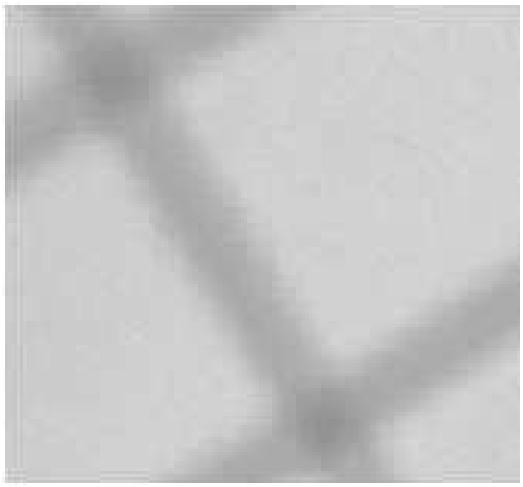
3.2.2 Ridge-based

Multiple representations of pixel intensity data from a grid image region are shown in Figure 3.4. The original image is shown in (a), followed by a negative version of the same region (b). Three dimensional views of the negative data (c) and (d) consider the greyscale intensity to be a third vector dimension, and show that the imaged grid lines appear as intensity ridges. Alternatively, without considering the negative image, the darker grid lines can be considered as valleys in the intensity data.

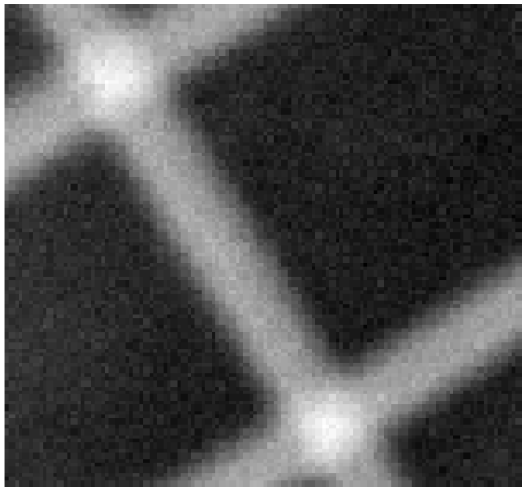
Early work by Haralick defined image intensity ridges as "zero crossings of the first directional derivative taken in a direction which extremizes the second directional derivative" [58]. This definition identifies ridges as maxima in the pixel intensity curvature, with the curvature defined by local principal directions. A review of ridge definitions for image processing is provided by [139], and a mathematically oriented review is conducted in [37]. For the remainder of this thesis, the term ridge is considered to mean either a ridge or valley feature. The distinction between these two terms is based upon arbitrary interpretation of pixel intensity data, specifically which extreme of the representable pixel values should be interpreted as the most positive in a coordinate system.

When considering grid lines as intensity ridges, as shown in Figure 3.4(d), it is apparent that extraction of the ridge feature naturally detects a grid line centre. Inherent line-centre detection removes the need for approximation from indirect data such as edges.

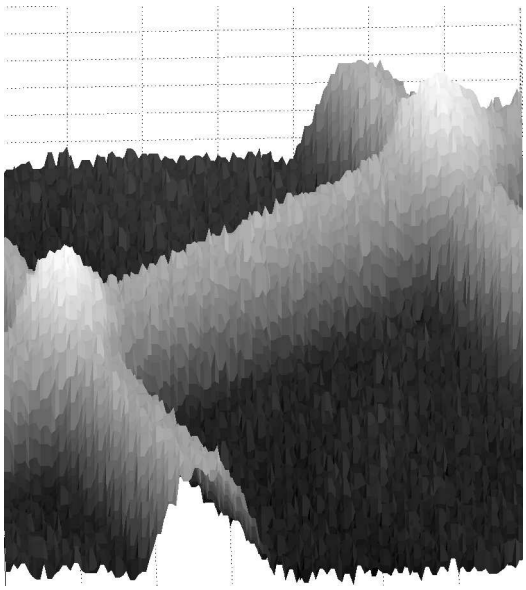
As in the case of edge detection, a low-pass filtering stage is typically performed before ridge detection to filter out high frequency noise, and to match detection to the approximate scale of feature. As changing DOF blur affects edge diffuseness and apparent line width, selection of a filter variance is no longer obvious. Scale-space techniques offer a solution, and are reviewed in Section 3.3.



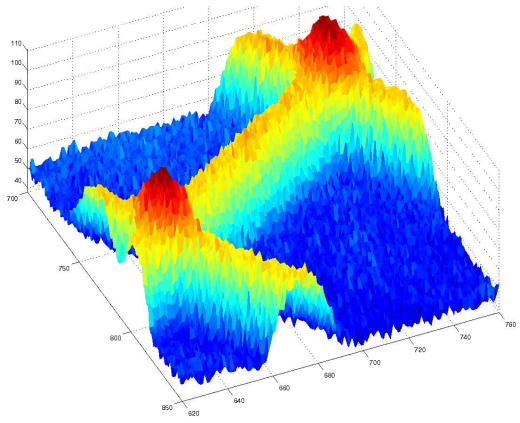
(a) Original image region.



(b) Negative pixel intensity.



(c) Three dimensional view of intensity ridge.



(d) Coloured view of intensity ridge.

Figure 3.4: Interpretation of image intensity data. Lines appear as intensity ridges or valleys.

3.2.3 Corner-based

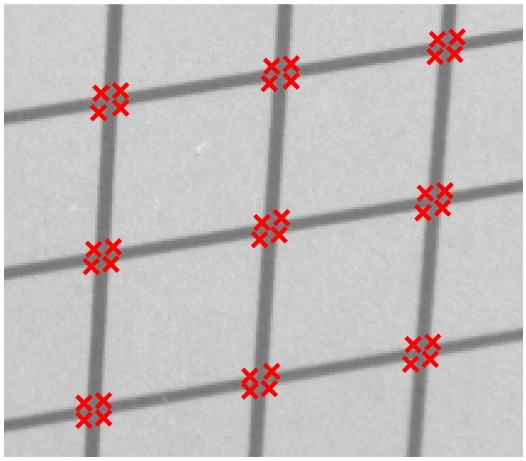
Image corners can be defined in many ways. One common definition considers corners to be sharp changes in the directions of edges extracted from the image. Many modern corner detectors do not rely on edge or other feature data, but instead look for changes in pixel similarity over small image patches. The Moravec corner detector [108], and an extension known as the Harris corner detector [59], are two common approaches to image corner extraction. Figure 3.5 shows various views of a Harris detector applied to a sample image. Multi-scale (scale-space) corner detection approaches are considered in literature such as [28, 29, 107, 186, 183, 94]. A comparison of two feature detectors (KLT and Harris) in the context of GPU acceleration are considered in [160].

Examining close-range grid lines, four corners are present around each line intersection. From these corner points, an intersection centre can be interpolated, as shown in Figure 3.5. Four difficulties are identified when considering intersection measurement from corners.

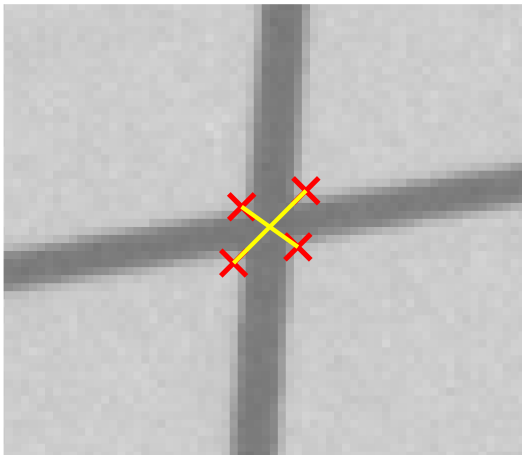
1. **Potential asymmetry in detection:** The four corners surrounding a grid line intersection are dissimilar in terms of pose. To accurately interpolate the intersection, detection of the corners must be perfectly symmetric in terms of bias. Given the discrete nature of pixel data and small pixel neighbourhood in which a corner is present, this requirement is difficult to meet with known corner detection algorithms.

2. **Indirect measurement:** Basing line intersection measurement on corners is an indirect approach, so small errors in feature detection have a significant affect on intersection interpolation accuracy. Furthermore, the corner points are not detected along the midpoint of the grid line, so detection errors in range from the feature (closer or further from the intersection) also reduce measurement accuracy.

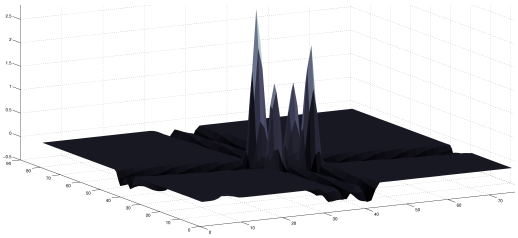
3. **Noise:** Using four corner points to interpolate the grid intersection is a sparse approach to measurement, increasing susceptibility to noise in the corner detection process. This differs from edge or ridge methods, where dense data along the feature can be used in a data fitting formulation.



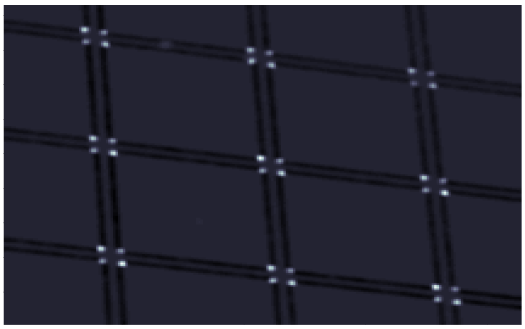
(a) Image region with strongest Harris corner responses overlaid.



(b) Close-up of single grid line intersection, with corners joined by lines estimating the intersection point.



(c) Three dimensional view of Harris corner metric response around a single grid line intersection.



(d) Top view of Harris corner metric response around a single grid line intersection. Lighter shading indicates a more positive response.

Figure 3.5: Harris corner detector [59] applied to a sample image region. In the Harris detection, the Gaussian standard deviation was $\sigma = 1.5$, and a sensitivity factor of $k = 0.04$.

4. **Very diffuse line edges:** Highly DOF blurred and thus diffuse lines are not amenable to similarity-based corner detection because the signal to noise ratio becomes small. The magnitude of peaks in corner response metrics become less prominent and difficult to distinguish from, for example, lighting variations.

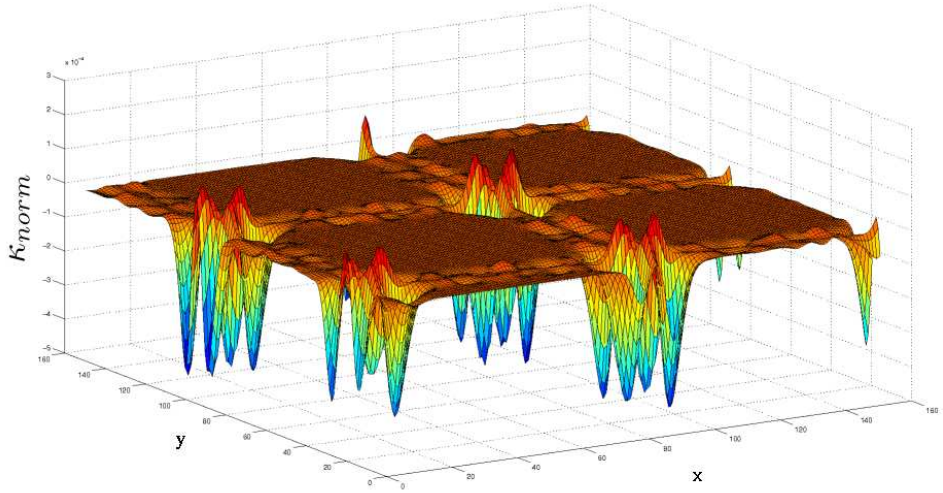
3.2.4 Junction-based

Junctions are naturally defined by the intersection of other features such as edges or ridges. Many classifications exist, including "T-" and "Y-junctions". Grid line intersections can be considered as either a grouping of four coincident "T-junctions", or as a single instance of an "X-junction". Two general classes of approaches have been used for junction detection in the literature: First are template matching approaches such as [120], and second are feature-based approaches that consider intersections of other features such as described by [125]. A scale-space approach for multi-scale junction detection is considered in [91].

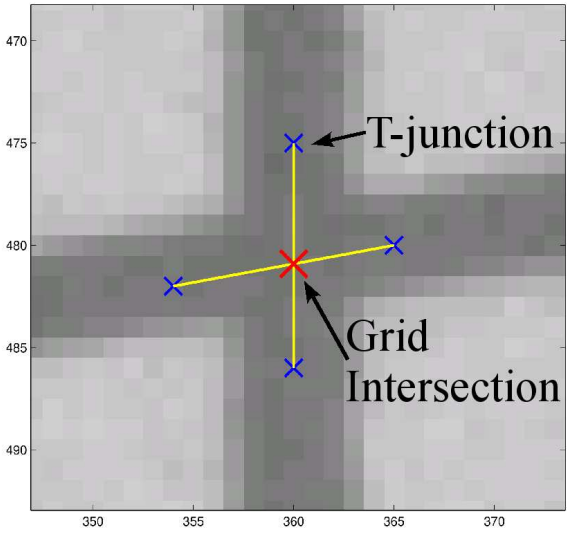
Figure 3.6 shows sample output from a scale-space junction detection approach based on [91], with the intention of illustrating T-junction detection in the context of grid measurement.

T-junction detection is similar to corner extraction in that sparse data, in this case four data points, are used to interpolate the line intersection location. Junctions have a significant advantage over corners, however, as errors in range from the intersection point do not affect the interpolation accuracy. Such range errors can be caused, for example, by slightly varying DOF across the intersection leading to differing scales of detection. Detected junctions are aligned with the grid line centre, so intersection measurement is invariant to scale-induced ranging errors.

When considering T-junctions, the grid line intersection is interpolated from only four data points. This sparse data (compared with ridge or edge fitting) cannot directly be used to compensate for curvature in the line.



(a) T-junction metric plot around a grid intersection.



(b) Interpolation of grid line intersection from surrounding T-junctions.

Figure 3.6: Sample junction detector output.

3.2.5 Thresholding and Skeletonization

An alternative approach to intersection measurement involves thresholding the image [117, 136] to produce a binary mask indicating presence of a grid line. A skeletonization process can then be applied, and the resulting representation used to locate grid intersections through parabola fitting or template matching. A survey of skeletonization (thinning) approaches is provided by [83]. Although dense data skeletons are used in the interpolation, the difficulty with such techniques is in the initial thresholding and binarization process. Some algorithms in the literature such as [31] address direct skeletonization from images, without first thresholding or binarizing the image. In the presence of DOF blur and varying lighting conditions, coupled with the potentially unimodal greyscale histograms of the metal samples (as observed by [167]), adaptive thresholding techniques are typically not reliable enough to consistently detect the grid lines. In DOF blur, asymmetric thresholding of opposing line edges biases the skeletonization result, direcly leading to measurement error.

3.3 Scale-space

3.3.1 Introduction

As described previously, close-range imaging generates a variety of challenges that complicate feature measurement. Varying blur and changing range to target manifest as change of size or scale of the imaged grid lines. The apparent line width changes continuously within and across images, so feature detectors cannot be matched to a scale in advance through a priori information. A natural solution to feature detection in these conditions is to consider multi-scale image processing techniques. A number of multi-scale approaches have been explored extensively in the literature including pyramid representations [102], wavelet [103] and fractal analysis [123], and scale-space techniques. Scale-space is chosen for the grid extraction process because it forms a framework within which different feature detectors can be applied, and is adaptable to a variety of grid features without significant rework.

A scale-space representation of an image is formed by embedding the pixel data into a one-parameter family of derived signals, as described by the early literature

[176, 76]. The basic idea behind a scale-space representation (in the context of this work) is that a two-dimensional image is transformed into a discrete three-dimensional data set, with the added dimension controlling the scale below which feature responses are in some manner suppressed. In this way, at higher scale levels in the data set, feature detectors can locate larger scale-features as opposed to the stronger responses caused by small scale structures. The parameter $t \in \Re$ is known as the scale parameter, with increasing values indicating an increase in the scale at which corresponding features are measured.

To be useful, a scale-space must not enhance noise or generate singularities with increasing scale parameter [76], and should therefore lead to simplification of the image structure as scale increases. A variety of requirements have been specified for the definition of a useful scale-space, as summarized by [92], including homogeneity and isotropy. For the continuous case, it has been shown in numerous works that the unique kernel exhibiting the required linear scale-space properties is the Gaussian function [5, 45, 89, 181].

A basic scale-space is defined on an image f as:

$$L(x, y; t) = g(x, y; t) * f \tag{3.1}$$

where g represents a two-dimensional Gaussian kernel with variance $\sigma^2 = t$:

$$g(x, y; t) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}} \tag{3.2}$$

The notation $L(x, y; t)$ denotes spatial location (x, y) in pixel coordinates, at scale level (variance) t pixels^2. Derivatives of this scale-space are defined as:

$$L_{x^\alpha y^\beta}(x, y; t) = \delta_{x^\alpha y^\beta} L(x, y; t) = g_{x^\alpha y^\beta}(x, y; t) * f \tag{3.3}$$

where α and β define the order of differentiation in the x and y image dimensions [89]. This definition provides the property that the Gaussian smoothing operator and derivative operator commute as:

$$\delta_{x^\alpha y^\beta}(g * f) = (\delta_{x^\alpha y^\beta} g) * f = g * (\delta_{x^\alpha y^\beta} f) \tag{3.4}$$

Since convolution in the spatial domain is equivalent to multiplication in the Fourier domain, the question arises of the best domain in which to generate the scale-space. Florack [42] has shown that Fourier domain formation is more accurate for small spatial scales, while spatial filtering provides better accuracy at larger scales. A formula is derived to determine the scale at which this division occurs. In terms of computational complexity, the Florack accuracy results are opposite to the desired behaviour for computational load. Fourier domain computations are typically preferred for large scale convolutions in terms of computational efficiency, whereas the Florack results prefer spatial filtering for improved accuracy with large support kernels.

A key scale-space concept is the idea that an image is filtered at multiple scale levels, as opposed to just a single filter variance level as is common in preprocessing for standard feature detectors like the Canny operator. By filtering at multiple levels and considering the collection of progressively blurred images as slices of a three-dimensional space, the image may be considered at all of the scale levels simultaneously, rather than as a collection of unrelated images [76].

An alternative view of the scale-space formulation is as the solution to the diffusion equation [92]:

$$\delta_t L = \frac{1}{2}\nabla^2 L \tag{3.5}$$

subject to the initial condition:

$$L(x, y; 0) = f(x, y) \tag{3.6}$$

The diffusion equation interpretation of scale-space filtering motivates some solutions for application to discrete image data.

Early work to apply linear Gaussian scale-space operators to discrete pixel data utilized discrete (sampled) approximations of the continuous Gaussian, but it has been shown that under this process the scale-space conditions do not necessarily hold. Lindeberg [90] proposed that a natural way to construct a discrete scale-space is to convolve the discrete signal with a kernel defined by solution of the semi-discretized

diffusion equation. The result was a discrete filtering kernel, defined as:

$$T(n;t) = e^{(-t)} I_n(t) \tag{3.7}$$

where I_n are the modified Bessel functions of integer order.

Extending this work and solving some of the open questions, Lim and Stiehl [88] proposed a generalized discrete scale-space formulation. The resultant discrete convolution kernel is separable and least rotationally asymmetric when expressed in terms of spatial derivatives. The convolution kernel for each dimension, with a smoothing parameter step size of $\Delta t = \frac{1}{3}$, is:

$$k = [\frac{1}{6} \quad \frac{2}{3} \quad \frac{1}{6}] \tag{3.8}$$

An alternative Gaussian-derived scale-space generating kernel that offers reduced computational cost, while preserving some important properties of the Gaussian, is presented in [131]. B-spline kernels for scale-space generation are considered in [126], which also describes a link between scale-space and wavelet techniques.

3.3.2 Feature Detection

After generation of a scale-space image representation using Equation (3.1) or the equivalent, feature detectors may be defined that operate within that space. An introduction to scale-space techniques in the context of image analysis is provided by [134]. A significant portion of the work in this area utilizes differential invariants and the detection of differential singularities [80, 43, 44]. Differential invariants are employed primarily to provide rotational and scale invariance of the feature detector, and differential singularities such as zero crossings of the differential invariants form the basis of feature detection.

Automatic Scale Selection

Scale-space is a multi-scale representation, so geometric feature detectors locate the same feature at multiple scales throughout the space. A method is therefore required to choose the scale at which the feature is best localized spatially. Lindeberg [93, 10,

95] proposed that local maxima over scale of combinations of normalized derivatives could produce an automatic scale selection formulation. He then further proposed that normalized spatial derivatives be formed in terms of a γ-parameter to control the scale selection for specific feature detectors.

3.3.3 Scale-space Ridge Detection

Grid lines to be detected in the images are treated as intensity ridges. A brief analysis of ridge behaviour in scale-space is presented by [20]. In the context of scale-space ridge detection, a coordinate system (p, q) can be defined at each pixel in which the axes are parallel to the local principal curvatures of the image intensity [93]. Using non-maximum suppression, a condition of locally maximal intensity in the principal curvature direction can be defined in the local coordinate system as:

$$\begin{cases} L_q & = 0 \quad \text{and} \\ L_{qq} & < 0 \end{cases} \tag{3.9}$$

Equation (3.9) can be rewritten in terms of its spatial derivatives, and used to define ridges at a specific scale level. Since the same ridge may be detected across multiple scales, a feature strength metric is used. The definition of a ridge is extended to include the requirement that this feature strength metric is maximum with respect to the scale parameter. Multiple ridge strength measures have been proposed by Lindeberg [93], one of which is the absolute value of the principal curvature. Ridges in scale-space may thus be defined by:

$$\begin{cases} L_q & = 0 \\ L_{qq} & < 0 \end{cases} \qquad \begin{cases} \delta_t(\varepsilon_{norm} L(x, y; t)) & = 0 \\ \delta_{tt}(\varepsilon_{norm} L(x, y; t)) & < 0 \end{cases} \tag{3.10}$$

where $\varepsilon_{norm} L(x, y; t)$ is the scale-normalized ridge strength metric. The intersection of the conditions defined in Equation (3.10) are interpolated to provide a sub-pixel estimate of the ridge location within the image plane.

The conditions in (3.10) form two isosurfaces through the scale-space. Where the ridge definition isosurface meets the scale selection surface, a ridge is defined.

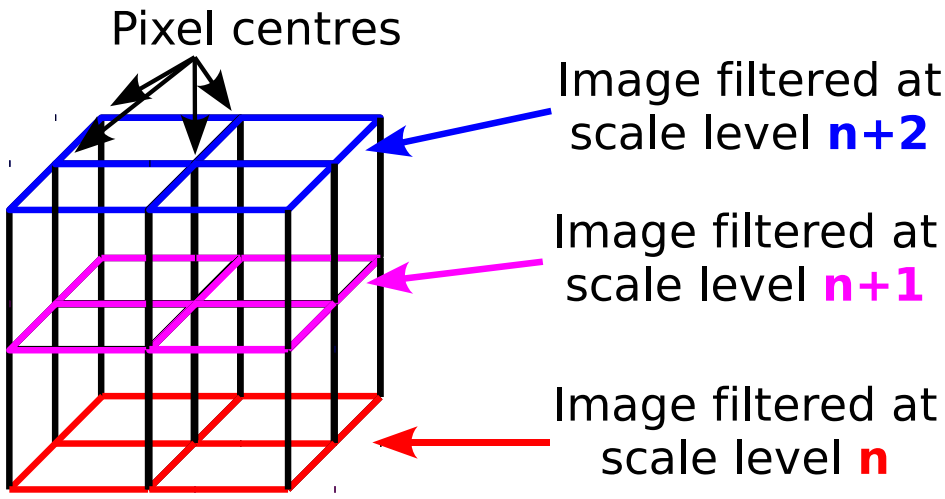


Figure 3.7: Small sample of the discrete scale-space organization (2×2 pixel image with 3 scale levels). The filtered image at each discrete scale level is coloured uniquely in the diagram, and all cube vertices represent pixel centres at a specific scale level.

Interpolation of surface intersections in the discrete three-dimensional scale-space can be accomplished using techniques based on the marching cubes algorithm [96, 161]. GPU-based acceleration of the marching cube algorithm is described by [70]. A survey of related literature is provided by [109], and a performance analysis of isosurface extraction algorithms is performed in [156].

The interpolation and intersection of metric zero isosurfaces is implemented in this work using a cube interpolation model, similar to the marching cubes algorithm. The interpolating cube is formed with adjacent image pixels at each scale level forming the vertices of each horizontal face (top and bottom faces of the cube). The vertical axis represents a step between scale levels, with one discrete scale level on the top face of the cube, and an adjacent scale level at the bottom face.

A small block of the discrete scale-space is illustrated in Figure 3.7, and a similar block with illustrative ridge segments is shown in Figure 3.8. The true scale space is much larger than the $2 \times 2 \times 2$ block shown in the figures, but the organization of the interpolating cubes is representative. Interpolation of ridge segments from isosurface intersections occurs in each block of the scale space, with at most one ridge segment passing through a single block.

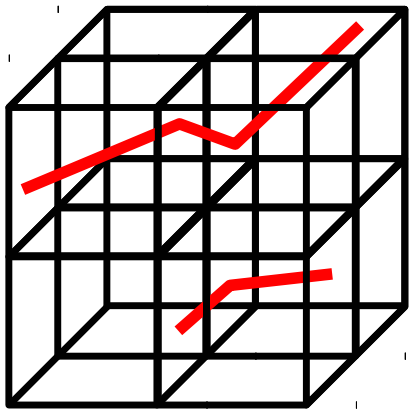


Figure 3.8: Sample $2 \times 2 \times 2$ block of interpolating cubes with sample ridge segments inside the cubes. The points where the ridge segments pierce the sides of a cube are interpolated, and connected to form continuous ridges.

Computation of the metric values required by (3.10) are performed at each cube vertex in the discrete scale space, as the vertices are by definition located at the filtered pixel centres. Zero crossings of the metric values between cube vertices are interpolated (along the cube edges), and when two edges of a cube face contain zero crossings, a line is considered to connect them. This line is an approximation of where a corresponding metric zero isosurface cuts through the face in the scale-space. Where both metrics contain zero crossing on a single interpolating cube face, and if these crossings intersect, then a ridge point is defined. When two faces of a cube contain ridge points, then a ridge segment is considered to connect these points, and is an approximation of where the conditions of Equation (3.10) have met within the cube volume.

A view of a single interpolation cube is shown in Figure 3.9. The metric isosurfaces defined by the conditions in Equation (3.10) are shown as interpolated on the cube faces (lines coloured red and blue). Intersection of the isosurfaces is represented as a line between two of the cube faces (coloured green). Ridge segments formed in adjacent cubes are subsequently linked together to form a continuous ridge.

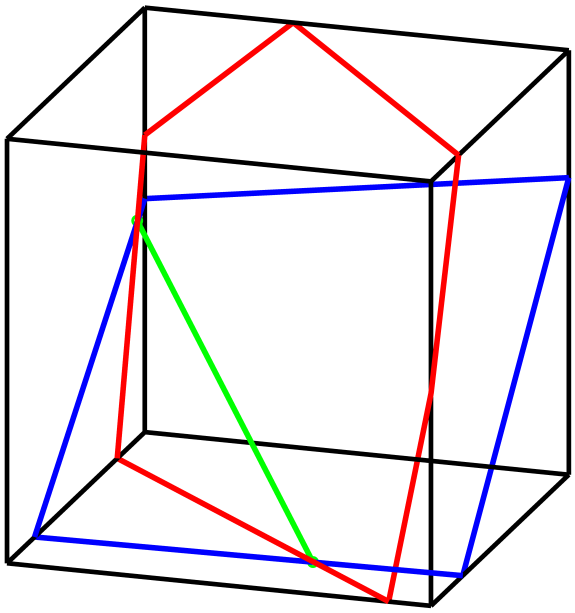
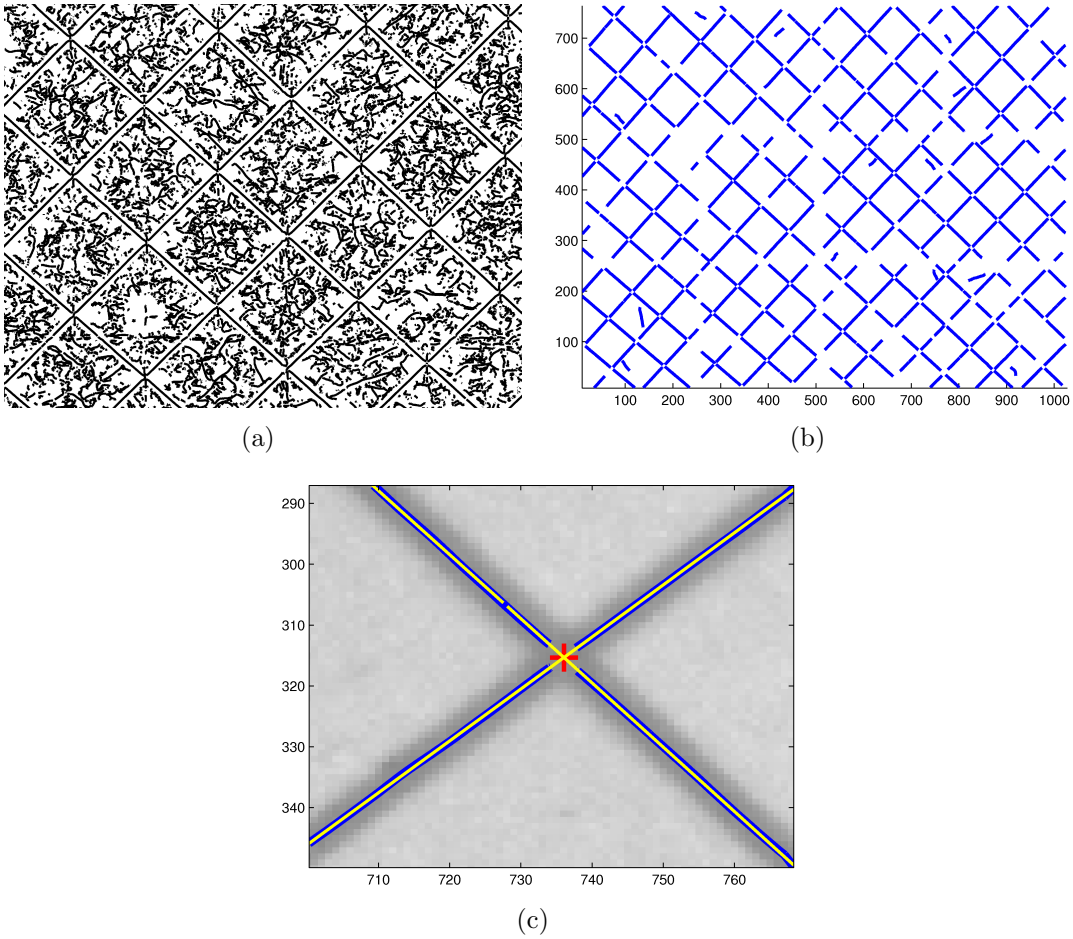


Figure 3.9: Isosurface intersections with the sides of an interpolation cube - shown as (red and blue) lines on cube faces. The interpolated surface intersection that forms the ridge segment is shown traversing between two cube faces (coloured green). The cube vertices are formed by adjacent pixel centers on the corners of a horizontal face, and a step between scale levels forms the vertical axis.



(a) (b)

(c)

Figure 3.10: Ridge extraction: (a) Zoomed view of unfiltered ridges; (b) Ridges after filtering; (c) Close-up of fitted parabolas (light/yellow lines) and corresponding grid intersection point.

Ridge Filtering and Parabola Fitting

Figure 3.10 shows the sequence of high level steps used to obtain the grid intersection points from raw ridge data. In Figure 3.10(a), a sample of unfiltered ridge data is shown. Between ridge lines, a substantial amount of noise is detected because well defined ridges are not present. Figure 3.10(b) shows the ridge lines after a multi-step filtering process (described in Chapter 4). The filtering is based upon scale-level gradient because true grid lines do not vary sharply in focus, and curvature-based filtering is used to eliminate sharply curved (non-grid line) structures.

To interpolate the grid intersection points, a search is performed in the neighbourhood around the end of each detected grid line segment to find the closest neighbouring ridges. Groups of four grid lines are then divided into pairs based upon similar direction in the image plane, and least squares parabolas are fitted to these pairs of lines. Parabolas are chosen because they can conform to the curvature of deformed surfaces. Figure 3.10(c) shows the detected ridge points and overlaid parabolas for a typical grid region. The intersection of these parabolas is computed, and taken as a sub-pixel estimate of the line intersection. The parabola fitting and intersection process is described in more detail in Chapter 4.

The proposed grid intersection detector extracts ridge points at an inherently sub-pixel level spatially, and to a sub-discretization level in the smoothing parameter dimension. Dense data along a ridge is generated in the form of multiple interpolated points, reducing the effect of detection artifacts and image noise when sub-pixel grid intersections are estimated through parabola fitting.

3.3.4 Computational Complexity

The scale-space computational workload increases linearly with the image size and number of scale levels. To provide a concept of the workload, the separable convolution required for scale-space generation can be considered. To obtain a scale filtering step of $\Delta t = n$ pixels^2 based on Equation 3.8, a symmetric kernel of size $1 + 6n$ is used, formed through repeated convolution of a base kernel as described further in Section 4.2.2. For this example, two approaches are considered to convolve this kernel with actual image data. First, a Fused Multiply-Add (FMA) can be used at

Table 3.1: Convolution Operation Total per Second (15 fps)

Image Size	Discrete Scales	Δt	Approach 1	Approach 2	
			FMA (Gops)	Multiply (Gops)	Add (Gops)
1024×768	8	1	1.32	0.76	0.57
	8	3	3.59	1.89	1.70
	8	6	6.98	3.59	3.40
	16	1	2.64	1.51	1.13
	16	3	7.17	3.77	3.40
	16	6	14.0	7.17	6.79
2448×2048	8	1	8.42	4.81	3.61
	8	3	22.9	12.0	10.8
	8	6	44.5	22.9	21.7
	16	1	16.8	9.63	7.22
	16	3	45.7	24.1	21.7
	16	6	89.0	45.7	43.3

each kernel element, leading to $1 + 6n$ operations for each convolution. After both horizontal and vertical convolutions are performed, and assuming that the scale-space contains m scale levels, $(2 + 12n)m$ FMA operations are required per image pixel.

A second approach exploits the observation that the convolution kernel is symmetric, meaning that a pixel value will be multiplied with the same coefficient twice by neighbours on opposite sides, but the same distance away. If intermediate data storage is available, the result of the first multiplication can be buffered for use by the second instance. In this case, $1 + 3n$ multiplies and $3n$ additions are necessary. With m scale levels and two dimensional convolution, $(2 + 6n)m$ multiplies and $6nm$ additions are required per pixel.

Table 3.1 shows the operational totals required for convolution using both of the approaches (FMA and Multiply/Add), for various image sizes, number of scale levels, and interscale filter variance (Δt). Video processing at 15 frames per second is assumed, and the image resolutions chosen correspond to a 0.8 Mpixel Point Grey Research Dragonfly camera, and also a 5 Mpixel Point Grey Grasshopper 2 camera.

Considering a (currently) high end Intel i7-975 quad-core processor, with a theoretical single precision capability of 110 GFLOPS (extrapolated from Intel export compliance metrics), the convolution phase of scale-space generation produces a substantial workload. Data movement is typically the major bottleneck when attempting

to achieve theoretical FLOPS performance on a modern CPU, and as such significantly reduces the effective processor capability. With inclusion of the metric value computations, isosurface intersection and ridge extraction, plus data capture and operating system activity, CPUs do not currently meet performance or scalability requirements.

To enable grid line intersection measurement at video frame rates, with a sufficient performance margin for practical implementation, Chapter 4 describes acceleration of the algorithm using GPU hardware.

3.4 Summary of this Chapter

Feature detection approaches have been considered for the measurement of grid line intersections in the presence of close-range DOF blur. A scale-space ridge based approach has been proposed for grid intersection extraction from close-range video sequences, allowing for dense data parabola fitting, and operation in changing range/blur conditions. Although ridge detection enables close-range grid line measurement, it is computationally intensive and impractical at video frame rates using a conventional CPU.

Chapter 4

GPGPU Accelerated Grid Line Intersection Measurement

Scale-space ridge extraction enables measurement of grid line intersections in close-range images. Computational intensity, however, makes the approach unsuitable for video-frame rate processing using commodity CPU hardware. This chapter begins with a review of GPU hardware in the context of general purpose computation, and then describes the proposed parallel formulation of the scale-space algorithm for GPGPU acceleration. The parabola fitting and intersection approach is described, and results from implementation, including timing and accuracy experiments, are presented.

Please note that some material in this chapter has been previously published in the Journal of Physics: Conference Series [74][1], Journal of Computer Aided Design and Applications [73], CCECE Conferences in 2007 [71] and 2008 [72], and other aspects have been submitted to the IEEE Transactions on Instrumentation and Measurement.

4.1 GPGPU

Graphics processing units have evolved rapidly in both design and programmability since approximately 2003 [111]. Originally with memory and processing pipelines designed exclusively for graphics rendering, there has been a move towards general

[1]Published under licence in Journal of Physics: Conference Series by IOP Publishing Ltd.

(non-graphical) application of the memory and processing resources. Current devices have enabled General Purpose GPU (GPGPU) computing through evolution of both the device architecture, and the exposed programming interfaces. A good description of the evolution of GPU design and programming techniques is provided by [119].

Early work in GPGPU programming attempted to masquerade general computational tasks as graphics-oriented vertex and fragment shaders [87, 65, 82, 15]. A good survey of these early techniques is provided by Owens et al. [118]. Computer vision using fragment shaders is considered in [47], with implementation of a projective motion tracking algorithm. Evolution during recent years has produced the current generation of GPU programming interfaces, including the Compute Unified Device Architecture (CUDA) for NVIDIA hardware, and the emerging OpenCL which is compatible with both AMD and NVIDIA graphics hardware (as well as other non-GPU platforms). A DirectCompute interface is available within Microsoft Windows through the DirectX API, but is bound to that platform. Within the current programming paradigms offered by CUDA and OpenCL, the GPU is treated as a coprocessor, with the CPU orchestrating overall data movement and processing flow.

Computational speedups obtained through GPU processing, compared with CPU-only implementations, are often reported with up 1000 times acceleration (such as [128]). Lee et al. [86] performs a detailed analysis of multiple algorithms alternately optimized for both CPU and GPU, and concludes that the performance gap between modern CPUs and GPUs is closer to 2.5 times. The CPU and GPU chosen for this comparison, however, are not of the same technology generation, with the GPU being older in terms of fabrication technology (65 nm compared with 45 nm) and depreciated by one NVIDIA release cycle. A key factor identified in the discrepancy of 2.5 times versus many reported 100-1000 times GPU speedups is that some reported acceleration results are with respect to unoptimized (or poorly optimized) CPU implementations. This observation is consistent with experience in the field, where it appears relatively common for programmers to spend large quantities of time optimizing GPU code, while neglecting equivalent levels of optimization on the CPU-side. Multiple published papers report GPU speedups with respect to Matlab implementations, which are generally interpreted scripts and not designed for computational performance. A comparison of optimization techniques and bottlenecks in modern

CPUs, and performance comparison against GPUs in the context of Computed Tomography (CT) reconstruction, is provided in [163].

For comparison purposes, at the time of writing the current generation NVIDIA GPU (GTX 580, 40 nm fabrication) has a raw single precision compute capability of 1.58 TFLOPS and a global memory access bandwidth of 192.4 GB/s. The current generation of Intel processors that are price competitive with the GTX 580 (not including motherboard), such as the i7-970 (32 nm fabrication) have an approximate single precision capability of 160 GFLOPS, and a memory bandwidth of 25.6 GB/s. As described in the following sections, CPUs and GPUs are optimized for different types of workloads, leading to the apparent performance gap between the platforms.

GPU results described in this thesis are based on an NVIDIA GeForce GTX 480 graphics card, which utilizes the NVIDIA GF100 architecture. A second revision of the architecture (GF100B/GF110) has been released, but is fundamentally consistent with the GF100. The following sections introduce concepts related to CUDA programming and design, but are not intended to be a replacement for the detailed CUDA programming guides [112, 110, 113, 111, 115] or textbooks [75] on the subject.

4.1.1 GF100 Architecture

The modern CPU is designed for optimal performance from a single or small number of threads, and therefore includes advanced caches, branch prediction, and out of order execution among other technologies [63] that require significant chip area. In contrast, a GPU dedicates less chip area to predictive or dynamic scheduling features, and instead provides large numbers of simple processors with high bandwidth access to memory. The NVIDIA GTX 480 contains 480 processor cores, compared with 6 cores in the Intel i7-970.

NVIDIA has defined their architectural paradigm as Single Instruction Multiple Thread (SIMT). The idea behind SIMT is that a single instruction unit dispatches an instruction to a group of processors, which individually execute that instruction on differing data. With a single instruction unit driving parallel processors, it is apparent that the instructions performed on the parallel data must be identical. Divergence of program flow within a single grouping of processors (with single instruction unit) causes inefficiency as all cores are forced to perform all execution paths required by

threads in the grouping. The SIMT paradigm differs from the more conventional Single Instruction Multiple Data (SIMD) paradigm primarily through the ability to handle divergent control paths within groups of threads. A grouping of SIMT threads in CUDA is named a warp, and in current generation devices contains 32 threads. The threads within a warp should ideally perform exactly the same instruction path, but in cases where control flow divergence occurs within the warp, the GPU scheduler simply executes all required paths and masks output from the cores that did not require a specific instruction. Warp sizing and minimal control flow divergence within warps are fundamental elements considered when designing and programming efficient CUDA algorithms.

The low level computational block on a GF100 GPU is an individual processing core, which contains both a floating point unit and an integer ALU. The cores are grouped together within a Streaming Multiprocessor (SM), which is a cluster of 32 cores and supporting functional blocks. Figure 4.11 shows a logical view of a single SM, of which there are 15 in the GTX 480 GPU. Four special function units are shared by the 32 cores to perform transcendental functions such as trigonometry and square roots, and 16 load/store units allow memory addresses in cache or off-chip DRAM to be computed for use by 16 of the cores in a clock cycle. A register file provides single cycle memory to the cores, and is local to the SM. A key feature of the GF100 architecture (compared with previous generations) is presence of two scheduler/instruction dispatch units within a single SM. The instruction units each serve 16 of the 32 processor cores, allowing two different SIMT sequences to be in flight simultaneously within an SM. In this way, the schedulers can make efficient use of the resources for many workloads, for example scheduling half of the cores to use the available load/store units, while the remaining cores utilize the special function units or perform local arithmetic computations.

Execution Model

From a programmer's perspective, internal structure of a multiprocessor core is transparent. Instead, an execution model defines the relative layout of individual threads. Figure 4.12 shows the logical thread hierarchy which in turn drives the structure of CUDA work partitioning and data sharing. The fundamental work unit is a thread

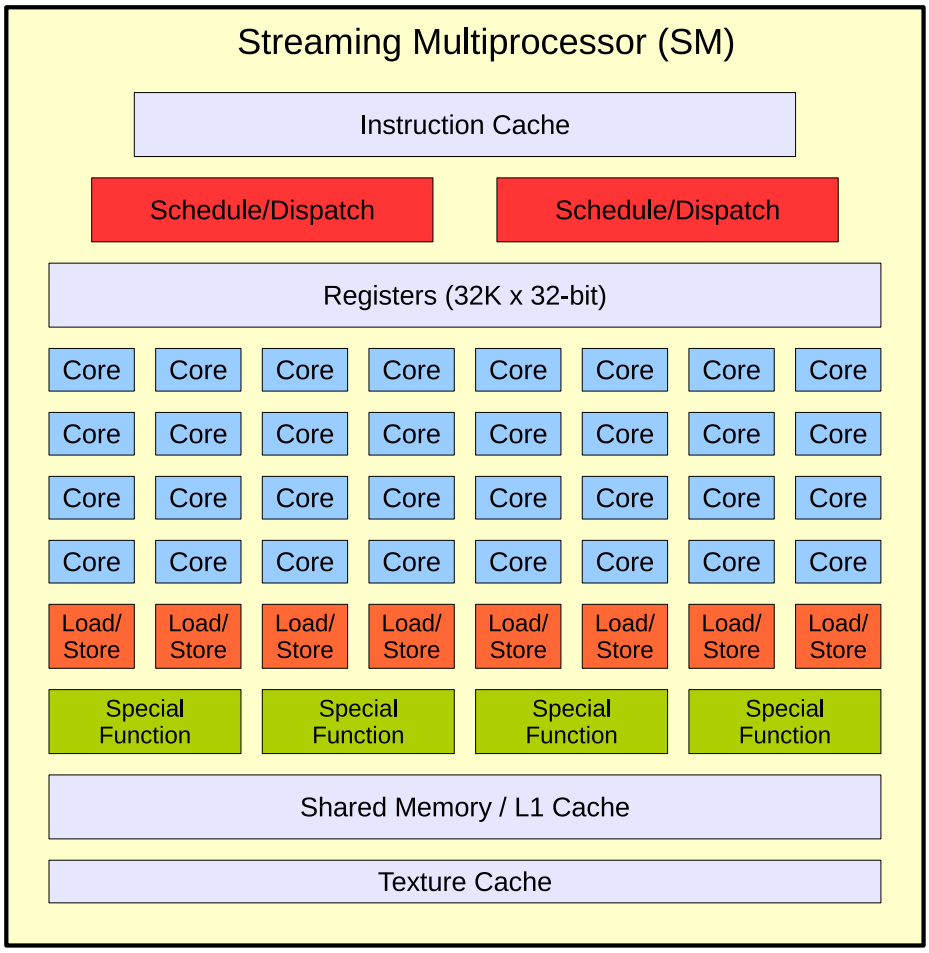


Figure 4.11: Block diagram of the NVIDIA GF100 Streaming Multiprocessor. Adapted from NVIDIA [113].

which performs a single stream of instructions, and is the finest-grained level of the hierarchy. Threads are grouped into blocks, which can be up to three-dimensional (logical address), with each thread receiving a unique address in the block. The blocks are integrally mapped into multiprocessors (SMs), and although more than one block may be physically scheduled into an SM, from a logical perspective the thread block has exclusive use of the multiprocessor. The blocks, in turn, are organized as a grid which can be up to two-dimensional. Blocks within the grid are logically dispatched to different multiprocessors, which leads to inter-thread communication ramifications as described later. With the dimension of the thread blocks defined as (D_X, D_Y, D_Z), a block contains $D_X \times D_Y \times D_Z$ threads. To be physically mapped into a multiprocessor, the thread block must not require more resources than are available within the SM. Especially in terms of memory resources, thread block partitioning plays an important role in CUDA algorithm design.

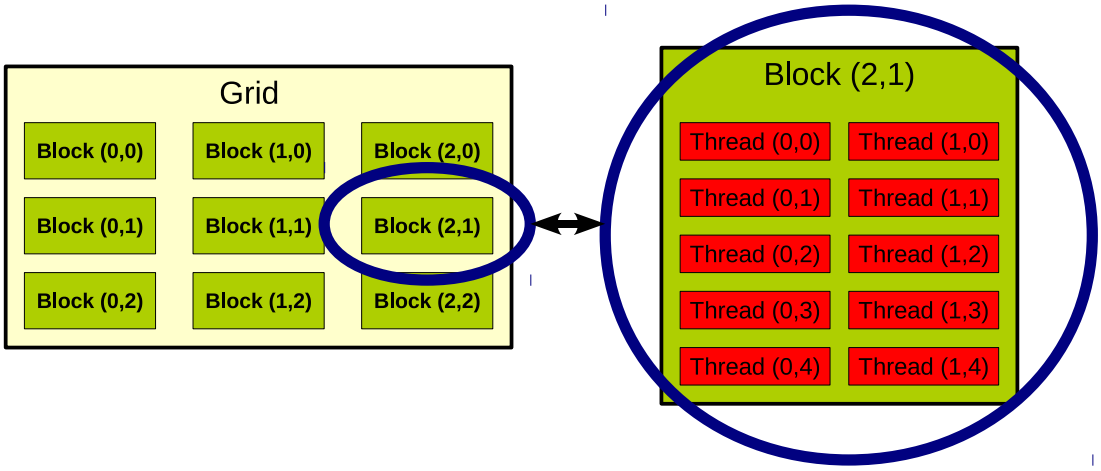


Figure 4.12: Logical organization of threads, thread blocks, and block grid in NVIDIA GF100 architecture for a block of dimension $(2, 5, 1)$, and a grid of dimension $(3, 3)$.

The GPU acts as a co-processor to the CPU in most implementations. Within CUDA applications, the CPU is responsible for transferring data between CPU-side and GPU-side memories, and for launching sets of threads for execution on the GPU. After launching a specific grid of threads on the GPU, the CPU can either block upon completion of the work, or can initiate other operations such as CPU-side processing

or memory transfers. Current generation devices allow memory to be copied concurrently with GPU computation, and multiple kernels can be launched simultaneously on the GPU hardware.

A program sequence that executes on the GPU is known as a kernel, and although it may execute as thousands of concurrent threads, program code that defines the kernel has a single definition. Within the program code, instructions must inherently compute the addresses of data to process based upon the unique identification of the thread within a thread block, and the block identification within the grid. Each thread of a running kernel therefore has a unique identification in terms of its location in the logical thread-block-grid structure, which it uses to identify the memory that should be accessed. Logical address in the thread hierarchy is the only differentiating factor between concurrent kernel threads running on the GPU.

Memory Architecture

A critical driver during algorithm design is efficient use of the various memories available to the GPU cores. Inefficient or naive use of data memory can generate inferior performance, negating any benefit from GPU implementation. High Performance Computing (HPC) applications on the CPU may consider cache line sizes and other memory-centric optimizations during programming, but on the GPU, design of an algorithm with consideration to efficiencies and constraints imposed by the memory hierarchy is critical for performance.

Figure 4.13 shows the scope of the primary memories used by most GPGPU applications. Individual threads have access to registers (Figure 4.13(a)), which are 32-bit memory locations and are private to the specific thread. Registers are accessible in a single clock cycle, and reside in the register file of the local SM. The register file is finite, so if the collection of threads in flight within an SM requires more register resources than are available, usage spills into slower global memory (typically captured by the L1 cache in current devices).

Thread blocks, as described in Section 4.1.1, have access to on-chip "shared memory" resources of the multiprocessor (Figure 4.13(b)). Shared memory is consistent in terms of address space and content across all threads in a specific block, so provides the ability for communication and data sharing. The scope of shared memory is

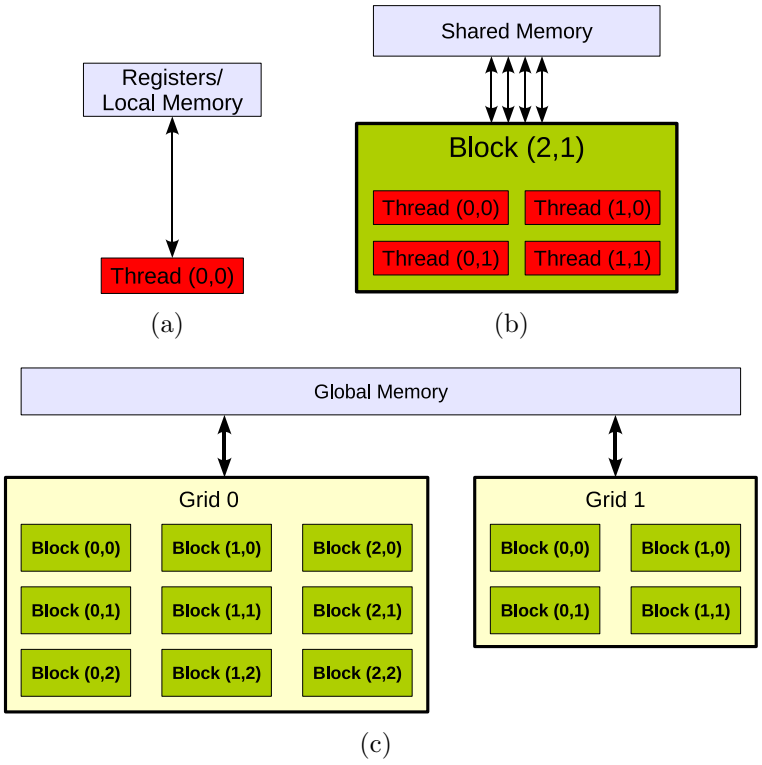


Figure 4.13: Inter-thread data sharing hierarchy in NVIDIA GF100 architecture for a single thread (a), a single thread block (b), and the block grid. Adapted from NVIDIA [113].

strictly restricted to a single thread block, and provides no communication ability between blocks or with the CPU. Even when multiple thread blocks are scheduled onto the same SM, each is given its own independent shared address space. Inter-thread communication and data sharing are key design considerations when determining thread block dimensions, and drive fundamental decisions in GPGPU algorithm design and implementation. From a practical perspective, shared memory appears as a user managed cache sitting between a thread block and global memory.

Threads within a block can communicate using shared memory, but communication between blocks requires an additional memory space known as the global memory (4.13(c)). Global memory is an off-chip DRAM resource, and provides a unified memory space that all threads on the GPU access. Global memory is also accessible by the CPU host system across the PCI-Express bus, and is therefore the primary means

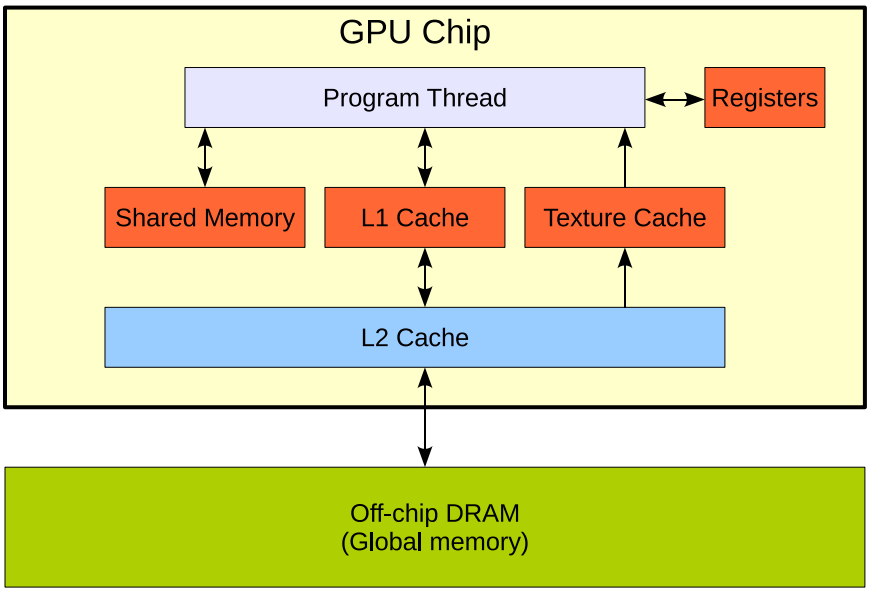


Figure 4.14: NVIDIA GF100 memory hierarchy from perspective of a single thread. Adapted from NVIDIA [113].

through which data is transferred between the CPU and GPU systems. Because it is an off-chip resource, global memory has significantly larger access latency (400-800 clock cycles [112]) than on-chip register or shared memory.

Additional memories are available to GPU kernels such as read-only constant and texture memory, with texture memory providing certain types of interpolation typically used by graphics engines. Further information about these memories and the subtle advantages that they provide in certain access scenarios is available in the NVIDIA literature [112].

The GF100 memory hierarchy is shown in Figure 4.14 from the perspective of cache structure. This structure has evolved from previous GPU generations, and provides greater performance for kernels that require irregular memory access patterns. A 768 KB unified L2 cache provides an interface to the external DRAM, and an L1 cache local to each multiprocessor increases performance for nonuniform memory access patterns and register spillage.

Many details of the memory systems in the GPU are not released by NVIDIA. Work to micro-benchmark specific aspects of the internal architecture such as memory latencies, pipeline depths, synchronization techniques, and cache details are presented

Table 4.2: GTX 480 Memory Summary. Adapted from NVIDIA [110].

Memory	Location	Cached	Read/Write	Scope	Lifetime
Register	On-chip	-	R/W	1 thread	Thread
Local	Off-chip	Yes	R/W	1 thread	Thread
Shared	On-chip	-	R/W	All threads in a block	Block
Global	Off-chip	Yes	R/W	All threads + CPU	Host
Constant	Off-chip	Yes	R	All threads + CPU	Host
Texture	Off-chip	Yes	R	All threads + CPU	Host

in, for example, [177, 170].

Table 4.2 summarizes the memory available to CUDA kernels, including their scopes and lifetimes.

Memory Access Patterns

Early GPU memories were designed to stream data in graphics rendering tasks, and not for more general access patterns such as gather/scatter common in general purpose computing. Recent GPUs have significantly relaxed the access requirements for efficient use of memory, but a number of constraints still remain. The primary driver behind memory access patterns is that physical memory is organized into banks (shared memory) or partitions (global memory). When threads in a warp need to simultaneously access parallel data that resides in a single physical bank, accesses to the memory may be serialized leading to loss of performance. Inefficient memory access patterns can immediately lead to an order of magnitude decrease in apparent GPU performance.

In the GF100 architecture, generalized L1 and L2 caches substantially simplify the access patterns required to utilize memory efficiently. Shared memory bank conflicts, however, should be explicitly avoided during programming. Global memory partition camping, likewise, can have an effect on overall kernel performance. Describing the details of meeting these requirements is beyond the scope of this thesis, so the reader is referred to [112, 115, 110]. Correct structuring of kernel and thread block layout, and techniques such as data padding and alignment modifications can produce large improvements in kernel performance.

Scalability

Multiprocessors form the basic computational block in the GF100 architecture, and are independent from one another except through the unified global memory. This independence is an important factor in performance scalability, as adding additional SMs to a system through graphics card updates directly scales the performance with number of multiprocessors, and requires no change to the CUDA code. Improvements in SM capability with new GPU revisions allow additional blocks to share a multiprocessor, providing a transparent increase in performance. Scalability across GPUs is simplified by the explicit decomposition of a CUDA program into thread blocks, which communicate through global memory. The explicit computational structure greatly enhances hardware scalability, as independent work units are clearly defined.

4.1.2 Applicability of GPGPU

Not all applications are suited for implementation on GPGPU hardware. Key attributes for GPU applications include arithmetic intensity, and ability to formulate the work processes in the paradigm of many independent SIMT threads. Comparisons of GPU, CPU, and programmable logic accelerators are explored in [22, 86]. An analysis of CPU and GPU performance for comparable power footprints is performed by Vuduc et al. [171], with the interesting conclusion that for "moderately irregular" compute patterns, GPU performance is comparable to addition of one or two multicore CPU sockets. The results were based on an NVIDIA GTX 285 in which memory access pattern requirement were strict, unlike the significantly relaxed requirements present in the GTX 480 (GF100) architecture. Cache improvements in the current generation of NVIDIA GPU are designed to increase performance for irregular memory access patterns, so should affect the Vuduc et al. conclusions.

Algorithm suitability for parallel GPU implementation is considered in [68], with some algorithms characteristics identified such as ability to separate the workload into independent work threads. Profiling based upon the intermediate level CUDA Parallel Thread Execution (PTX) pseudo-assembly language [114] is considered in [23], with the intent to simplify performance tuning. Aside from the NVIDIA documentation on performance tuning, literature such as [127, 85] consider techniques for specific

applications on the GTX 480. Use of GPUs for image processing and computer vision tasks is considered by [48] as a general inverse rendering problem, [121, 180, 21] for motion tracking, [101] for image edge detection, and [122, 79] for optical flow.

The GF100 introduced IEEE compliant floating point arithmetic units, and enabled double precision processing. For many HPC applications, lack of double precision floating point was a major obstacle in past device architectures. Although double precision is currently supported, there is a performance penalty incurred compared with single precision computation. In the professional GPU line based on the GF100 architecture, such as the NVIDIA Tesla C2050, double precision computation achieves half the throughput of single precision. In the commercial graphics product line (GTX 480), double precision performance is reduced to 1/8 of the single precision throughput. For applications strictly requiring double precision math, this is nearly an order of magnitude performance penalty.

Considerations for GPGPU Suitability

A multiprocessor can contain up to 1536 threads using CUDA compute capability 2.0, allowing $15 \times 1536 = 23040$ threads to be in flight on a GTX 480 GPU. Compared with the smaller number of actual processor cores (480 cores), the schedulers have the ability to hide long latency operations such as external DRAM fetches through hardware multithreading.

Cache-miss global memory latency is on the order of 400-800 clock cycles. Hiding this latency through multithreading would require significantly more warps than the 48 supported by a multiprocessor. For this reason, GPUs are well suited for applications with high arithmetic intensity, meaning that for every off-chip memory access, there should be multiple local arithmetic operations performed. In current hardware, NVIDIA reports that if each external memory access is accompanied by 15 local arithmetic operations, then about 40 warps are required to effectively hide a global memory latency of 600 cycles [112]. For the true range of 400-800 cycles, the number of required blocks therefore ranges from 27-53 concurrent warps. It is apparent that if arithmetic intensity drops below 15 arithmetic instructions per global memory access, the multiprocessors resources are forced to idle while external memory fetches take

place. The L2 cache appears to address this issue for some workloads, but independent stream processing by the SMs may constantly require fresh cache-miss input. Arithmetic intensity is thus a critical element of algorithm suitability for GPGPU implementation, with the required level of intensity dependent on the application specific memory access patterns.

The most frequent reason that a CUDA thread must wait before execution is that input data is not available [112]. The wait time depends on the source of the pending data, and for efficient use of GPU resources, there should be sufficient concurrent threads so that non-blocked threads can perform work while others wait for data. For stalls caused by register dependence, a thread must wait for a previous instruction to produce output. The current execution time to complete an instruction is typically 22 clock cycles [112], so presence of at least 22 warps in a multiprocessor is required to effectively hide register dependence stalls.

Another major cause of stalled threads is the presence of synchronization and memory fence calls within the kernel. A synchronization barrier causes threads within a block to stall until every thread has reached the same barrier, and is primarily used to avoid parallel data hazards in shared memory. To keep the SM compute resources busy while some threads are stalled at a pending barrier, multiple thread blocks should be present in the single SM. Synchronization barriers are local to a thread block, so the presence of multiple blocks can allow the scheduler to reduce idling inefficiencies.

Even with high arithmetic intensity, a multiprocessor scheduler requires concurrent warps to efficiently hide latencies in the presence of data dependencies. There is a complex trade-off required during GPGPU implementation, as additional warps in a thread block help to hide latency, but also require additional processor memory resources. To allow multiple thread blocks to coexist within a single SM (helping to hide synchronization stalls), no thread block should use the majority of multiprocessor registers or shared memory. On the other hand, large scale concurrency within a thread block helps to hide global memory access times through arithmetic intensity, so larger block sizes and associated memory resources are often desired. Performance tuning by changing block sizes and memory utilization is typically required to achieve adequate performance from the GPU.

When considering GPU-based acceleration in the context of an algorithm, it is

important to incorporate Amdahl's law into the analysis. The GPU is not suited for all workloads, so some processing must typically be performed on the CPU. If acceleration by the GPU does not affect the entire execution time, implementation effort should be compared to overall speedup. Memory transfers between the CPU host system and GPU over the PCIe bus have lower bandwidth than local memory accesses, so it is often advantageous to perform some level of inefficient computation on the GPU to avoid data transfer between platforms.

GPGPU for Scale-Space Ridge Extraction

Scale-space ridge measurement can extract imaged grid lines in the presence of changing target range and DOF blur, as described by Section 3.3. The need to calculate and interpolate metric values at multiple scale levels produces a load that is not well matched to commodity CPU systems at video frame rates. GPGPU acceleration of the scale-space ridge detection algorithm is described in the remainder of this chapter. The GPGPU approach is selected for three primary reasons.

1. **Commodity hardware:** Other approaches to acceleration such as multiprocessor platforms or FPGA accelerators are not as easy to procure or maintain as commodity GPU cards. In terms of raw computational potential per dollar, GPUs have substantial advantage over CPU systems.

2. **Efficient mapping to GPU architecture:** Scale-space ridge extraction performs arithmetically intense computations in well defined spatial neighbourhoods. The workload can therefore be divided into independent processing blocks that map efficiently into GPGPU hardware.

3. **Scalability:** A GPU implementation scales to higher resolution cameras for improved accuracy, and higher frame rates for increased scanning speed (faster camera motion). Additional graphics cards or improved multiprocessors immediately lead to speedup, without software rework. Adding or upgrading GPUs in an industrial system is typically easier than a CPU upgrade, as the GPU and its supporting hardware are independent peripherals not tied to operating system execution. Multiple GPU cards can coexist in current systems.

4.2 Parallel Framework and Implementation

4.2.1 Algorithm Sequencing and Flow

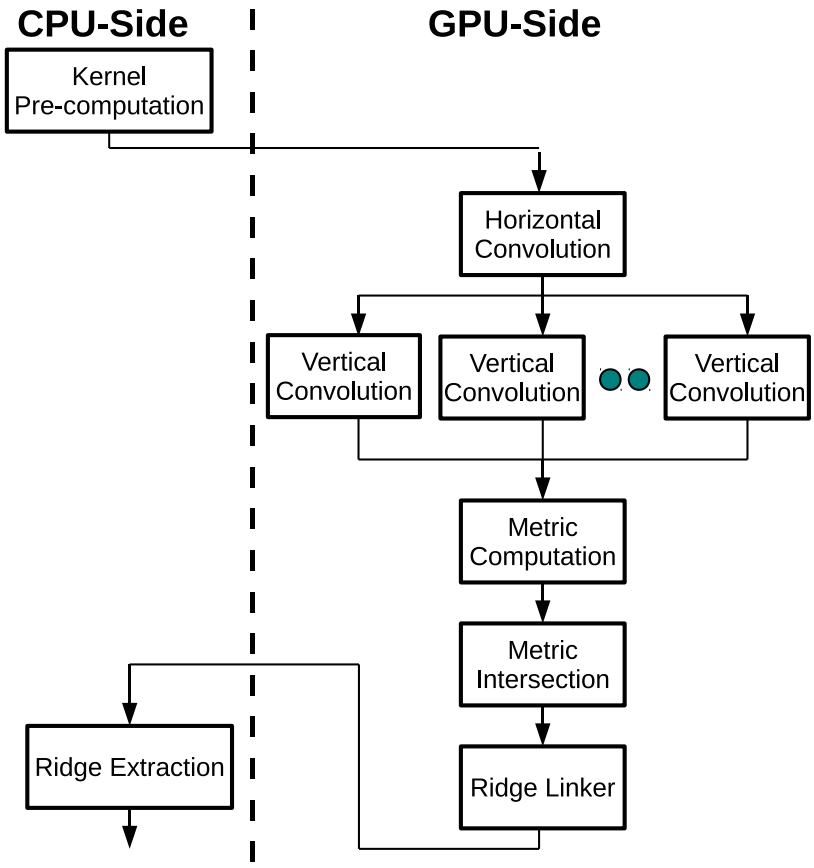


Figure 4.15: Division of work flow between CPU and GPU for grid line extraction (reproduced from Kinsner et al. [73]).

Not all workloads are suited for GPGPU implementation, primarily because of the desired arithmetic intensity and memory access patterns. For this reason, pre- and post-processing of data on the more versatile CPU can enhance overall system performance. Figure 4.15 shows the proposed work division and processing flow for grid intersection measurement. The CPU controls program sequencing, beginning with calculation of the convolution kernels for scale-space generation. The CPU then transfers a video frame to global memory on the GPU, and dispatches a sequence of

kernels to process the data. Upon completion of the ridge linker kernel, the resultant data is transferred back to host system RAM where the CPU performs final processing to form variable length ridges in memory, and to complete the grid intersection calculations.

4.2.2 GPGPU Processing

Scale-space Generation

The first stage of processing is generation of a scale-space image representation, which is later used by the feature detector. Scale-space formation requires convolution with a generating kernel such as (3.8), as described in Section 3.3. The two-dimensional convolution is separable, so horizontal and vertical phases are employed to improve efficiency while loading data from global memory into the multiprocessor shared memories. The convolution operations produces both horizontally and vertically filtered data at multiple scale levels.

To provide a generic approach that can be applied to varying scale levels in different applications, an intermediate convolution kernel is first produced. Convolution of data with this intermediate kernel produces a filter operation with variance of 1 pixel^2, and is thus named the "unit step kernel". The original kernel (3.8) produces a filter variance of $\Delta t = \frac{1}{3}$, so self convolution of this kernel with itself is used to produce the unit step kernel which has a larger support radius than the original.

To obtain an image scale-space representation with scale levels separated by more than 1 pixel^2, the unit step kernel is progressively applied through repeated convolution. Although this leads to additional computational effort and compounding of single precision floating point errors, it allows a single kernel to be used more generically, and simplifies the requirements on border apron data where the kernel extends beyond the active data set near the boundaries. Repeated unit step kernel convolution produces the effect of a single application of a larger kernel that directly provides the necessary interscale variance, called the interscale kernel.

An alternative to repeated convolution with the unit step kernel is to produce a larger support kernel by self-convolving the unit kernel the necessary number of times, and then applying the larger support kernel to the data in a single step. For

repeated application of the unit step kernel, the number of accumulate operations per output pixel is $k(2r+1)$ where r is the radius of the step kernel, and k is the number of step kernel applications to form the next scale level. The number of required multiplications is slightly less in the case of the larger support kernel, requiring $2kr+1$ multiplications, not including the amortized operations to form the kernel itself. For an example filtering scale of $t = 15$ pixels^2, the number of accumulation are 105 and 91 respectively. This tradeoff is deemed acceptable for efficiency introduced by controlled border apron usage (described later).

In a single precision floating point GPU, it is attractive to form the larger support kernel using higher precision (on a CPU for example) and apply it only once in single precision, rather than iteratively accruing and compounding single precision arithmetic errors during repeated convolution. The disadvantage of large support kernel application is that the size of the kernel depends on the scale differential at which it is being applied, and hence the GPU instructions that perform the convolution must be capable of handling kernels of variable size. To reduce GPU code complexity and allow for better optimization, the approach of repeated small kernel application is used, with intermediate data stored in single cycle shared memory and all shared accesses coherent.

The convolution work flow required to generate a complete scale-space representation is shown in Figure 4.16. Arrows represent the flow of data, and the notation nx denotes n applications of the interscale kernel, each application of which is composed of m repeated applications of the unit step kernel. Image data is loaded from global memory and horizontally convolved with the unit step kernel repeatedly until horizontal filtering at a required scale level has been achieved. This data then takes two paths: (1) it is stored to global memory as a horizontally convolved scale slice, and (2) the data is passed back into the horizontal convolution kernel where it receives additional horizontal filtering to achieve the next larger horizontal scale slice. When all horizontal convolutions are complete, the vertical convolution kernel loads the horizontal output from a specific scale slice (from global memory), and performs the necessary number of vertical convolutions with the unit step kernel to achieve the required output scale.

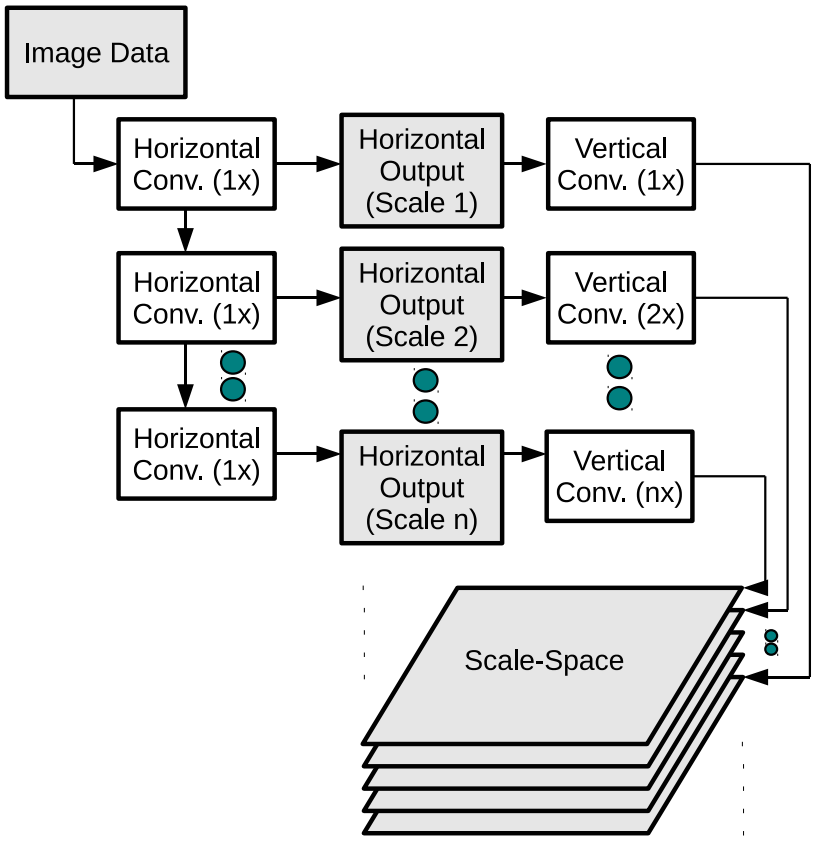


Figure 4.16: Convolution for the generation of an n-level scale-space. nx denotes n applications of the interscale kernel, each application of which is composed of m repeated applications of the unit step kernel. Grey boxes indicate global memory storage on the GPU. Figure reproduced from Kinsner et al. [73].

Horizontal Convolution Kernel

The horizontal convolution phase begins with the initial image data, and repeatedly convolves this data with the unit step kernel, thereby defining the horizontal filtering operation between any two scale levels. After each set of convolutions (new scale level), the filtered data is saved to global memory, forming an array of partially filtered images. Only a single load of the original data from global memory is required to generate all of the horizontally filtered scales, followed by a write-back of the image size for each scale level. The intermediate data between horizontal filtering stages remains resident in the multiprocessor shared memory, avoiding unnecessary global

memory accesses.

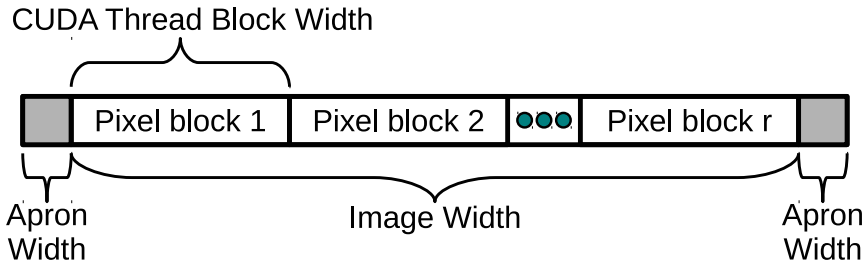


Figure 4.17: Horizontal convolution kernel operating on a single row of the image. A thread block has fewer threads than the image width has pixels, so it is repeatedly applied to the image row. A single thread therefore processes multiple pixels in the image row, improving efficiency.

Figure 4.17 shows the processing strategy used to horizontally convolve each image row. An isolated block of threads is assigned to process each row of pixels in the image, requiring M thread blocks to process an image with M rows. To optimize register overhead and to pipeline global memory accesses, a single thread loads and performs convolution upon multiple (r) pixels within a row. The compiler is directed to completely unroll the r iteration loop defining a thread's work on multiple pixels, removing unnecessary loop overhead. Even with a large unrolling factor, the pixels handled by iterations of the worker thread are completely independent, so the optimizing compiler or thread scheduler can modify the work flow if necessary to control register usage. The number of threads in a thread block for an $M \times N$ image is therefore $\frac{M}{r}$, where r is the repeat work factor of a single thread, and where r is ideally chosen such that the number of threads is an integral number of warps.

Convolution near the image edges requires pixel data outside of the active image region, so border apron data outside of the image boundary is replicated within shared memory to eliminate divergent branching and artifacts. Use of the unit step kernel bounds the number of pixels required in the border apron, but to minimize idle threads and simplify addressing arithmetic, the apron is defined as a half-warp in size. Larger support kernels would require a larger apron size, which should ideally be set to an integral number of warps.

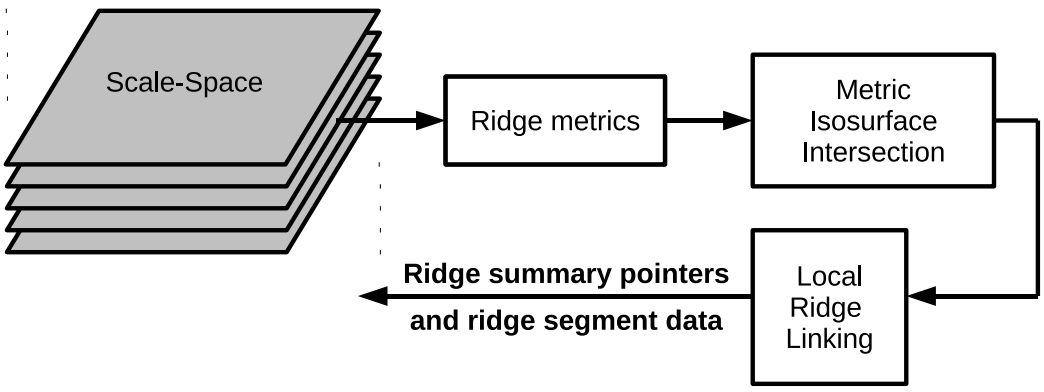


Figure 4.18: GPU-side ridge extraction flow from scale-space to ridge output. Reproduced from Kinsner et al. [74].

Vertical Convolution Kernel

The horizontally convolved image data at each scale level is transferred from global memory column-wise. Vertical convolution is performed in a similar manner to horizontal, including the handling of border aprons. The horizontally convolved data from earlier GPU processing is treated separately at each scale by repeated application of the vertical step kernel, as shown by Figure 4.16.

Output from the full convolution at each scale level is stored to global memory with an artificial image border of 16 pixels around the filtered image periphery. This storage allows subsequent kernels to access pixel data coordinates slightly outside of the image border (for derivative approximations) without risk of memory access violations on the GPU, removing the need for divergent data border checking logic. The output stored to global memory comprises the fully filtered scale-space image representation. Figure 4.18 shows the remaining GPU-side processing required to form ridge output from the scale-space representation in global memory.

Ridge Metric Computation Kernel

The metric computation kernel performs calculations at each pixel and scale level. The computations are based on Equation (3.10), and derivative approximations are generated through central difference approximation with coefficients pre-computed and stored in constant memory. Since the derivative operators utilize data in the

neighbourhood around each pixel, a border region is loaded into shared memory with radius large enough to support the highest order derivative approximation required. Multiprocessor shared memory is not large enough to hold an entire image or the complete scale-space, so the work must be broken into multiple thread blocks which each handle a portion of the data.

Thread block sizing involves a trade-off between shared memory usage and redundant loading of apron data. The derivative approximations for a pixel require a two-dimensional neighbourhood of scale-space data to be available, so for small thread block sizes, the apron is a significant portion of the data loaded. For the ridge detector implementation, a 6 pixel border apron is required. Pixels closer to the image edge than the apron size load mirrored data from the boundary apron stored to global memory by the convolution kernels. The resultant boundary metric values are invalid, and are rejected as noise later during ridge formation.

Early versions of the CUDA compiler were unable to optimize register usage in the metric computations, producing kernels that could not be executed because of register overuse. The solution was to manually decompose the metric equations into simpler operations, and to place synchronization barriers occasionally in the kernel. The current generation of CUDA compilers have improved optimization capability, and combined with increased register resources in the multiprocessors, can directly compile many term equations into executable code that fits within the SM resources.

The results from Equation (3.10) at each pixel are stored to global memory. The two conditions in the equation that must be zero are stored as floating point data, allowing interpolation of the zero isosurfaces in a later kernel. The remaining two terms of the equation, which simply check for negative curvature, are not stored in memory. Only the sign of the computation is required for later use, so the sign of each of the two terms is encoded using flag bits in a single byte data primitive, saving three storage bytes per pixel (compared with single precision floating point).

Metric Isosurface Intersection Kernel

The metric intersection process is summarized in Algorithm 4.1. The zeros of the isosurfaces (defined by Equation 3.10) are linearly interpolated along the edges of each surface interpolating cube. Where multiple zeros exist on the edges of a single

Algorithm 4.1 Ridge Metric Intersection Kernel

```
Load Data
for Voxels in parallel do
    for Three edges originating at interpolating cube origin vertex do
        [1] Check for zero crossing of each metric (differing sign at edge vertices)
        [2] Linearly interpolate zero crossings on edge
        [3] Block on completion of neighbouring cube zero interpolations
    end for
    for Three faces of cube common to origin vertex do
        if Exactly two metric zeros on edges then
            Consider the line joining the two zeros
            if Both metrics have isosurface intersections on face then
                if The metric isosurfaces intersect in a single point then
                    if Intersect is within the face then
                        [1] Record ridge intersection with face as i,j,k in local coordinate system
                        [2] Set the relevant bit in the cube fingerprint
                    end if
                end if
            end if
        end if
    end for
end for
```

face, they are joined with lines. If interpolating lines connect both metrics, and if these lines intersect on a cube face, then this is an isosurface intersection point. The algorithm is an extension of the classic marching cubes process.

To avoid duplicate computation, a single thread on the GPU performs zero point interpolation for three edges of an interpolating cube (edges which end in a common cube vertex), as shown in Figure 4.19(a). The remaining edges are processed by neighbouring threads, and because the relative order of thread execution is not guaranteed on CUDA hardware, synchronization ensures that all threads have processed and stored their associated data before interthread sharing is allowed. To avoid divergent branching within a thread block, all threads follow similar execution paths regardless of input, and decisions are made through multiplication of values with logical comparison results.

Table 4.3: Conventions used in interpolation cube intersection fingerprint

Face pierced	Scale-space axis	Fingerprint bitmask
F1	-Z	0x01
F2	-Y	0x02
F3	-X	0x04
F4	+X	0x08
F5	+Y	0x10
F6	+Z	0x20

For each metric, a zero is considered to intercept the edge if the values at opposing vertices have different signs. When a metric zero is present, the location of that zero is linearly interpolated along the edge. Results from the three edges are stored to shared memory, and the thread waits until all neighbours have completed their work. From any metric zero points on the cube edges, a thread then interpolates intersection locations on the three faces associated with that cube (F1-F3 in Figure 4.19(b)), and waits until all neighbours have also finished. Where two intersection points exist among the six faces of an interpolating cube, these points are joined with a straight line, and the result is recorded as a scale-space ridge fragment.

Data output volume plays a significant role in this kernel since unnecessary data storage leads to additional transfers to and from global memory. To reduce the memory overhead, a maximum of two face intersections are stored for the three faces being processed by each thread. If more than two intersections are found among the three faces, then an ambiguous situation is present and no ridge data is recorded. The storage associated with a single interpolation cube comprises a fingerprint that denotes on which faces intersections have occurred, and global data that stores the cube i, j, k data for the position of each intersection on the relevant faces. The intersection fingerprint is a single byte associated with each interpolation cube, in which individual bits flag ridge segment intersections with a specific cube face. The fingerprint follows the bit convention of Table 4.3, and allows later kernels to use a byte-wise lookup table to quickly determine how many and which faces have ridges passing through them.

Figure 4.20 shows the raw ridge segment output from the metric intersection kernel. In the images, colour and height indicate the scale at which a ridge was

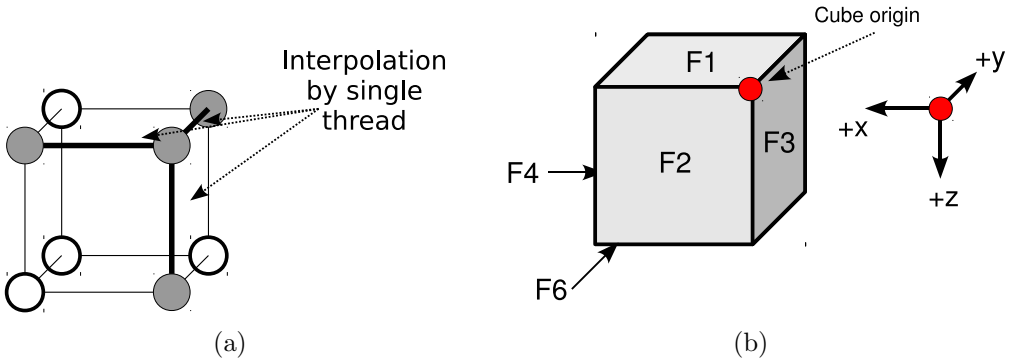


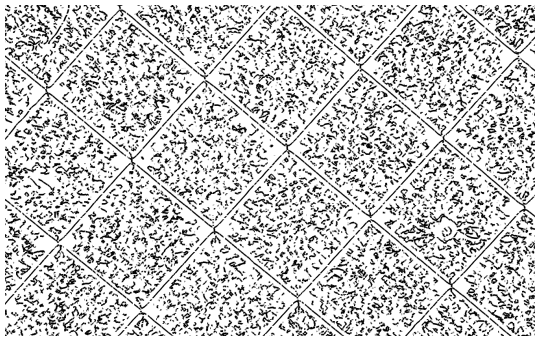
Figure 4.19: Metric isosurface intersection kernel. (a) Interpolating cube edges generated by a single thread (figure adapted from Kinsner et al. [73]). (b) Interpolating cube labelling convention.

detected. Noise is visible between the grid lines where no valid ridge data is present.

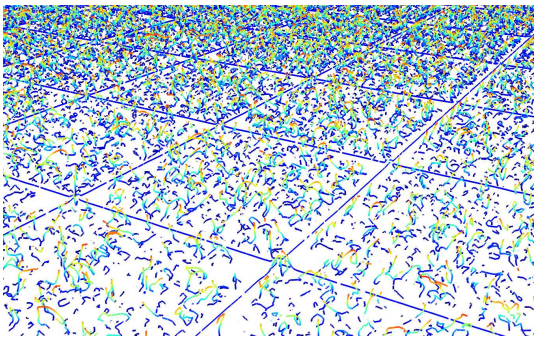
4.2.3 Ridge Linking Kernel

The ridge segments defined between interpolating cube faces must be traced throughout the scale-space and linked together, forming the output set of full length ridges that will be used to interpolate grid line intersection points. Ridge detection is based on image data, so the output does not conform to preset GPU thread block sizes or memory coalescence requirements. The CPU is better suited to forming the variable length output ridges, but visiting every interpolation cube in the scale-space to check for valid segments is a large search space. For an 1024×768 pixel image with 8 scale levels, there are over 5.5 million cubes to examine. At 15 video frames per second, more than 82 million interpolation cubes must be visited and processed per second just to locate ridges, not including post-processing.

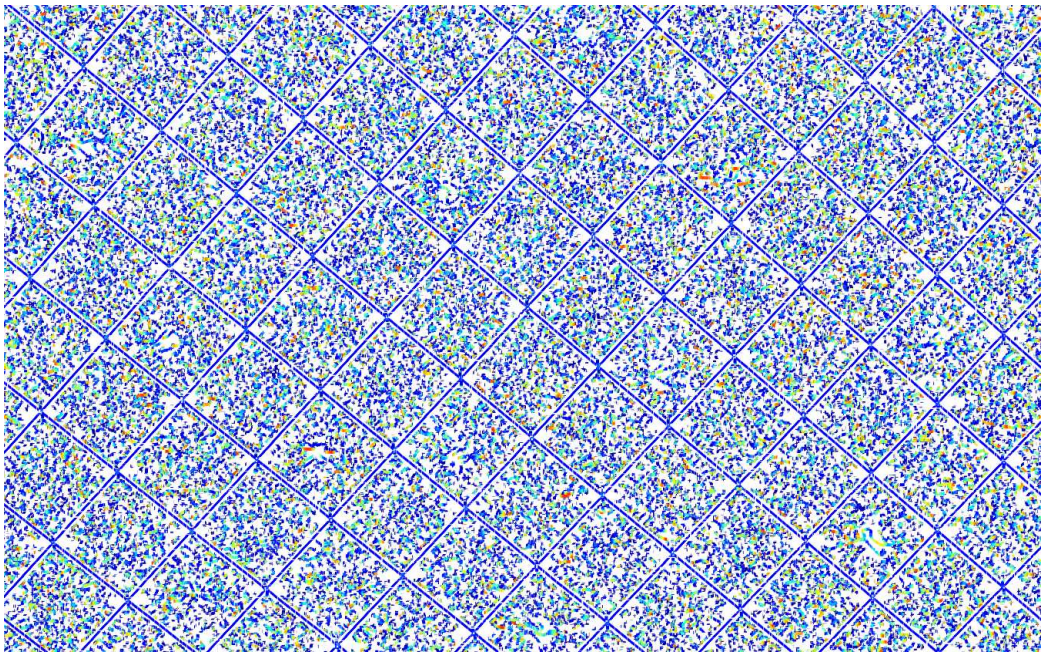
To simplify the task of producing ridge output on the CPU, a GPU-side ridge linking kernel examines the entire set of interpolation cubes, and generates summary output that the CPU uses to rapidly locate valid ridges. The summary data must be traversed by the CPU to determine where valid ridges start, but the search space is reduced compared with the raw intersection data. In the current implementation, only $\frac{1}{4}$ the number of data elements need be visited by the CPU compared with the complete space. More significantly, the summary data elements require very little



(a) Ridge segments - straight grid lines and unfiltered noise between the lines



(b) Ridge segments - height representing the scale-space scale at which a segment was detected



(c) Ridge segments - straight grid lines and unfiltered noise between the lines

Figure 4.20: Ridge output from a sample image, with colour and height indicating the detection scale level. Unfiltered noise is detected between the straight grid lines. Reproduced from Kinsner et al. [74].

logic to identify valid ridges, as opposed to the logic needed when visiting the original data. A key assumption in the summary process is that ridges are long compared with the thread block size used by the summary kernel. With this assumption, valid ridges span more than one thread block, and therefore penetrate the boundary/sides of a block that processes a portion of the interpolating cube set. Ridges that do not span multiple thread blocks are too short to be valid grid lines, and these short ridges can therefore be rejected by the kernel. Such short ridges are caused by, for example, small scratches or dirt on the imaged surface. For reference, thread blocks in the implementation are 8×8 pixels ($\times 7$ scale steps), whereas the distance between grid lines in an image is one the order of 100 pixels. A valid ridge should therefore traverse at least 10 thread blocks between intersections with perpendicular grid lines.

The ridge linking kernel examines blocks of interpolating cubes and determines which boundary faces (periphery of the block) have ridges passing through them. Any identified ridges are then traced through the block to determine whether they continue to another block boundary, or terminate internally. Summary data is then generated on the border, and is later used by the CPU to identify where ridges begin and end. The border data is comprised of a partitioned array of structure primitives (int2) containing integer values. A logical partition separates the array into a row space and a column space, with a one-to-one mapping between the thread block border faces and the border summary data.

In the GPGPU implementation, the ridge linker operates on blocks of size $8 \times 8 \times 7$ to control shared memory usage. Figure 4.21(a) shows the border data as it surrounds a single block of interpolation cubes at a single scale slice. There are many blocks of cubes to be processed, so the border data forms a grid pattern separating them, with each exit from a block having associated with it a unique border data location. The organization of the data in global GPU memory is shown in Figure 4.21(b), and Table 4.4 presents the storage convention used within the border data output. The storage convention allows the CPU to rapidly traverse the data set and identify valid ridge start points within a block, from which the algorithm can trace out and record the complete ridge path. Within Table 4.4, whenever a value is positive the magnitude specifies the global address of the ridge start location. Each interpolating cube in the space has a unique address, organized in row major form.

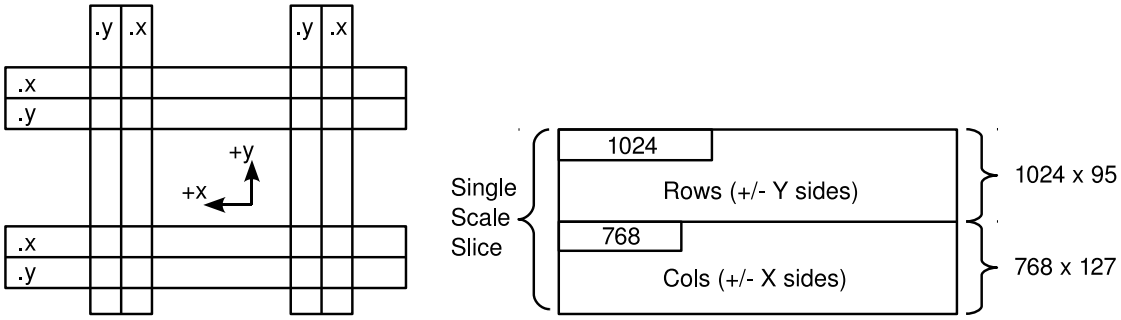


Figure 4.21: Linker kernel data storage conventions.

Table 4.4: Conventions used in Border Summary Data

X	Y	Interpretation
0	0	No interesting events
+	0	Ridge starts on boundary and ends within block
-	0	Ridge starts on boundary, continues into far block
+	+	Ridge starts and ends within the adjacent blocks
+	-	Ridge begins in neighbour, continues into far block
-	-	Ridge passes through these blocks

4.2.4 CPU Post-processing

Variable Length Ridge Output

Upon completion of the GPU-side ridge linking and border data summary process, data is transferred from the GPU global memory to the host system RAM. The CPU then performs multiple post-processing stages, which lead to finished grid line intersection estimates.

The first post-processing stage is formation of complete ridges in memory. The ridges are of variable length, and consist of sequences of three-dimensional points through the scale-space, defined by interpolated intersections with the faces of the interpolation cubes (ridge intersection kernel). Extraction of the variable length ridges starts with traversal of the border summary data. From the border data content and the conventions of Table 4.4, the CPU can rapidly identify the global interpolating cube address of a ridge endpoint.

From an endpoint, a tracing process begins to produce the full ridge in memory. The algorithm iteratively follows the segments through the interpolating cube space, storing new points as they are encountered to a list. As the follower process passes through borders that defined the linking process on the GPU, the algorithm locates the corresponding summary data element, and clears it to avoid later instantiation of the tracer algorithm on the already traversed data. Ridge tracing continues until an invalid or ambiguous ridge segment is encountered, or until the global scale-space boundary is reached.

Storage of the variable length ridges is through a global point list in memory. An associated array of structures records the bounding addresses of the ridge in the point list, as well as statistical data such as length an approximated slope.

Ridge Filtering

Only a small number of ridges extracted from the scale-space correspond to true grid lines in the image. Most ridges are caused by imperfections in the imaged metal surface, or simply noise where no valid ridge exists. Figure 4.22 shows the collection of ridge segments from a portion of a typical grid image. The grid lines as well as extracted noise between them (where there is no valid ridge signal) are visible. Many

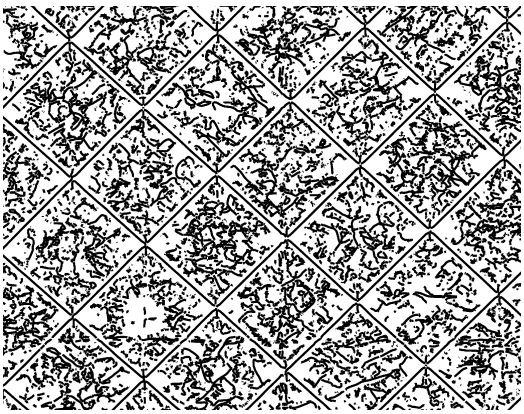


Figure 4.22: Raw ridges extracted from a portion of an image.

of the noise ridges are suppressed by the ridge linker kernel which requires crossing of a block border, but some remain. True grid lines are relatively straight in the image, and because the scale-space is a three-dimensional representation, consistency of detection scale can be be used for filtering. Sharp changes in either direction of a ridge, or in the detected scale indicate that a ridge is not a grid line, so spatial filtering removes most noise data. A final filtering stage, based on ridge length, rejects short segments.

A key consideration in grid line extraction is that where perpendicular lines intersect, a valid intensity ridge does not exist (trifurcation occurs). At these points, a ridge in the scale-space either terminates, or continues through the intersection but at a much larger detection scale. Filtering using detection-scale gradient naturally breaks the grid lines at these sharp changes of scale, providing segments that can be used for intersection estimation.

4.3 Parabola Fitting and Intersection Estimation

From ridges detected around a grid line intersection, polynomials are fit to the pairs of ridges belonging to the same grid line, but on opposite sides of an intersection. The polynomials may then be intersected to interpolate a subpixel location of the true intersection point. The ridge data that is used to define the polynomial is dense, with approximately 50-100 data points on each side of the intersection (one point from

each interpolating cube face intersection). This parameterized approach reduces the effect of small errors in the ridge points, which are introduced by linear interpolation on the faces of the interpolating cubes, and also by floating point errors in the metric computations.

Before a parabola can be fit to pairs of ridges surrounding an intersection, the correspondence between ridges must be determined. To accomplish this, a list of all filtered endpoints is formed, and candidate ridge groupings are created based upon spatial endpoint proximity. The approximated slopes of the ridges are then used to pair corresponding segments for parabola fitting.

There is flexibility in the choice of grid line parameterization order and fitting technique. Second-order parameterization was used because the grid spacing is small relative to the smooth curvatures on the part surfaces being measured. In the sheet metal forming application, the original grid dimensions are chosen based on maximum part curvatures that are known in advance. Surface curvatures higher than second order between grid lines are not physically plausible with correct choice of grid scaling, and if they occur, do not meet the homogeneous deformation requirement of the surface strain calculation technique.

A variety of techniques can be used to fit and intersect parabolas. Mitchell [106] considers some methods, including the iterative fitting process described by Ahn et al. [3]. A least squares fitting and conic section-based intersection approach is described in this section.

Least squares fitting based on singular value decomposition (SVD) can be used to minimize the error energy between a function and data. To simplify subsequent parabola intersection, the least squares fit can be performed in the image coordinate system. A parabola, defined as $P(x) = c_2x^2 + c_1x + c_0$, is fit to the data to determine the coefficients c_i. The parabolas $P_1(x)$ and $P_2(x)$ can then be equated as $P_1(x) = P_2(x)$. Solving for the x coordinate, the coefficients are simply subtracted to form a new quadratic equation, and the roots found. The C-based GNU Science Library [50] can be used to perform least squares fitting and root finding. The solution in a region bounded by the original data points is taken as the estimated intersection.

The disadvantage of fitting parabolas in the image coordinate system is that the least square errors may be non-orthogonal to the function. To solve this problem,

the parabola can be fit to the data in a local coordinate system. Practical grid line curvatures are small across a single intersection distance, so by fitting the parabola to the data in a rotated coordinate system, least squares approaches approximate orthogonal error terms. Intersecting parabolas defined in different coordinate systems, however, is more complex.

A local coordinate system (x_p, y_p) can be defined, with the x_p axis aligned in the principal direction of the data to be fit. Within this system, a parabola is expressed as:

$$(x - x_c)^2 = p(y - y_c) \tag{4.11}$$

where (x_c, y_c) define the location of the parabola vertex, and p the focus distance. An angle θ determines the rotation between the image and local parabola coordinate systems. Using θ, the image-based coordinates of the parabola vertex (X_c, Y_c) can be determined. The four parameters p, (X_c, Y_c), and θ completely define a general parabola in the image coordinate system [3].

The parabola parameters may be determined from the ridge data through least squares fitting. The angle θ is estimated from a linear fit of the data, and the points then rotated into the local coordinate system (x_p, y_p). A parabola $P(x) = c_2x^2 + c_1x + c_0$ is then least squares fit to the data. From expansion of Equation 4.11 into the form of $P(x)$, the parabola parameters can be recovered as:

$$p = \frac{1}{c_2} \tag{4.12}$$

$$x_c = \frac{-pc_1}{2} \tag{4.13}$$

$$y_c = c_0 - \frac{x_c^2}{p} \tag{4.14}$$

A normalized error bound is used to reject mismatched or malformed grid lines, calculated as the sum of squares of the residuals normalized to the number of data points used in the fit.

To express two parabolas in a common coordinate system (in preparation for intersection), the transform between the parabola coordinate system and the image system is incorporated into Equation 4.11. This expression can then be rearranged

into the form of a conic section:

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_0 & b_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + c = 0 \tag{4.15}$$

Using the notation C_θ and S_θ to represent $cos(\theta)$ and $sin(\theta)$, respectively, the conic section parameters are simplified as:

$$a_{00} = C_\theta^2 \tag{4.16}$$
$$a_{11} = S_\theta^2 \tag{4.17}$$
$$a_{01} = a_{01} = C_\theta S_\theta \tag{4.18}$$
$$b_0 = pS_\theta - 2C_\theta^2 X_c - 2C_\theta S_\theta Y_c \tag{4.19}$$
$$b_1 = -2C_\theta S_\theta X_c - 2S_\theta^2 Y_c - pC_\theta \tag{4.20}$$
$$c = C_\theta^2 X_c^2 + S_\theta^2 Y_c^2 + 2C_\theta S_\theta X_c Y_c - pS_\theta X_c + pC_\theta Y_c \tag{4.21}$$

These parameters are calculated for both parabolas to be intersected, and a non-linear search is then performed to find the image coordinate point that satisfies both conic sections. The search is initialized at the mean image coordinate of the relevant ridge data. The solution point is taken as the sub-pixel grid line intersection location.

4.4 Results

Results from a full implementation of the GPGPU-accelerated grid line intersection measurement process are presented. Three classes of results are described: (1) Timing and GPGPU resource utilization results, (2) intersection measurement accuracy from synthetic data sequences, and (3) results from real-world video of gridded sheet metal targets.

Table 4.5: NVIDIA GTX480 Details

Model	NVIDIA GTX480
Architecture	GF100
Processor cores	480
Transistors	3.0 billion
Process technology	40 nm
Memory	1.5 GB GDDR5
Memory clock	1848 MHz
Processor clock	1401 MHz
Memory interface width	384 bit

Table 4.6: Kernel Execution Time (single frame)

Kernel Name	GPU Time [ms]
Horizontal Convolution	2.28
Vertical Convolution	3.97
Metric Computation	15.55
Metric Intersection	21.71
Ridge Linker	2.32
Total	45.83

4.4.1 Test Platform

The experiments were performed with an NVIDIA GeForce GTX 480 graphics card in a host system containing two Intel Core2 Duo (E8400) processors at 3GHz, 4GB of RAM, and running CUDA version 3.0 under Ubuntu Linux 9.04. The test sequences were 8-bit grayscale video data from a Point Grey Research Dragonfly camera, 1024×768 pixels in size at 15 fps. Details of the camera were provided in Section 2.3.2. Table 4.5 contains information about the NVIDIA GTX 480 GPU.

4.4.2 Timing and Resource Utilization Results

Timing Results

The execution times of each kernel, as measured by the GPU performance counters for a typical image, are presented in Table 4.6. The memory transfer time may be

Table 4.7: Kernel Block Efficiency

Kernel Name	Occupancy	Divergent Branches	Instructions Executed
Horizontal Convolution	100%	51	1375816
Vertical Convolution	100%	1038	1538496
Metric Computation	33%	208	15091030
Metric Intersection	58%	169894	20333403
Ridge Linker	88%	28020	1962479

hidden by asynchronously streaming data to and from the graphics device in parallel with execution, so is not included. Data transfer volumes are small compared with the PCI Express bus bandwidth. Across 50 image frames in a typical video sequence, the average frame processing time was 47.39 mS (21.1 fps) with a standard deviation of 2.05 ms. This timing is consistent across thousands of video frames processed. The small increase over the total in Table 4.6 is from overhead between kernel calls.

GPU Code Performance

Table 4.7 presents the CUDA profiler output with respect to occupancy, divergent branch performance, and instructions executed. All of the kernels exhibit some divergent branching because the task could not be formulated for fully coherent memory accesses (primarily while handling border apron data). Refinement of the kernels was used to reduce incoherent memory access patterns, while at the same time maintaining overall efficiency by reducing complex memory arithmetic and divergent conditions. Some techniques used during kernel refinement included: (1) constant memory lookup tables (cached) to reduce the number of repetitive online computation totals when precomputation was feasible; (2) processing of multiple pixels per computational thread to amortize data load overhead; (3) block grid and thread block size optimization to improve the utilization and hide memory latencies within each multiprocessor.

Resource utilization within the GPU is summarized in Table 4.8. Note that the local memory requirement of zero has been met for all kernels, indicating that multiprocessor memory is sufficient for the kernels as designed, thereby avoiding accesses to higher latency “local” overflow memory.

Table 4.8: Total Resource Utilization

Method	Registers	Shared Mem.	Local Mem.
Horizontal Conv.	7	4224 bytes	0
Vertical Conv.	17	3208 bytes	0
Metric Computation	54	4928 bytes	0
Metric Intersection	26	8128 bytes	0
Metric Linking	15	2240 bytes	0

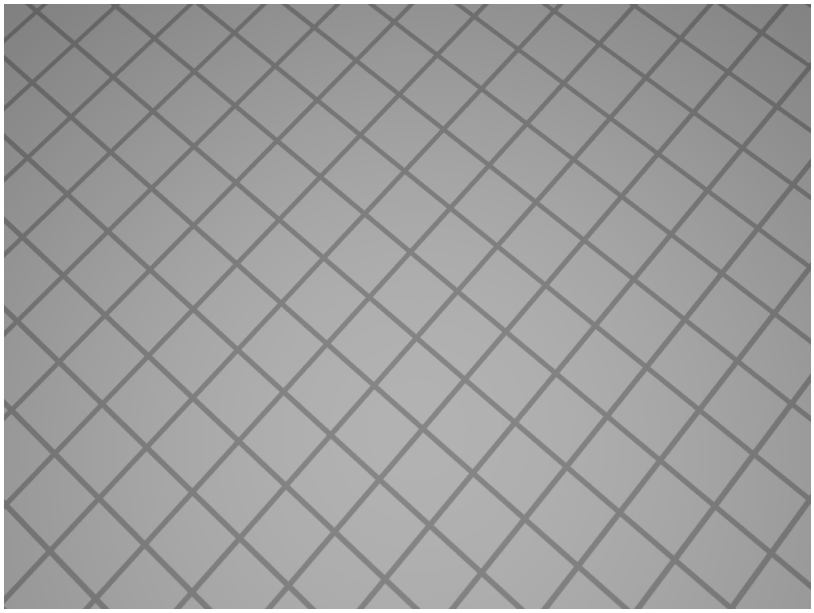
CPU Post-processing

The post-processing stages involve extraction of the GPU-generated ridge data and formation of final ridges in host system RAM. The overall time required for creation of the variable length output ridges in system RAM is dependent upon the number of ridges that must be formed, and for images in this application is faster than the 15 frames per second camera frame rate. The level of filtering performed on the ridge data also affects the CPU-side execution time, as does the search for spatially close ridge endpoints to use in least squares parabola fitting. In the current implementation CPU processing executes faster than the GPU side, so little optimization effort was spent on the CPU code. Current generation processors are much more capable than the Core2 Duo (E8400) used in this work, so CPU processing is not a bottleneck in the system.

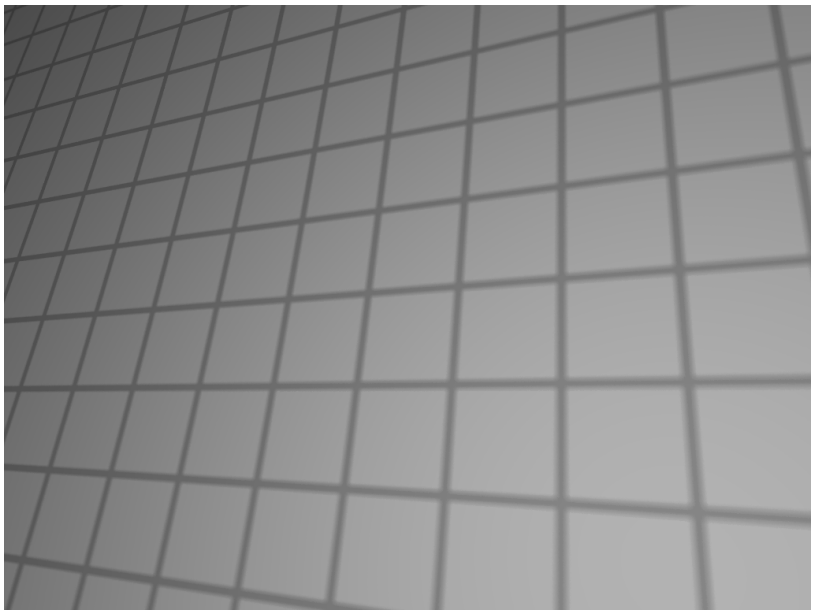
4.4.3 Accuracy Results

To validate the accuracy of the proposed (and implemented) approach to grid line intersection measurement, ground truth data is required for comparison with the extracted coordinates. A synthetic sequence simulating 15 successive camera frames was generated using the Persistence of Vision Raytracer (POV-Ray) [129]. The model, scene description, and camera trajectory were designed to simulate the appearance of real data sequences. Using projective geometry, the model points were projected onto the synthetic camera image planes, and compared with the detected grid line intersections. The scripts written to generate the POV-Ray output are included in Appendix A, and two sample output images are shown in Figure 4.23.

A sequence of 15 synthetic (raytraced) frames are shown in Figure 4.26, with the



(a)



(b)

Figure 4.23: Sample synthetic images.

Table 4.9: Synthetic image grid measurement errors.

Frame Number	Mean error [pixels]		Standard deviation [pixels^2]	
	Horizontal	Vertical	Horizontal	Vertical
1	-0.0010	-0.0053	0.0354	0.0292
2	0.0019	-0.0068	0.0279	0.0252
3	-0.0057	-0.0128	0.0318	0.0255
4	-0.0010	-0.0100	0.0334	0.0284
5	0.0032	-0.0080	0.0320	0.0246
6	-0.0062	-0.0082	0.0684	0.0424
7	0.0009	-0.0042	0.0357	0.0258
8	0.0067	-0.0000	0.0393	0.0248
9	-0.0007	0.0056	0.0488	0.0311
10	-0.0058	0.0062	0.0510	0.0347
11	0.0043	0.0214	0.0620	0.0347
12	-0.0100	0.0246	0.0559	0.0340
13	-0.0005	0.0300	0.0801	0.0474
14	-0.0052	0.0377	0.0952	0.0547
15	0.0066	0.0313	0.1210	0.0785

GPU-accelerated grid intersection locations overlaid. Due to ambiguous scale-space interpolation cube instances, not all grid intersections are detected. To simplify the display, and to demonstrate that missed detections are not persistent, the grid intersections are colour coded. Red markers indicate a newly detected grid intersection point, not seen previously in the sequence. Green markers indicate an intersection detected in the current frame, and also detected previously. Yellow markers indicate the estimated location of a previously seen intersection, which for reasons of noise or estimation uncertainty was not reliably detected in the current frame. To assist reader interpretation of the interframe motion, two coloured tracking circles have been overlaid on the frames. The number n associated with each frame indicates the number of intersections detected in that frame.

Based on analytic intersection locations in each frame, accuracy of the detected intersections was tested. The means and standard deviations of the errors for the 15 frame sequence are listed in Table 4.9, and Figure 4.24 depicts the horizontal and vertical error mean and standard deviations for each frame. Figure 4.25 shows the error distribution from two typical frames, with the position of each point indicating

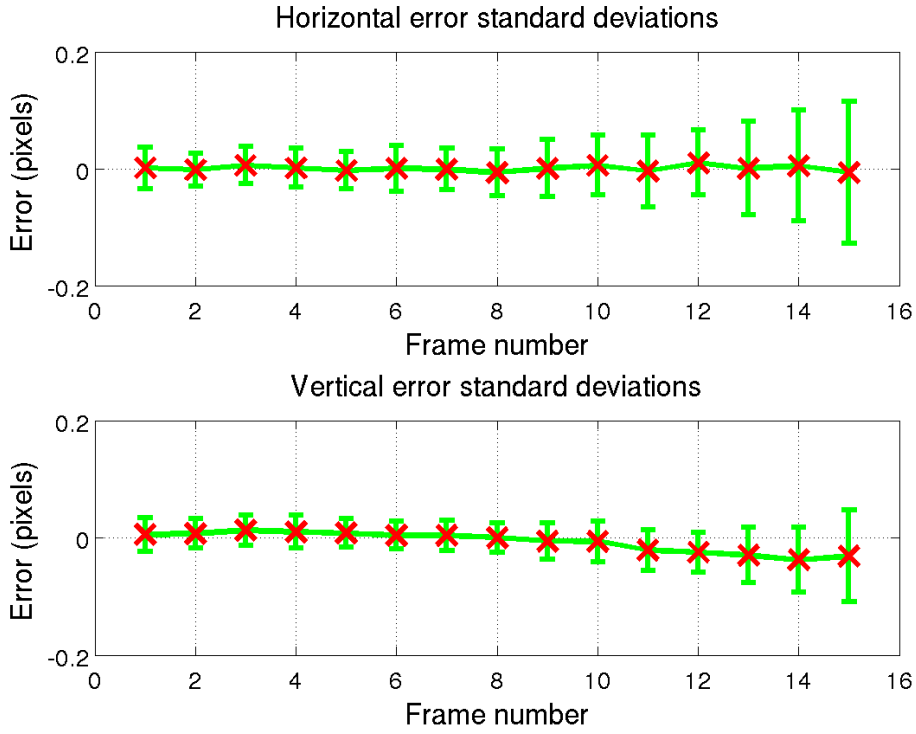
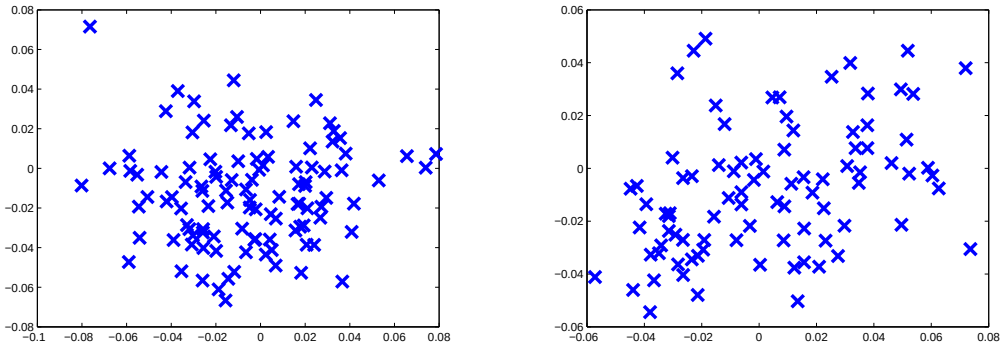


Figure 4.24: Synthetic video sequence - mean intersection detection errors with single standard deviation error bars

the horizontal and vertical error between the analytic and detected intersection points. As seen in Figure 4.24, the error standard deviation increases with the frame count, which corresponds to the increasing depth-of-field blur in the images (as is visible in Figure 4.26). The raytraced sequence intentionally increases camera incidence angle (and thus DOF blur) with increasing frame number to test the measurement accuracy in varying focus conditions.

The worst case error standard deviation was for the most blurred frame (frame 15), as expected, with a horizontal standard deviation of 0.121 pixels^2 and a vertical value of 0.079 pixels^2. Most frames obtained standard deviations below 0.05 pixels^2, with the well focussed frames receiving the least error. Increased DOF blur is expected to increase detection error, primarily because spatial localization information is lost.

Compared with feature detection results in the literature, the scale-space ridge detection approach performs well. For a checkerboard calibration pattern, Douskos



(a) (b)

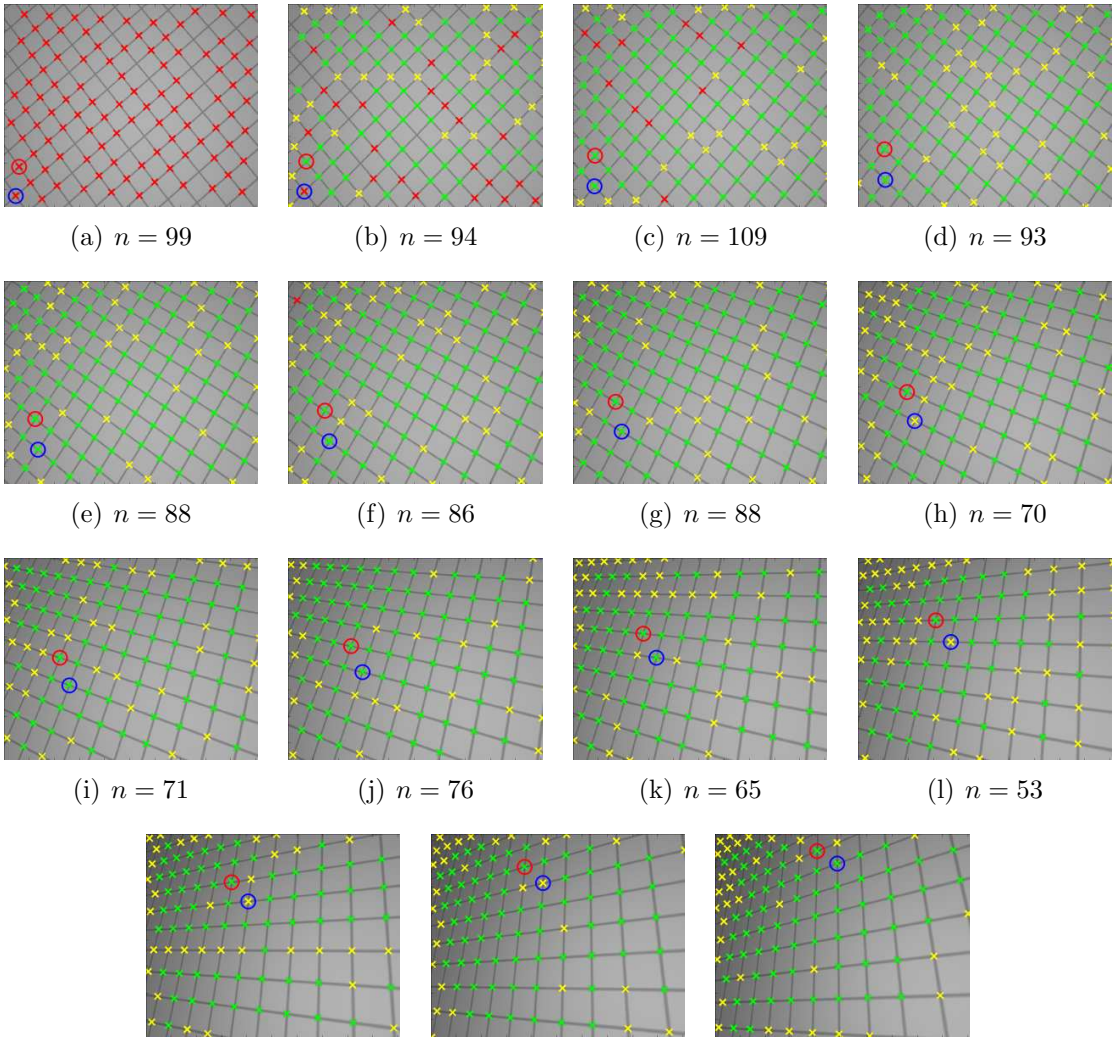
Figure 4.25: Error distribution, in pixels, of detected grid intersections relative to ground truth values (frames 3 and 5, synthetic sequence)

et al. [33] report corner extraction to ≈ 0.1 pixel accuracy. These results are from well focussed images, so the comparable scale-space results in the presence of DOF blur implies improved detection accuracy. Steger et al. [148] reported edge detection error with maximum standard deviation of $\approx \frac{1}{30}$ pixels, and Devernay et al. [30] measured edge position error with standard deviation $< \frac{1}{10}$ pixels. Considering the blurred imaging conditions and interpolated intersection location, the proposed algorithms perform well.

4.4.4 Experimental Video Sequence Results

Experiments using actual video sequences were used to test the robustness of the grid line intersection detection process. Figure 4.27 shows the output from one such experiment, with 15 successive frames shown and with detected grid intersections overlaid (using the same color coding as shown in Figure 4.26).

Noise, surface imperfections, scale-space ambiguities, and parabola fitting errors lead to some intersections being rejected in certain frames. No attempt is made to bridge broken line segments, or to iteratively reduce parabola fitting error through additional filtering or extrapolation. The grid images are inherently part of a video sequence, so as long as temporally neighbouring frames detect an intersection missed in a specific frame, no significant information is lost. Testing has shown that most



(a) $n = 99$ (b) $n = 94$ (c) $n = 109$ (d) $n = 93$

(e) $n = 88$ (f) $n = 86$ (g) $n = 88$ (h) $n = 70$

(i) $n = 71$ (j) $n = 76$ (k) $n = 65$ (l) $n = 53$

(m) $n = 66$ (n) $n = 75$ (o) $n = 71$

Figure 4.26: Synthetic video sequence frames with detected intersections overlaid. Red markers indicate intersections newly detected in the current frame, green markers indicate an intersection detected in the current frame and also in previous frames. Yellow markers provide the estimated locations of intersections detected in a previous frame, but not detected in the current frame. Colored circles have been logically fixed to two grid intersections, to provide a visual means to track the camera motion. The number n associated with each frame indicates the number of intersections detected.

missed intersections are not persistent, as can be seen for example in Figure 4.27. Where a missed intersection is in fact persistent across frames, it is because of physical damage to a grid such as a scratch or reflection from dirt that obscures a portion of the line.

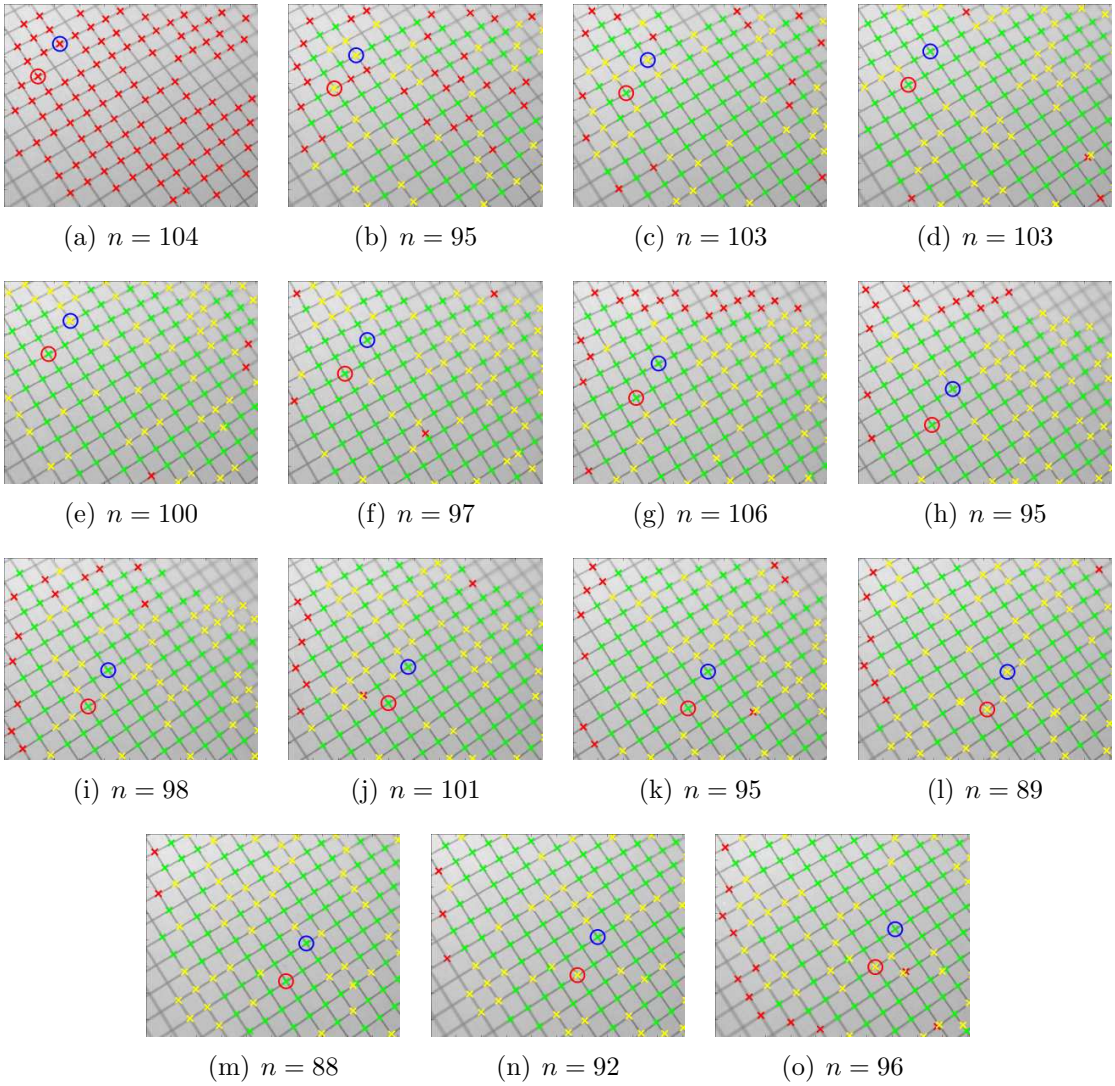
4.4.5 Discussion

Fidelity of poorly focussed intersections

Given the presence of depth-of-field blur around grid intersections, the exact response of the grid intersection locations is perturbed by the effect of lens distortions and other non-linear effects. At larger scale-levels, the increased filter variance reduces trust in the spatial location of the feature of interest. For these reasons, and without extensive calibration and modelling of a specific vision system in the presence of significant blur, grid intersections that are detected at large scale levels are taken to be untrusted. These intersections are used to track the motion of the grid between frames of a video sequence, but are not considered to be accurate data points as input to a three-dimensional reconstruction problem. When all intersections are simultaneously blurred, motion blur caused by an operator sharply jerking the camera cannot be ruled out. The intersections in these occasional highly blurred images should be used for tracking only, and can be identified by large detection scale, and also through increased error in the parabola fitting process. Some analysis of scale-space response in the presence of blur is provided by [179].

Extension to non-separable kernels

The GPU algorithm, as implemented for this work, requires a separable scale-space generating kernel. Extension to non-separable generating kernels would require a new convolution engine, replacing the GPU horizontal and vertical convolution kernels with a single convolution stage. It is expected that some decrease in performance would result, primarily related to memory access complications across two-dimensional regions of global memory. Given the relatively small size of multiprocessor shared memory, multiple convolution passes across a single portion of raw data (including sharing of partial results between thread blocks) would likely be required



(a) $n = 104$ (b) $n = 95$ (c) $n = 103$ (d) $n = 103$

(e) $n = 100$ (f) $n = 97$ (g) $n = 106$ (h) $n = 95$

(i) $n = 98$ (j) $n = 101$ (k) $n = 95$ (l) $n = 89$

(m) $n = 88$ (n) $n = 92$ (o) $n = 96$

Figure 4.27: Actual test video sequence. Red markers indicate intersections newly detected in the current frame, green markers indicate an intersection detected in the current frame and also in previous frames. Yellow markers provide the estimated locations of intersections detected in a previous frame, but not detected in the current frame. Colored circles have been logically fixed to two grid intersections, to provide a visual means to track the camera motion. The number n associated with each frame indicates the number of intersections detected.

for practical filter kernel sizes.

4.5 Summary of this Chapter

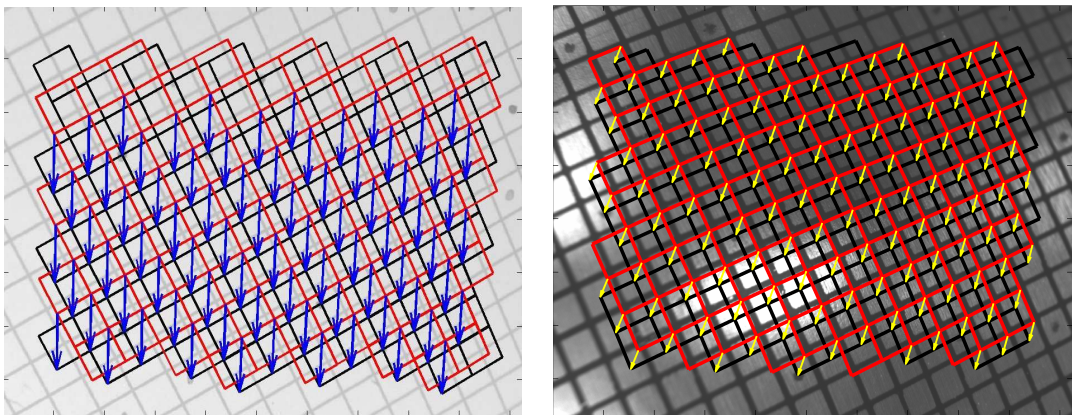
This chapter has provided a review of GPUs in the context of general purpose computational acceleration, and has described parallel design of the scale-space ridge extraction algorithm for GPGPU implementation. The algorithm has been demonstrated at faster than 15 frames per second, which is the associated camera frame rate. The system is capable of processing volumes of data in sustained video sequences, typically extracting more than 1500 sub-pixel grid intersections per second. Synthetic experiments have been performed to validate the sub-pixel detection accuracy of the system, and real video sequences have been used to verify system performance in the presence of noise and other real world conditions.

Chapter 5

Topological Interframe Grid Tracking

Multiple view triangulation using a single (monocular) camera requires movement of either the camera or target to obtain views from varied poses. Correlation of features between images is also required, so that rays corresponding to the same feature can be intersected. Correlating grid line features across a video sequence can be solved through interframe motion tracking, allowing a single grid intersection to be identified in all frames for which it is visible. Even when outside of the camera field of view for many frames, the regular grid structure allows intersections to be reacquired as they enter back into view. Tracking must be error free across an entire video sequence to enable calibration and robust triangulation, even when some frames are strongly blurred. Failure to extract the correct interframe transform splits a video sequence, preventing feature correlation across the failed frame (temporally), and therefore preventing triangulation using viewpoints both before and after the failure.

Sample interframe motion vectors from video sequences of a flat calibration plane and deformed metal dome are shown in Figure 5.1. The regular pattern and well defined grid line intersections provide a robust feature set, avoiding the more general problem of locating and correlating natural features, such as interest points, between images. Interframe motion tracking has been extensively studied in the literature, in some cases for motion compensation in video compression algorithms. GPGPU applied to interframe motion tracking is described by Park et al. [121].



(a) (b)

Figure 5.1: Sample interframe motion vectors from (a) planar calibration target, (b) deformed metal dome.

Although grid tracking and point correlation are required by many camera calibration algorithms, they are not addressed in the literature. Most camera calibration targets are chosen to be smaller than the camera field of view, allowing boundaries to be detected and used for absolute grid positioning. Tracking of the close-range grid images poses three challenges not typically encountered by motion tracking applications.

1. **Grid encompasses entire field of view:** The camera field of view is filled by the surface grid, so no external marks or shape information about the grid outline can be discerned. A large grid (relative to camera field of view) is chosen to better pose the hand-eye calibration problem, as described in Chapter 6.

2. **Grid pattern is regular, with multiple shifts appearing similar:** The grid pattern is repeating, so aside from occasional surface defects such as scratches or dirt, units of the grid are indistinguishable from one another. Coupled with the occasionally erratic motion generated by human manipulation of the camera, different shifts of the grid pattern can produce the same apparent result. Region matching approaches such as Sum of Absolute/Squared Differences (SAD/SSD) therefore do not provide unique solutions.

3. **DOF blur changes appearance of intersections between frames:** With

changing DOF blur, correlating intersections and therefore interframe motion using scale or width of the intersections fails. In frames where the entire field is blurred, interframe tracking must still be obtained to avoid splitting the video sequence.

Given occasional dirt or scratches on the sheet metal surface, an obvious approach is to track these surface imperfections. With realistic (non-diffuse) lighting, however, the apparent size, shape, and existence of these types of features are transient and unstable. Even with diffuse lighting, DOF blur can quickly obscure such surface features.

To generate irregularity in the grid pattern, fiducials are occasionally placed within grid squares. For this work, the fiducials are single spots from a fine tipped felt marker, but dots or other shapes can be included in the grid production process for larger scale implementation. Simplification of interframe tracking could be achieved through addition of uniquely coded information to the fiducials, such as a unique identification number or graphic code. [40] considers this approach for checkerboard grid patterns. Uniquely coded fiducial markers are not used in this work, however, because they significantly complicate the grid production process and may be unreliable in the presence of blur.

Irregularity could alternatively be added to the grid pattern through irregular line spacing or line width, simplifying the correlation of intersections between frames, and providing a unique interframe registration solution. To maintain simplicity of grid printing and to provide a more robust solution, this work restricts itself to the more difficult problem of regular (uniform) grids.

The remainder of this chapter describes a novel interframe tracking process that exploits topology of the grid structure. Dimensionality of the projective tracking problem is reduced, leading to a parameter space that can be searched exhaustively. Grid asymmetry such as fiducial markers are not explicitly extracted or measured, but are used passively by the algorithm, greatly relaxing requirements on fiducial production method and characteristics.

5.1 Interframe Tracking Using Fiducials

Fiducials introduce non-uniformity into the grid pattern. To maintain this function, the fiducial pattern should be asymmetric within the image field of view, and should produce a unique registration solution for practical grid motions. Fiducials can enable tracking in two ways, either explicit or implicit.

5.1.1 Explicit Fiducial Tracking

Explicit tracking involves fiducial feature extraction such as blob measurement, followed by motion estimation from the feature points across frames. Scale-space blob detection [94] has been demonstrated to extract the blob fiducials in the presence of DOF blur (Kinsner et al. [74]). An initial estimate of the interframe motion and therefore correlation between fiducial points can be produced, and then refined through an approach such as Random Sample Concensus (RANSAC) [60]. Initial (putative) matching can be performed using historical motion vectors, region correlation techniques such as SAD/SSD, properties of the fiducial mark, and other approaches. In practice, motion vectors from previous frames have been found in test sequences to be unreliable as predictors of current frame motion. This is primarily because human manipulation of the camera routinely reverses the direction of motion between consecutive video frames, and the projective nature of the camera causes small angular accelerations to produce large changes in apparent interframe trajectory.

Solution to the general tracking problem between images of a target involves computation of the projective transform between the two images. The resultant 2-D projective transform (homography) has eight degrees of freedom, and can be computed from a minimum of four point correspondences [60]. The challenge in this application is initial matching of the fiducial features, especially in the presence of varying DOF blur. Even with RANSAC approaches, the degrees of freedom in the transform coupled with changing image conditions impede robust tracking.

5.1.2 Implicit Fiducial Tracking

A second approach to interframe tracking assisted by fiducials is an intrinsic approach, where the marks are not explicitly extracted, but the properties of which are included in some higher level description of the image. This approach carries the advantage that a feature detector need not be selected to robustly locate fiducials in the presence of varying depth of field blur, and there is much more flexibility in fiducial appearance and production method.

The algorithm proposed here combines grid topology with fiducial structure. Fiducials are defined only approximately in terms of shape, size, and placement, and can even be hand drawn marks placed randomly within the grid. Existing work to automate detection of checkerboard calibration patterns is available from [143] and [144], but they consider pose estimation from single checkerboard frames, and do not facilitate interframe motion tracking. Checkerboard patters differ in structure compared with a line-based grid, and thus require a different topology construction approach.

5.2 Proposed New Interframe Tracking Method

By building a topological representation of the grid structure, the dimensionality of the projective tracking problem is reduced from a general homography (eight continuous degrees of freedom (d.o.f.)) to discrete shifts in the directions of the grid axes (two discrete d.o.f.). Tracking fiducials are intrinsically measured during the topology construction process, obtained through active pixel ratios within thresholded connected components. The processing steps are shown in Figure 5.2.

5.2.1 Thresholding

To avoid the need for explicit fiducial and line detection, an image is thresholded (binarized) to isolate the image regions between grid lines. The goal of thresholding in this context is to separate grid lines and fiducials from the remainder of the image. The following algorithm stages are resilient to noise and artifacts, and because grid line intersection measurement (which must be accurate) does not utilize the binary output, the process does not have to be precise. In many images, varying illumination

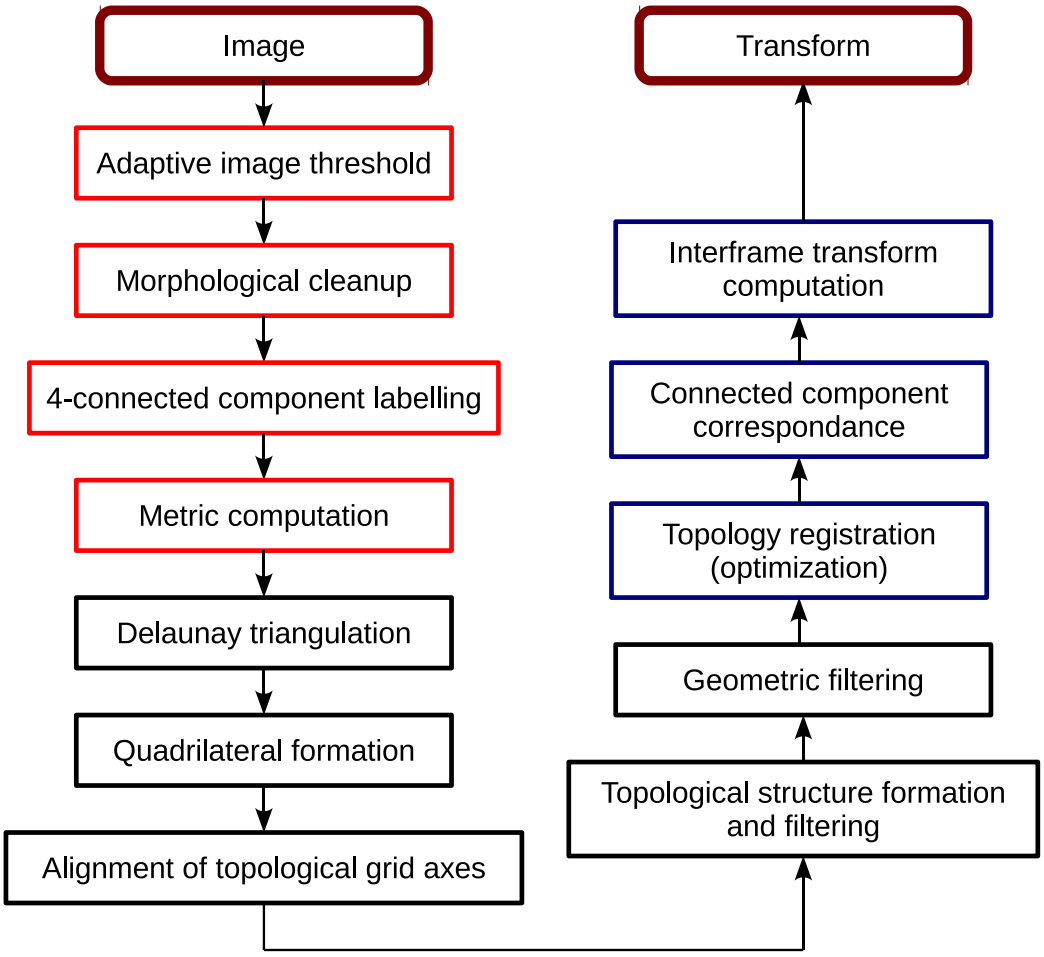


Figure 5.2: Topological tracking - algorithm flow. Pre-processing steps are coloured red (first set of boxes), structure extraction and filtering are in black, and interframe transform calculation in blue.

and low contrast preclude the use of a global thresholding algorithms such as the Otsu method [117]. Instead, local adaptive thresholding is required to compensate for spatially changing illumination and reflections. The thresholding algorithm should notably reduce or remove the effect of specular reflections from the part surface, decoupling the grid registration problem from the changing position of reflections on curved surfaces.

For images of both planar calibration grids and deformed metal surfaces, the Shafait et al. thresholding method is chosen [141]. This method is based upon the

Sauvola et al. [136] technique, but is accelerated using integral images [169] to enable rapid computation of local image statistics. Surveys and analyses of binary thresholding techniques in the context of document binarization are considered in [6] and [140], both of which select the Sauvola appoach as producing the best results. Figure 5.3 shows sample thresholded image output from the Shafait method. For the grids tested, a Shafait radius parameter fixed at 30 produced good results.

An alternative thresholding method that has been found to be effective on deformed sheet metal surfaces is the optimization approach of Ray et al. [130]. The technique has no parameters to adjust, but requires at least hundreds of iterations to converge (for the images tested). This algorithm is useful for severely distorted and deformed sheet metal images, in which the stamping process has damaged portions of the grid. For practical calibration and deformed part surfaces, the Sauvola algorithm performs well and is significantly faster than the iterative optimization approach.

5.2.2 Morphological Cleanup

Image thresholding often produces speckle noise and other small scale erroneous features. To remove these features, a morphological close operator [25] is applied. The result from a sample image region is shown in the left column of Figure 5.4, with some speckle noise removed from the grid unit interior.

In a later processing step, the tracking algorithm uses connected component labelling to identify the interiors of grid units, which are separated by the thresholded grid lines. Any small gaps in the lines cause grid elements to be erroneously joined together. A simple morphological erosion [25] phase is used to close any small gaps in the thresholded grid lines, isolating grid unit interiors that were weakly connected before erosion. The right column of Figure 5.4 shows the effect of the erosion operation on a motion blurred image region.

The structuring element used for morphological closing and erosion is a cross operator of radius 1 pixel, as shown in Equation 5.1.

$$s_e = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{5.1}$$

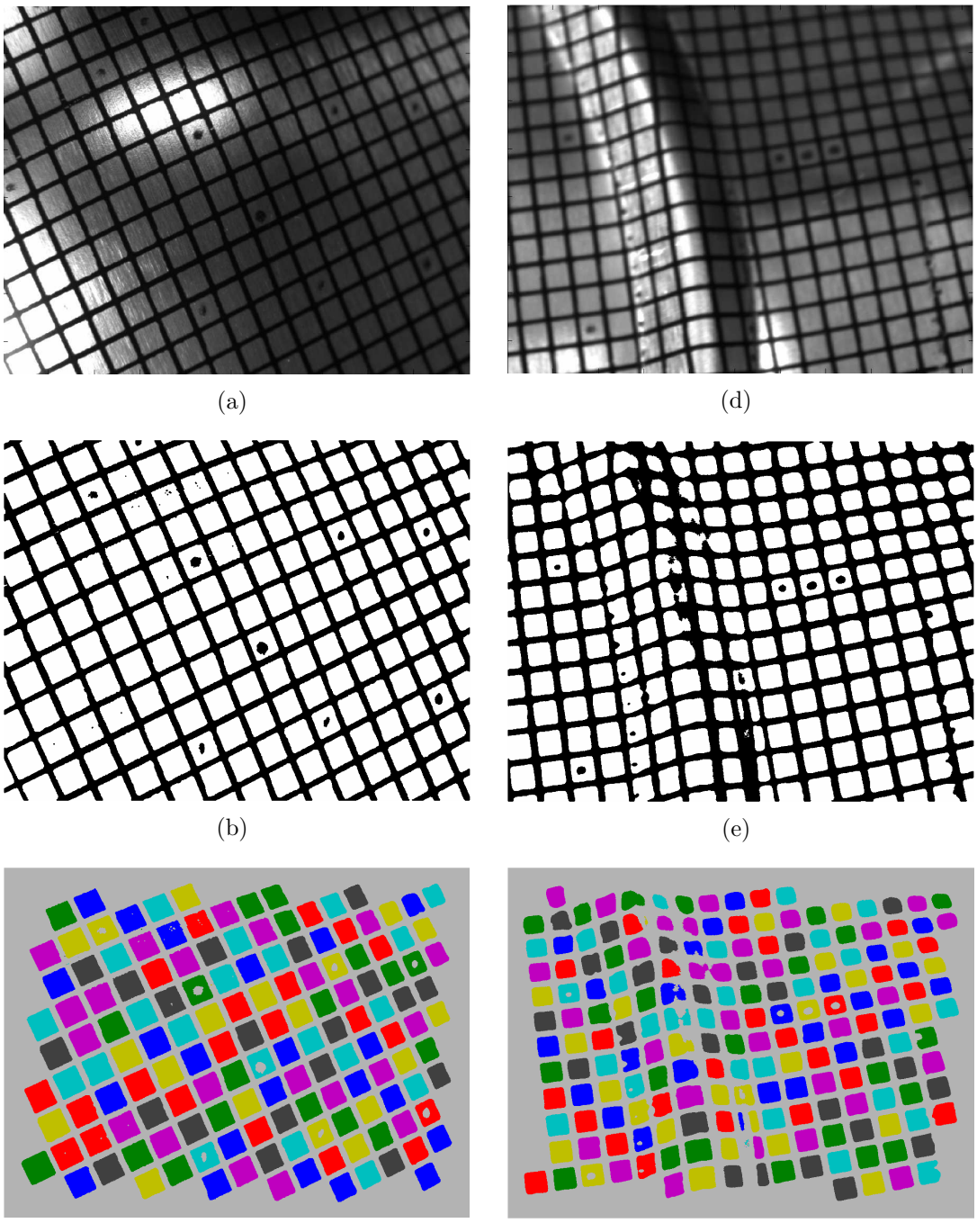


Figure 5.3: Two sample images using the Shafait thresholding algorithm. The original image is shown in (a),(d), followed by the thresholded image in (b),(e), and a connected component colouring of the thresholded data in (c),(f).

5.2.3 Connected Component Labelling

Connected component labelling is used to uniquely identify the isolated image regions between grid lines. The topological relation between these regions will form the basis of the tracking approach. Thresholded grid lines separate the interior grid regions from one another, so connected component identification associates pixels with the unique grid interior region that they belong to. Four-connected component labelling is used because regions are well defined, and high frequency noise has been suppressed through morphological processing. Output from the connected component labelling process is shown in Figures 5.3(c) and 5.3(f).

Connected component analysis can be performed with linear complexity in the number of pixels [157].

5.2.4 Metric Computation

Interframe tracking estimates motion by comparing the similarity of grid elements across various possible shifts of the grid. To compare the grid elements, a metric value is computed at each connected component region, and this number is used by the optimization objective function in a later step. Key requirements of the metric are that fiducial information must be incorporated, and that the value should be relatively invariant to changes of the connected component size as camera range changes.

After thresholding, both grid lines and fiducials have a common binary value, while the remaining pixels which form the connected components should have the complimentary value. When fiducials occur within a grid unit, they appear as a hole or protrusion into the connected component region. The number of pixels that are active in a connected component (n_{thresh}) can be compared with the number of pixels in a filled version (n_{total}) of the component. For grid elements with no fiducial, there should be no (or few) pixels on the interior of the region that are not set. When a fiducial if present, however, its area is effectively subtracted from the connected component, reducing the pixel count compared with a filled version of the same component. The metric value defined for the i-th connected component in frame k is therefore:

$$\sigma_i^k = \frac{n_{thresh}^i}{n_{total}^i} \tag{5.2}$$

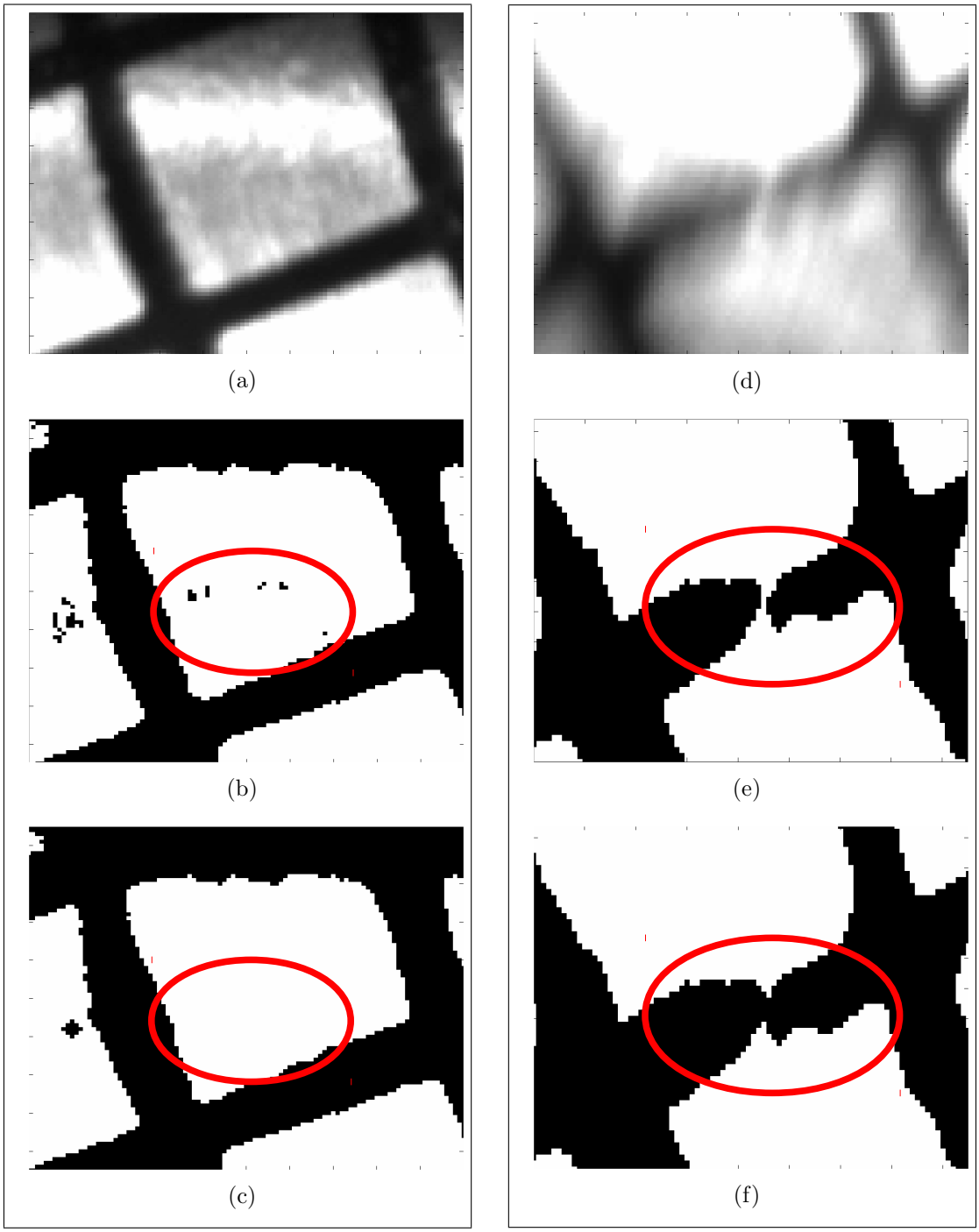
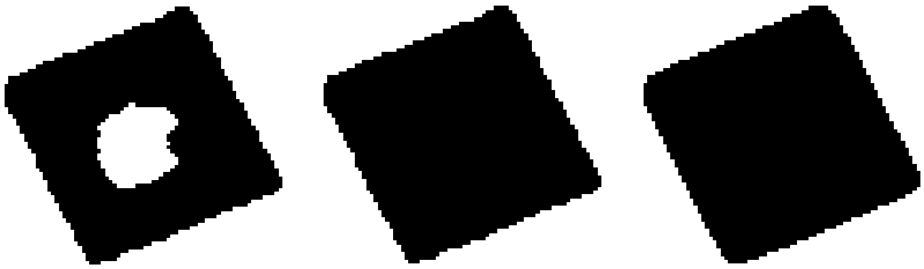


Figure 5.4: Sample results from morphological processing. Two images regions are shown, the first with some thresholded noise, and the second with a broken grid line caused by motion blur. For both images the original is shown first ((a),(d)), followed by the raw thresholded version ((b),(e)) and the morphologically processed data ((c),(f)).



(a) Original, n=2572 (b) Filled, n=2922 (c) Convex hull, n=3031

Figure 5.5: Sample connected component with fiducial. (a) Thresholded region, (b) region after filling internal holes, (c) convex hull of (a). n indicates number of active pixels in the region.

Counting of the active pixels n_{thresh} in a region can be incorporated into the connected component labelling process. Region filling (to compute n_{filled}) can be accomplished through a number of algorithms. The first approach of simply filling interior holes in the connected component can be achieved through morphological operators, inverse pixel connected component labelling, or a variety of other techniques. An alternative approach is to compute the convex hull of the connected component, algorithms for which are well studied and available in textbooks such as [116]. Figure 5.5 shows a typical connected component with an internal fiducial, with nearly identical results from filling and convex hull construction. Convex hull analysis provides an advantage in situations where a fiducial or surface mark is not completely enclosed by the connected component (e.g. touching the grid line), an example of which is shown in Figure 5.6. Internal hole filling does not detect the fiducial, while convex hull analysis fills the protrusion into the component, thereby reflecting the feature in the metric value. The disadvantage of convex hull analysis is that few connected components have fiducials that are not completely enclosed, but many regions have slightly concave boundaries. Convex hull analysis increases the n_{filled} value for most components, thereby reducing the mean metric value.

The relatively small number of connected components that must be visited, typically on the order of 100-200 per image, leads to acceptable CPU computational load. Numerous convex hull algorithms are available, many with computational complexities of $\mathcal{O}(n \log n)$ for n input points [116].

(a) Original, n=1504 (b) Filled, n=1504 (c) Convex hull, n=1730

Figure 5.6: Sample connected component with defect from manufacturing press. (a) Thresholded region, (b) region after filling internal holes, (c) convex hull of (a). n indicates number of active pixels in the region

For grid units with no fiducial, the ratio defined in Equation 5.2 is near unity. When a fiducial or other distinguishing mark exists within the interior of a connected component, the ratio is less than one. A histogram of the metric values for 14568 connected components, across 100 frames of a deformed metal dome video sequence, is shown in Figure 5.7. Filled components (as opposed to convex hulls) are used in the test. The vertical axis is shown with a log-scale because the histogram bin near unity holds the majority of values (93.44% of the metric values are larger than 0.98).

At this stage of processing, simple filtering is performed. Any connected components with less than a threshold number of pixels are rejected to avoid considering small noise regions in the grid topology. The centroids of all remaining connected component regions are then calculated, as shown in Figure 5.8, forming the vertices for subsequent triangulation.

5.2.5 Delaunay Triangulation

The topology of the connected component regions must be extracted and recorded. To accomplish this, Delaunay triangulation is applied [26] to centroids of the regions. Delaunay triangulation is a well known and studied algorithm, and provides a well balanced triangulation by maximizing the minimum angle in a triangle (avoiding "skinny" triangles). Specifically, the triangulation is formed from a set of vertices

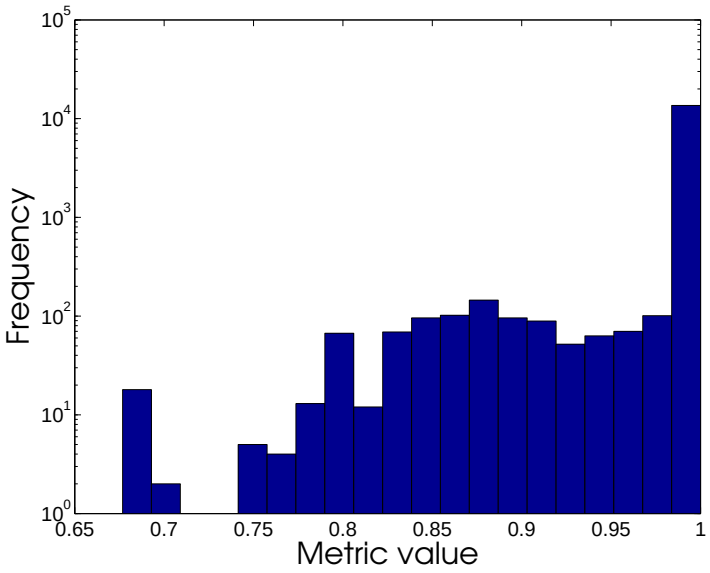


Figure 5.7: Histogram of metric values from 100 image frames of a deformed metal dome, including 14568 connected component regions. The vertical axis is log-scale because the histogram bin near unity holds the majority of values.

such that no fourth point is inside the circumcircle of the three points joined in a triangle. Example output is shown in Figure 5.9(a).

An advantage of this approach is that the nearest neighbour graph is a subgraph of the Delaunay triangulation, allowing neighbouring regions in the regular grid topology to be rapidly located. For camera imaging angles close to the deformed surface normal (camera plane parallel to part surface), this neighbour search method is effective. The use of Delaunay triangulation in the presence of projective distortion, as caused by shallow camera view angles, is considered at the end of Section 5.2.6.

Delaunay triangulations can be performed in $\mathcal{O}(n \log n)$ time for n input points [26].

5.2.6 Quadrilateral Formation

Delaunay triangulation of the connected components produces, on average, six edges connected to each vertex. Only four of the edges correspond to principal grid directions, so the remaining edges must be pruned. There are multiple methods to

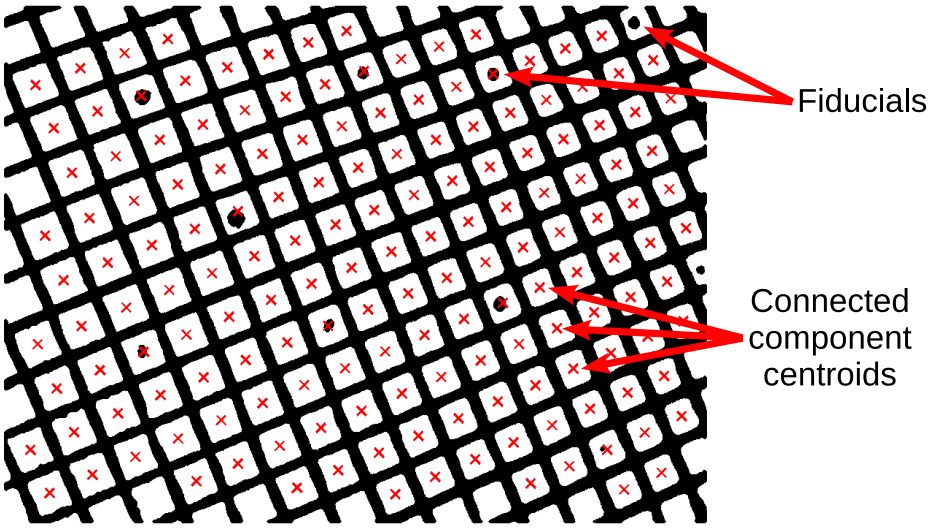


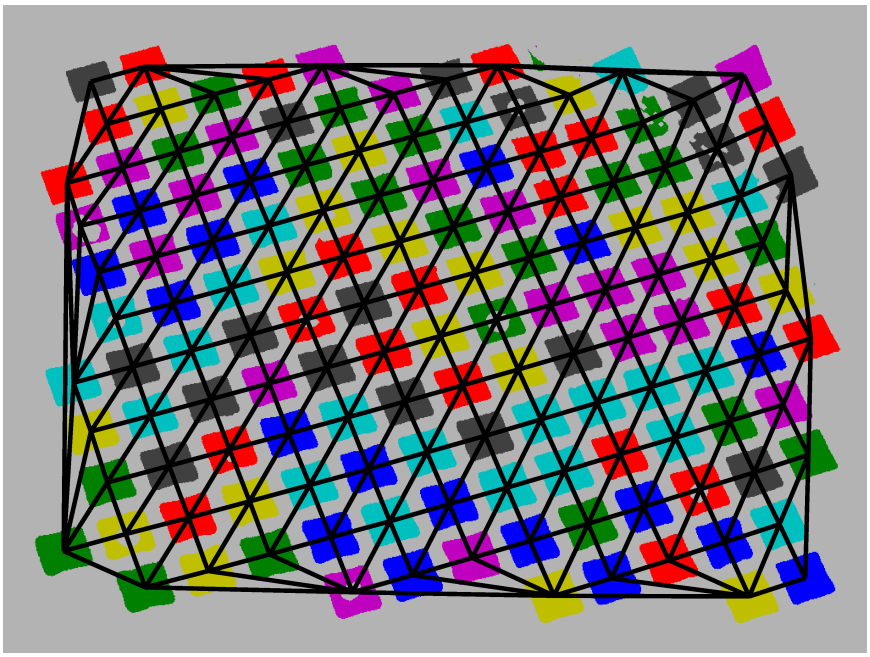
Figure 5.8: Thresholded grid image frame with fiducials visible. Centroids of connected components are denoted by 'x' marks.

accomplish this, with the simplest using the relative length of edges connecting a group of local vertices. The grid pattern printed on the metal surface is rectilinear, so in orthogonal views of each triangle the hypotenuse corresponds to the edge that should be trimmed. From this observation, complementary triangles that share a common "long edge" can be paired to form a quadrilateral. The common long edge is removed from the representation, leaving edges that link connected components in the principal grid directions. Figure 5.9(b) shows an example of this pruning process. The result of pruning leaves a collection of quadrilateral structures in memory, with each entry including the point indices of the four associated vertices.

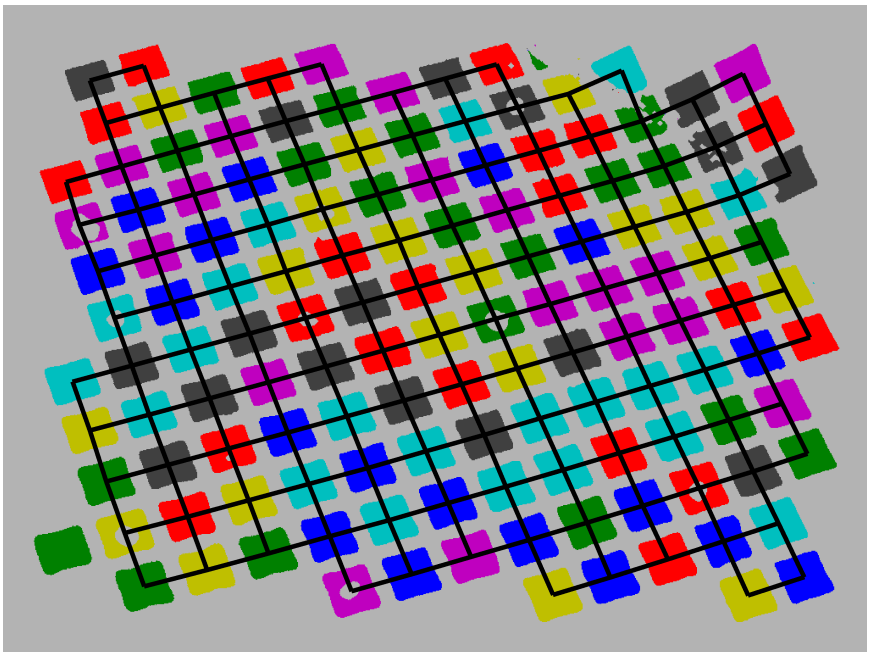
To simplify the matching of long edges, the vertices of each triangle are permuted such that indices of the longest edge are in a common position in the triangle record.

Long Edge Analysis

The approach of pruning the long edge of each triangle assumes that the grid pattern is rectilinear (orthogonal grid lines), and also that there is little distortion in the image. The camera forms projective images of the target, so an analysis of the "long edge" approach to pruning is required. A simulation is set up as shown in Figure 5.10, in which a square is imaged using projective geometry and a simulated pinhole camera.



(a) Delaunay triangulation.



(b) Pruned quadrilaterals.

Figure 5.9: Sample of long edge pruning from the Delaunay triangulation. The results is the set of quadrilaterals connecting grid units in the principal grid directions.

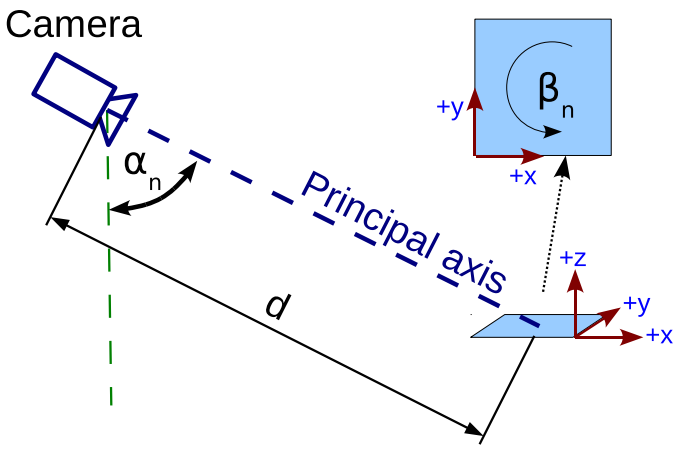


Figure 5.10: Simulation parameters for long edge assumption in grid unit matching.

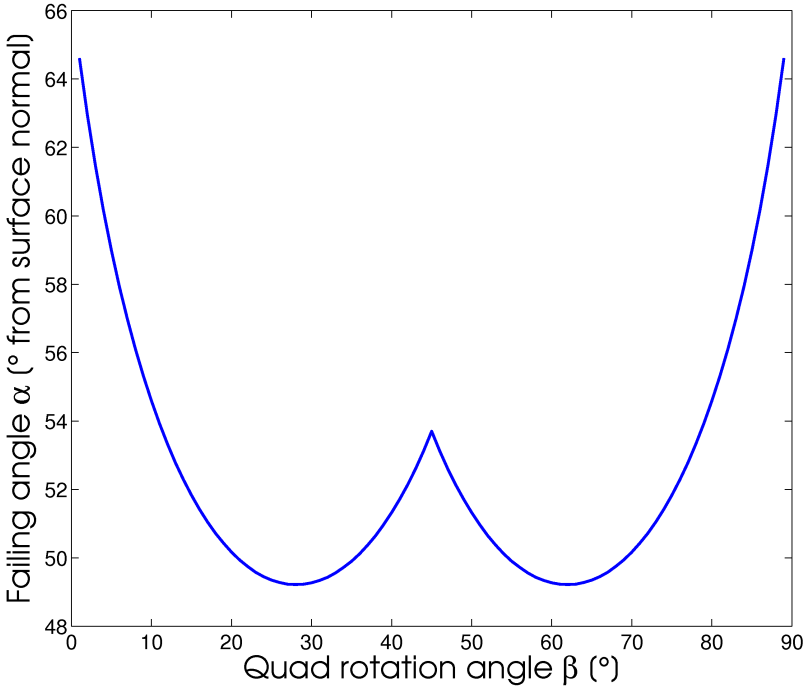
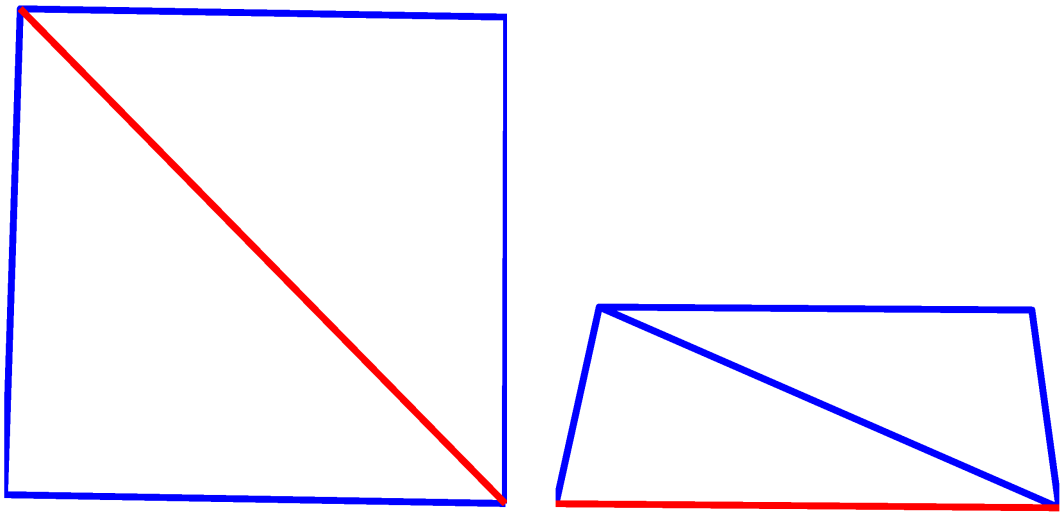


Figure 5.11: Critical camera viewing angle beyond which grid unit Delaunay diagonal is shorter than another edge.

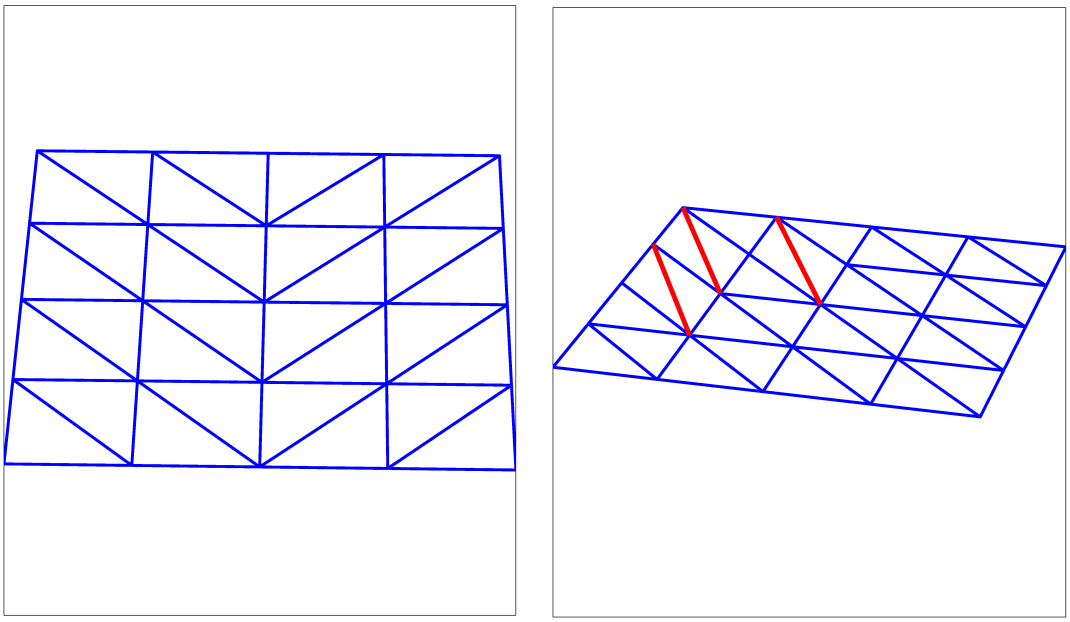


(a) Long edge test passed ($\alpha_n = 10°$, $\beta_n = 1°$). (b) Long edge test failed ($\alpha_n = 72°$, $\beta_n = 1°$).

Figure 5.12: Sample triangulations from the long edge view angle simulation. The light (red) line is the longest edge, and is ideally the hypotenuse of the triangulation. When view angle causes a non-hypotenuse edge to become the longest, edge length can no longer be used to locate the common triangle edge forming a quadrilateral. Perspective distortion is visible in the second image.

Delaunay triangulation is performed on the imaged vertices, and the diagonal edge length compared to the square boundary edges. When the diagonal edge (common between two triangles) appears shorter than another edge, the long edge approach to diagonal identification has failed. Rotation and camera angle parameters are defined as shown in Figure 5.10, and for a set of rotation angles β_n, the camera angle α_n that causes failure is recorded. The camera distance d is held constant, and the results for 90° of α_n rotation are shown in Figure 5.11. An example of a projection that passes the long edge test is shown in Figure 5.12(a), with the long edge coloured in red. A corresponding failure is shown in Figure 5.12(b).

The analysis shows that the long edge approach to triangle pruning is appropriate for the close-range imaging application. The worst case failure angle occurs at approximately 49° from the surface normal, which is beyond practical view angles in the presence of DOF blur.



(a) Delaunay test passed ($\alpha_n = 50°$, $\beta_n = 1°$). (b) Delaunuay test failed ($\alpha_n = 66°$, $\beta_n = 15°$).

Figure 5.13: Sample triangulations from the Delaunay angle simulation. The blue lines indicate valid Delaunay edges, as defined by the true geometry of the grid points before imaging. In (b), the red lines indicate failure edges which do not conform to the correct geometry. The failures occur because projective imaging causes points to appear near each other in the image plane.

Delaunay Triangulation Under Perspective Imaging

Delaunay triangulation is used to join neighbouring grid units in the topology, producing a nearest neighbour graph with some additional diagonal edges. The camera forms projective images of the gridded target, so analysis of the Delaunay triangulation in the presence of projective imaging is required. A simulation is set up in a similar manner to the "long edge analysis", with the same parameters as shown in Figure 5.10. A small grid structure with 25 vertices is imaged using projective geometry and a simulated pinhole camera. Delaunay triangulation is then performed on the imaged vertices, and the resultant graph edges compared with legal edges in the grid structure. As the camera viewing angle from the surface normal increases,

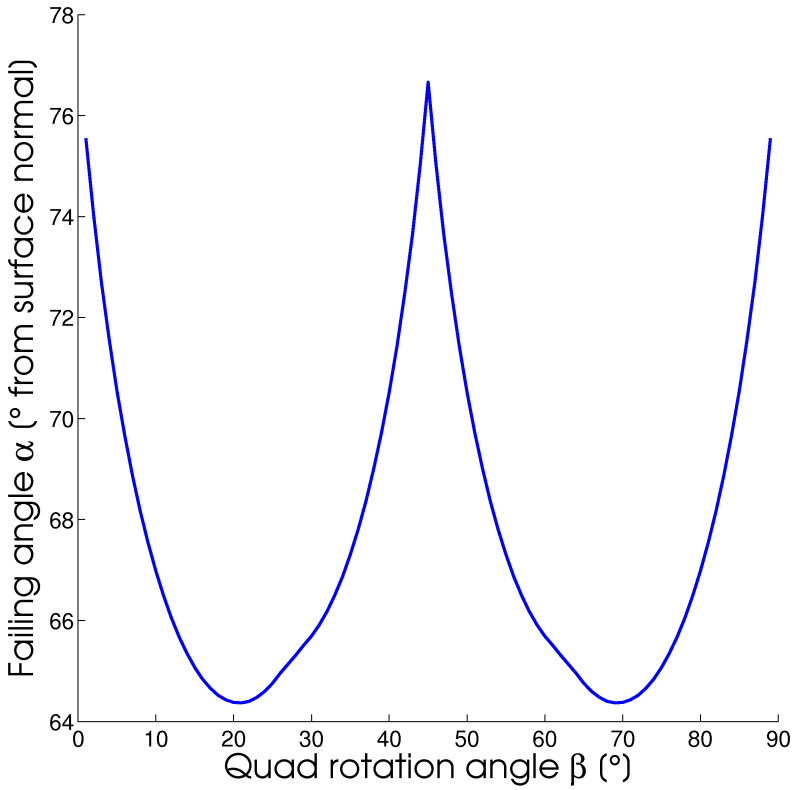


Figure 5.14: Critical camera viewing angle beyond which Delaunay edges cease to conform to the true grid geometry.

perspective distortion eventually causes the triangulation to connect vertices that appear as nearest neighbours in the image, but that are in fact not neighbours in the true grid geometry. The viewing angle (α_n) and grid rotation angle (β_n) at which the Delaunay triangulation generates an invalid edge, relative to the true grid geometry, is shown in Figure 5.14. The worst case failure angle occurs at approximately 64° from the surface normal, which is beyond practical view angles in the presence of DOF blur. Delanuay triangulation is therefore a suitable method for nearest neighbour identification during close-range imaging of a rectilinear grid.

5.2.7 Grid Axis Alignment

To reduce the degrees of freedom in the interframe optimization and to simplify the topology construction process, principal grid directions of the quadrilateral (quad) representation are aligned between frames. In the first frame of a video sequence, a quad near the centre of the image is chosen, and two of its orthogonal edges are selected to define the principal directions of the grid structure. All quads identified in the previous pruning stage are then permuted such that the vertices follow a common order, defined by the principal grid directions.

In all following image frames, a quad near the image centre is chosen, and its edges that best match the previous principal grid directions are selected as the defining vectors in the current frame. All quads are then permuted to have a common vertex ordering, based upon the principal vectors. Sample output is shown in Figure 5.15, with each colour indicating a specific vertex identification in the common ordering. The orientation of all quads is the same after permutation, simplifying subsequent processing stages.

5.2.8 Topological Structure Formation and Filtering

From the collection of vertices forming quad structures, neighbours can be joined to record the topological structure of the grid. Vertices are chosen to be nodes in the topological representation (as opposed to quads for example), because each vertex has a metric value associated with it. This approach allows the later optimization to overlay grids and directly compare metric values at nodes of the representation.

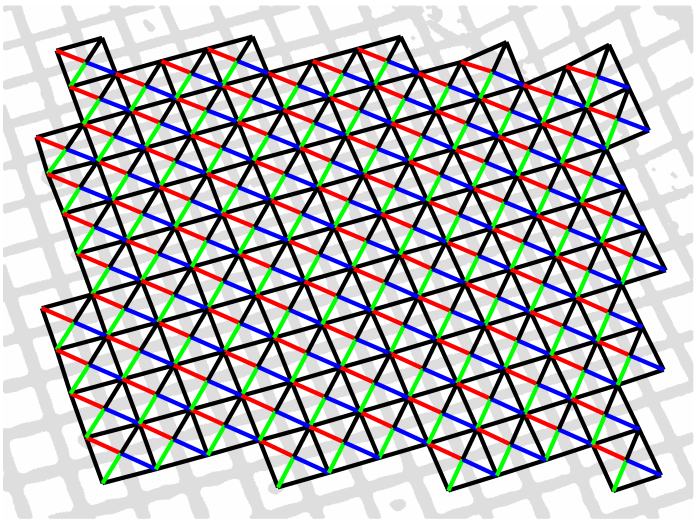


Figure 5.15: Vertex ordering relative to principal grid directions. Each colour indicates the position of a vertex in the common ordering, with each quad oriented in the same way.

A vertex near the image centre is used as a seed, and a breath-first traversal of the topology is performed using a FIFO queue. Many representations of vertex neighbour relationships are possible, but for the rectilinear grid in this work, a simple two dimensional array containing vertex IDs is sufficient. The two dimensional nature allows neighbour relationships to be quickly constructed from the grid, and even if occasional vertices are missed or rejected, the topology beyond the missed element may be filled in from alternate directions. Construction of the topology is summarized in Algorithm 5.1.

The consistent ordering of vertices in the quad structures (with respect to the principal grid directions) enables rapid topological filtering. Of the quads containing a specific vertex, that vertex ID should appear at a unique position in each quad. Any repeat of vertex location indicates that quads are superimposed on one another, which is not physically possible for the grid chosen. These ambiguous cases indicate a local failure of thresholding or damage to the physical grid surface, so no neighbouring vertices are queued. The damaged portion of the topology will be filled in, as much as possible without ambiguity, by processing of vertices that approach the region from different directions.

Algorithm 5.1 Grid Topology Construction

```
Queue vertex that is near image centre
while Queue not empty do
    Retrieve vertex from head of queue
    if Vertex ID not already visited/processed then
        Record vertex ID and metric value in the topology matrix
        Scan the quad structures and find all edge-connected neighbouring vertices
        if More than 4 quads contain vertex, or non-unique positions in quads then
            Finish loop iteration
        else
            Queue all neighbouring vertices, and record addresses where they should
exist in the topology matrix (relative to the current position)
        end if
    end if
end while
return Completed topology matrix, containing metric values and IDs of associated
vertices
```

A second inherent filter stage is also performed during topology construction. When a vertex is being processed, all quads that contain that vertex ID are located. If there are more than four quads containing the vertex, the direct conclusion is that more than four squares join together at a point. This is not physically possible for a rectilinear grid, and so identifies an ambiguity that must be rejected. No vertices are queued in this situation, as the correct neighbours cannot be identified.

Additional Filtering

A topological filtering stage is used to prune vertices from the grid structure that have only one edge connection. These correspond to nodes lying at the end of a single edge, usually near the image periphery, and have doubtful fidelity because there are no supporting topological connections from alternate sides. The pruning stage simply traverses the topological grid representation, and removes any vertices that have a single edge-connected neighbour.

A final filter removes any regions where repeated vertex IDs exist. These cases indicate that edges have formed a triangle, which is inconsistent with the physical grid structure.

5.2.9 Geometric Filtering

In seriously damaged or marginally lit portions of a grid image, regions of incorrect topology may pass through previous filtering stages. To catch these instances, the geometric properties of the grid topology are considered by correlating vertices with the connected component centroids that they represent. For each square of four vertices, the filter simply compares opposing edges, and verifies that they are of comparable length. Although the square may appear as a skewed rectangle after imaging, opposing edges should be relatively uniform in magnitude. Significant difference between them indicates a keystone or other physically implausible shape.

Examples of the topologies extracted by the described stages are shown in Figures 5.16, 5.17, and 5.18. For reference, the sequential filtering stages are:

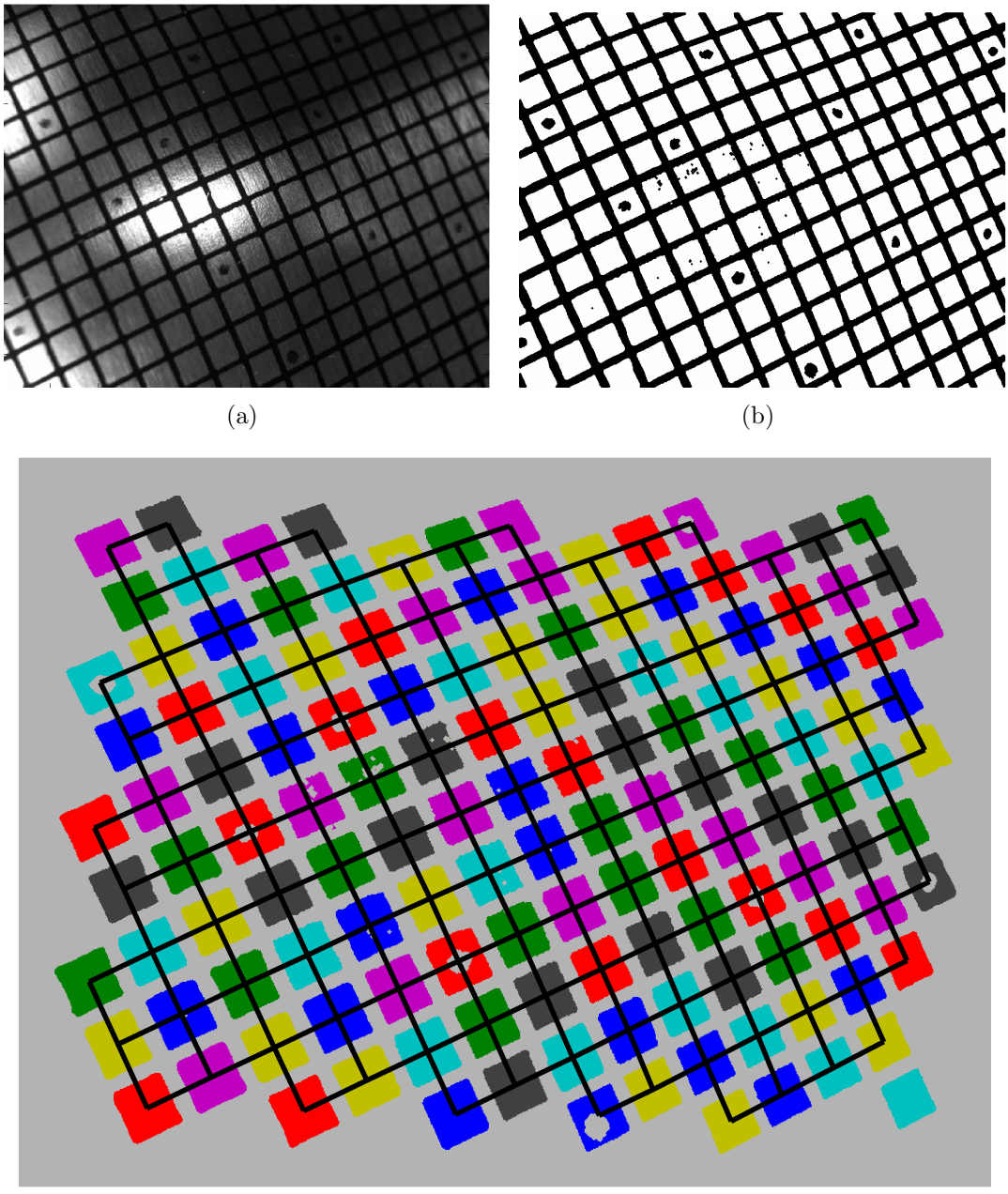
1. Topological: Unique vertex position in ordered quad structures.
2. Topological: At most four quads containing single vertex.
3. Topological: Pruning of terminal vertices that have no supporting edges.
4. Topological: Repeated vertex IDs are invalidated.
5. Geometric: Keystone and non-rhomboid grid geometries are rejected.

5.3 Interframe registration

To determine the interframe motion between two images, the topological grid representations are integrally shifted across one another. The best match in terms of an objective function is selected as the correct registration. A previous processing stage aligned the principal grid directions between frames, eliminating rotation as a degree of freedom in the registration. The resultant search space, consisting of discrete shifts of the grid topologies, is small enough that an exhaustive search is tractable.

5.3.1 Optimization Objective Function

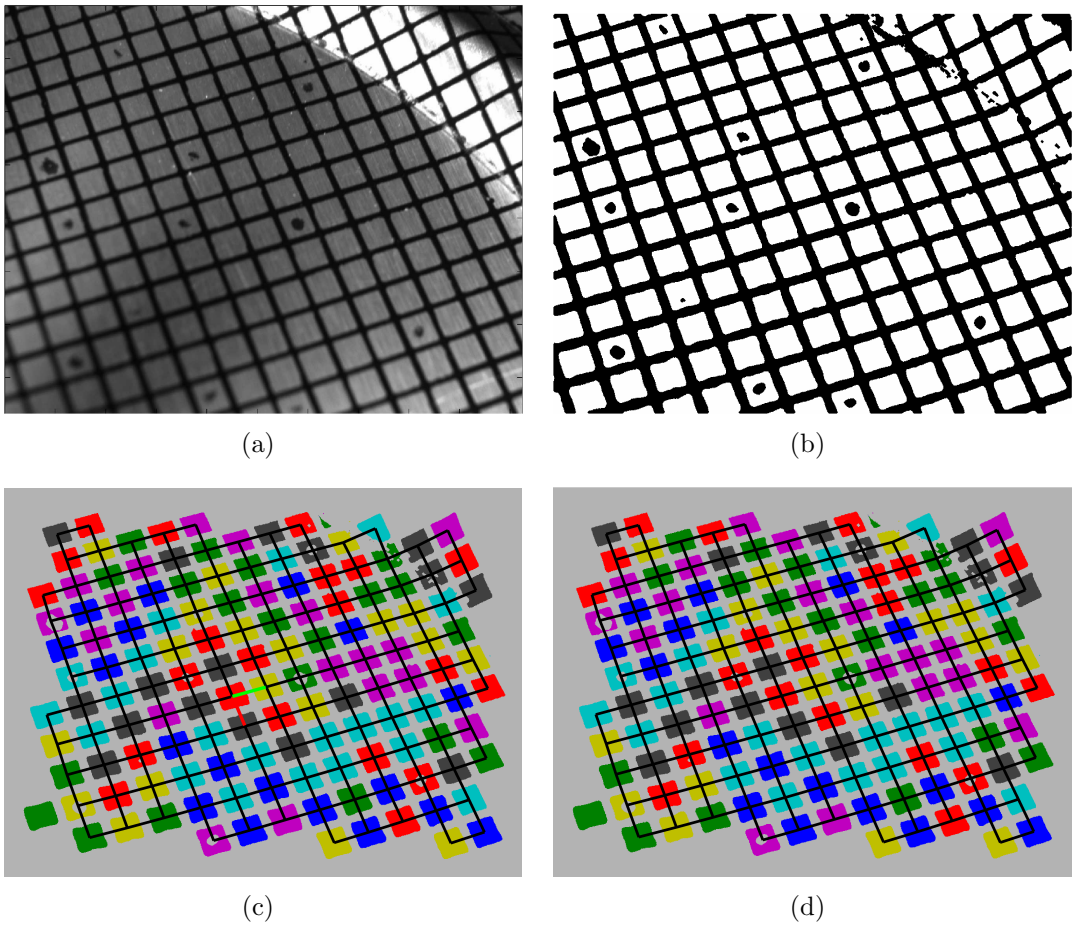
Optimization of the interframe grid motion is based on an objective function that measures the quality of grid registration for a given estimate. Using the topological



(a) (b)

(c)

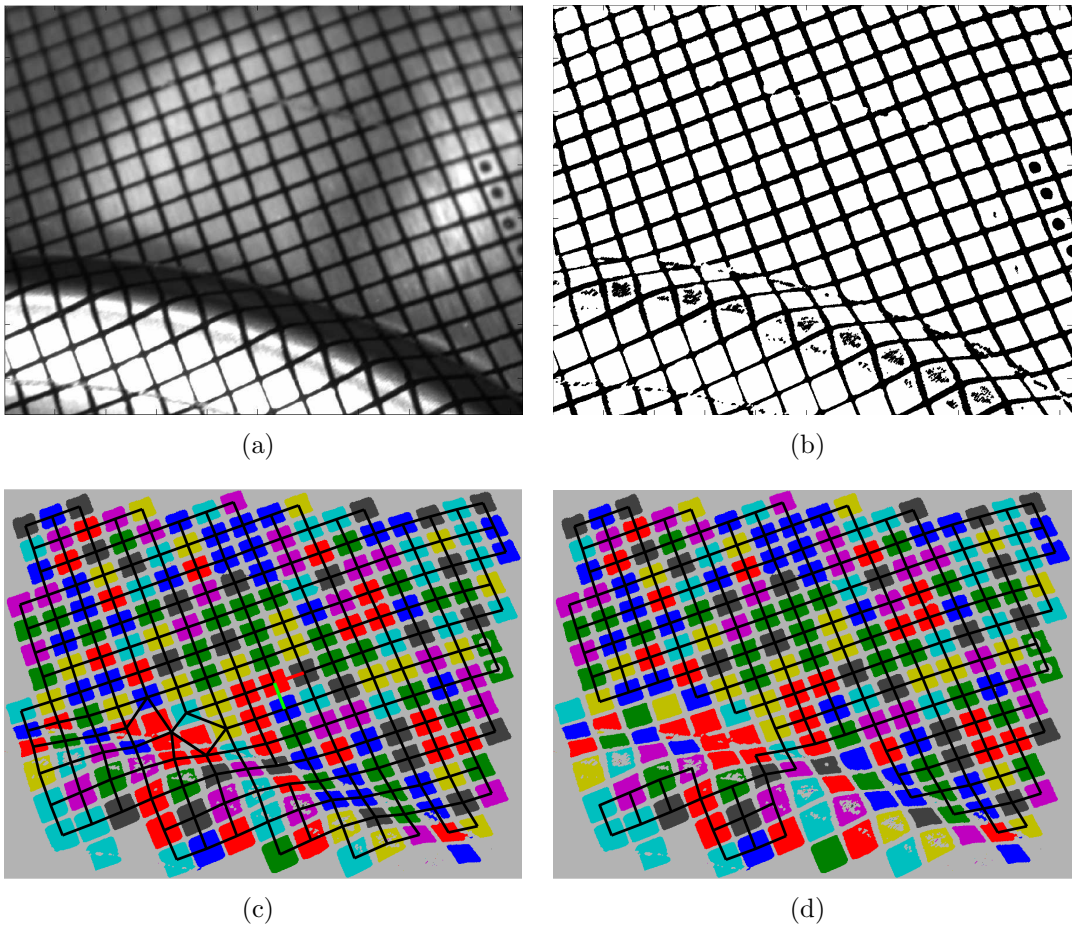
Figure 5.16: Sample topology data from a deformed metal dome.



(a) (b)

(c) (d)

Figure 5.17: Sample topology data from a deformed metal dome, with unfiltered output shown in (c), and filtered output in (d). For this image the original data is well formed, and no vertices are filtered out.



(a) (b)

(c) (d)

Figure 5.18: Sample topology data from a deformed metal dome, with unfiltered output shown in (c), and filtered output in (d). Grid damage and poor lighting in the high curvature region cause the extracted topology to be malformed. The filtering stages reject these regions.

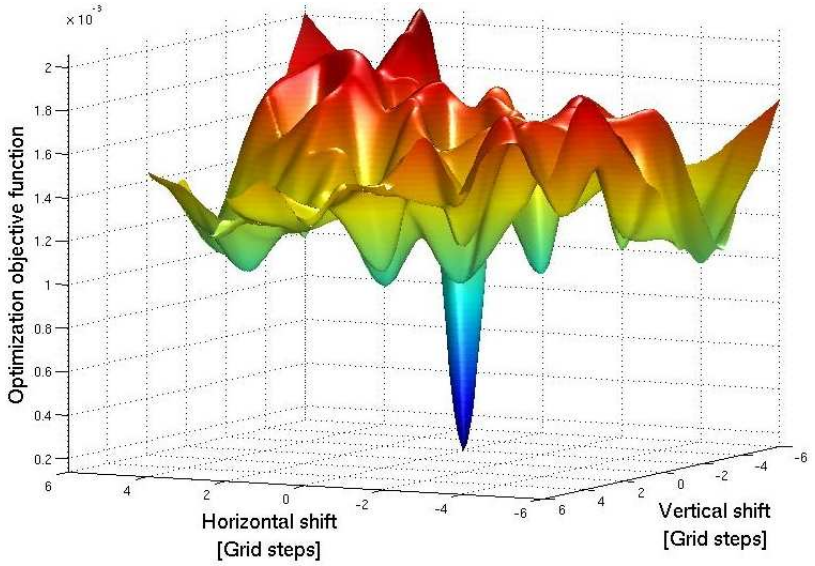


Figure 5.19: Optimization objective function for typical interframe motion, with large downward spike indicating the minimum objective value, and thus the best interframe grid registration.

grid representation (G_k for frame k), the optimization is fully decoupled from image coordinates or feature locations. The objective function $F(i,j)$ is the sum of squared differences between grid element metrics for an interframe shift of (i,j) grid units, plus a linear penalty term $P(i,j)$ based on the 2-norm of the shift.

The following objective function for the optimization is proposed:

$$F(i,j) = \sum_{\substack{(x,y)|(x,y)\in G^k \\ (x+i,y+j)\in G^{k-1}}} (\sigma^k_{x,y} - \sigma^{k-1}_{x+i,y+j})^2 + P(i,j) \tag{5.3}$$

The metric $\sigma^k_{(x,y)}$, as defined by Equation 5.2, is the ratio of thresholded pixels that are active within a given connected component (n_{thresh}), to the number of pixels in a filled/convex hull instance of the component (n_{total}). The coordinate convention (x,y) refers to the 2D location of the connected component in the topological representation.

A plot of the objective function (5.3) for typical interframe motion is shown in 5.19. The global minimum represents optimal interframe grid registration, and thus corresponds to the motion between frames.

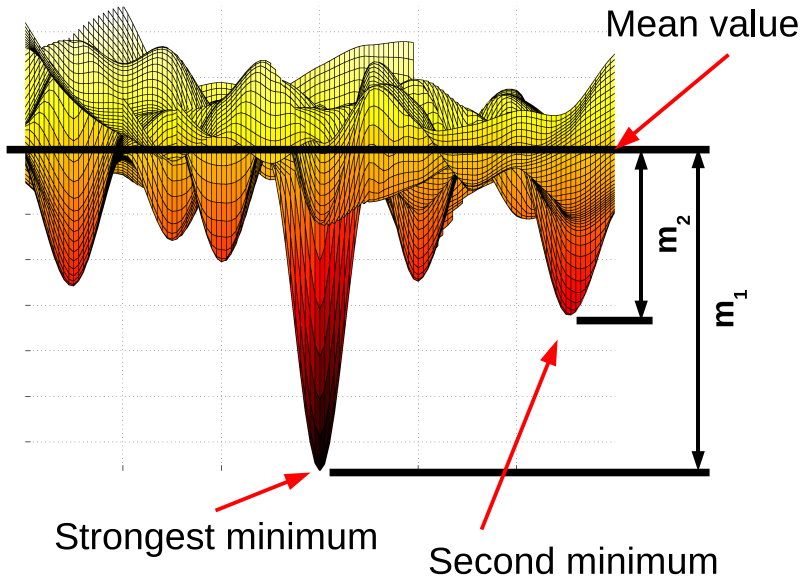


Figure 5.20: Objective function side view for a typical frame. Function mean value, and strongest two minima are marked with black lines. Measurements m_1 and m_2 used to define r_m are shown.

5.3.2 Interframe Transform Computation

The grid representation uses connected component centroids as vertices. Using the output from grid registration (optimization), each vertex can be cast from the topological representation back into image coordinates. Within this process, interframe correspondences between connected component centroids can be established. These correspondences provide the required input, in image coordinates, for computation of the interframe transform. It is important to note that although the topological grid shift is an integral number of grid units, once cast back into image coordinates, the transform is computed to pixel accuracy.

The homography relating the two image frames can be calculated using multiple techniques [60]. Computation typically involves a linear estimate followed by a non-linear optimization to refine the solution.

Table 5.1: Experimental results. Statistics on the ratio r_m are shown from four experimental video sequences, and the number of frame pairs with an r_m value larger than a threshold are listed. Correct interframe motion tracking was obtained for all frames.

Sequence number	Number frames	Mean of r_m	Standard deviation	Maximum value of r_m	# frames $r_m > 0.9$	# frames $r_m > 0.8$
1	1100	0.4740	0.0823	0.9100	1	2
2	550	0.4837	0.0790	0.8552	0	3
3	1800	0.4773	0.0794	0.8886	0	2
4	900	0.4865	0.0756	0.7997	0	0

5.4 Experimental results

Tests were performed on various experimental calibration data sequences to validate the algorithm and verify robustness in changing range, focus, and lighting conditions. Figure 5.21 shows an example result with both the current and previous frame grid topologies overlaid, and the detected interframe motion indicated by arrows. Table 5.1 shows results from four independent video sequences, with statistics provided on the ratio (r_m) between the magnitudes of the largest and second largest minima in the optimization objective function. Figure 5.20 shows the measurements m_1 and m_2, corresponding to the strongest and second strongest minima, respectively. From these, $r_m = \frac{|m_2|}{|m_1|}$. Magnitudes were measured relative to the objective function mean value, and the ratio r_m provides a measure of the contrast between the correct and second best (incorrect) grid registrations. All calculated interframe transforms were manually inspected to confirm correct algorithm output.

The results in Table 5.1 demonstrate robust performance by both the topological construction process and the optimization objective function. The rightmost two columns identify the number of frames for which the second strongest objective minimum was more than 90% and 80%, respectively, of the true minimum. These ratios indicate a lower objective contrast between the correct and an erroneous grid motion. Of the 4350 interframe transforms reported, less than 0.2% of the transforms had a similarity ratio above 80%, and only one frame generated a ratio above 90% (ratio of 0.91). The mean values of r_m report average similarities of less than 50%, with standard deviation smaller than 10%. These results demonstrate significant robustness of

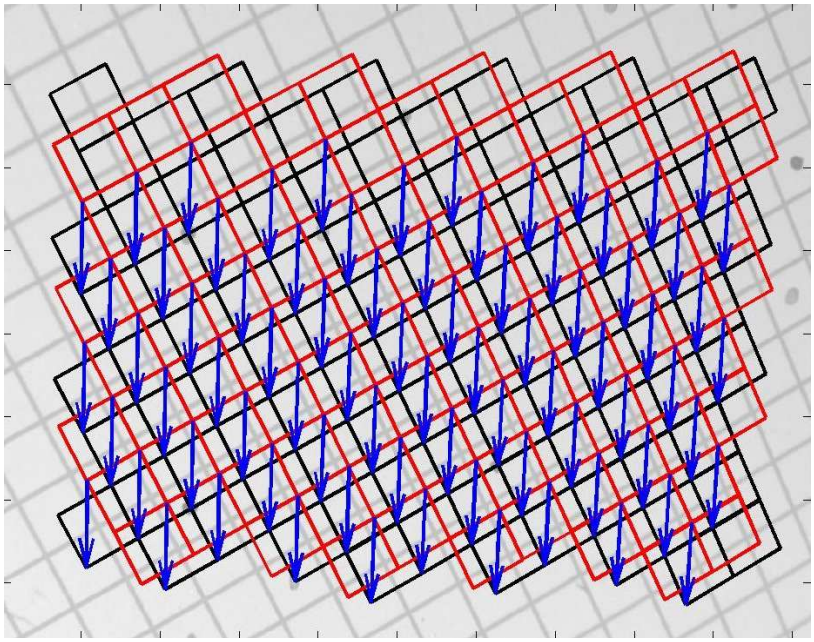


Figure 5.21: Interframe motion flow field. Current frame (dark/black lines) and previous frame (light/red lines) grid topologies shown with arrows indicating the detected interframe motion. Background image for current frame included.

the proposed algorithm in real imaging conditions, including damaged grid regions, and a variety of lighting and focus variations.

5.5 Summary of this Chapter

This chapter has described a novel and practical algorithm for accurate interframe tracking of line-based camera calibration grids larger than the camera field of view. The topological approach exploits the grid structure, and is used to reduce dimensionality of the projective tracking problem. Fiducial markers used to provide asymmetry in the grid have few design requirements, affording flexibility and avoiding the need for specific feature detectors. As an important stage in camera calibration and triangulation, this work extends the literature by exploiting structure unique to such applications.

Chapter 6

Calibration

Calibration is required for two aspects of the proposed surface measurement system. First is the camera itself, which must be calibrated to enable reconstruction of world geometry from image points. Second is the hand-eye relationship between the camera coordinate system and the coordinate measuring machine on which it is mounted. The hand-eye relationship allows position and pose information recorded by the CMM to be translated into camera location and orientation, and therefore used to triangulate points in a global coordinate frame.

This chapter briefly reviews techniques for camera and hand-eye calibrations, and describes an approach to calibrate the system parameters. The chapter then contributes two pre-processing algorithms that prepare the close-range image data as input for planar calibration techniques. The first algorithm coherently assigns calibration model coordinates to detected feature points throughout video sequences, and a second algorithm approximates the grid structure in highly blurred frames to enable model coordinate tracking.

6.1 Camera Calibration

6.1.1 Calibration Methods

Camera calibration involves resolution of intrinsic and extrinsic camera parameters, as introduced in Section 2.2.1, which model the relation between world coordinates

and image points. For a purely projective camera, linear techniques can be used to directly solve the camera calibration problem from known world coordinates and associated image points. Most modern camera models designed for reconstruction accuracy, however, include lens distortion parameters which are estimated using non-linear optimization techniques. Initialization of the problem is typically performed linearly by omitting lens distortions, followed by iterative nonlinear refinement of all parameters.

Early work in close-range photogrammetric calibration by Brown [11] modelled a camera including radial and decentering distortion parameters, which are common today in many practical camera models. The camera intrinsic parameters, including lens distortions, are directly affected by any changes to the zoom, focus, or aperture. Most calibration algorithms assume that intrinsic parameters are invariant across images, and because reconstruction accuracy is a driver of system performance, a fixed parameter camera is used for this work. Any changes to focus or aperture therefore necessitate recalibration of the camera. Modelling and calibration of cameras with automatic zoom lenses is considered in [174], with a resultant increase in camera model complexity.

In the context of camera intrinsic parameter calibration, four methods (and their derivatives) are common in the current literature. These are Tsai-based calibration [166], Zhang-based methods [185], Heikkila [62] methods, and auto-calibration approaches [164]. The Tsai, Zhang, and Heikkila methods require calibration targets with known feature geometries, while auto-calibration approaches use features in a natural scene to estimate the camera parameters. Early laboratory scale camera calibration techniques used three dimensional targets, often measured accurately using theodolites. Modern literature on calibrating from a 3D target includes Forbes et al. [46], in which a calibration cube with uniquely coded surface markers is used, and [184] which uses a non-coded target. Production and maintenance of accurate and stable three dimensional targets is expensive, so planar patterns are more common in practical calibration setups and are the focus of most current literature. Three dimensional objects typically produce more stable calibration output at the cost of target complexity, which as observed by [4], results from (1) uncoupling of the intrinsic and extrinsic parameters, and (2) depth information that reduces the coupling between

distortion and focal length parameters. Narrow depth of field precludes the use of three dimensional targets at close-range, so plane-based methods are exclusively applicable to this work. An interesting alternative approach, presented in [142], uses the touch probe of a CMM as a calibration target, allowing the probe to be moved around to produce a set of known calibration points suitable for the camera settings and conditions.

Related work includes Bruzzone and Mangili [13], who reported an approach for calibration of a camera fixed to the toolpoint of a gantry CMM. The technique uses only linear formulations, and iterates between solution of camera and distortion parameters. Non-linear optimization is avoided, and reconstructed point errors from ground truth were ≈ 0.5 mm (with 15-50 cm camera to target imaging range). This work was performed before the current generation of calibration techniques, such as the Zhang method, were developed.

Extensive literature is available on the common calibration techniques (Tsai, Zhang, and others), so the approaches are not described here beyond brief summary and reference to the critical publications.

Tsai Method

Although originally published in 1987, Tsai's camera calibration method [166] is well known today. Either a three-dimensional or planar calibration target can be used, and the model includes a single radial distortion coefficient. The technique is based on a formulation called the Radial Alignment Constraint (RAC). Modification was proposed by Zhuang et al. [187] to handle parallel orientation between calibration and image planes. Horn [67] performs a modern analysis of the Tsai technique, comparing the differences in required computation between 3D and planar targets, and proposes modification of parameter representations such as use of quaternions to represent rotation in the nonlinear optimization.

Zhang Method

The Zhang algorithm exploits multiple views of a planar calibration target, taken from a variety of poses. Homographies between each image and the calibration plane features are computed, and a closed form solution is used to initialize the camera

parameters. A nonlinear optimization is then performed to refine the parameters and to incorporate lens distortion. To obtain coherent homographies across a set of images, the coordinates assigned to the calibration target must be consistent across frames. A direct consequence of this requirement is the need for precise interframe motion tracking, which then enables globally coherent coordinate assignment to the calibration plane features.

As observed by Zhang, in many real cameras distortion is dominated by radial components, and more extensive models not only produce negligible improvements, but can produce numerical instability in the optimizations [185]. As a consequence, the Zhang approach as originally formulated accounts for two radial distortion coefficients, and no tangential or other more complex terms. A formulation of the Zhang calibration approach is described in Appendix B, so further details are not provided here.

Work by Ricolfe-Viala et al. [133] extends the Zhang method through pre-processing to correct for lens distortion, and data normalization (mean and average point offset) to better pose the calculations. They also observe that the position of model points should be included in the nonlinear optimization, as opposed to assuming correct target geometry.

A calibration algorithm that avoids nonlinear optimization is presented by Sanchez et al. [159]. The results of the approach are compared with the Zhang method, both before and after Levenberg-Marquardt optimization. The Sanchez calibration results are similar to the final Zhang post-optimization output, but are superior to the initial linear estimate produced by the Zhang approach, which disregards radial lens distortion. Final results from both methods are comparable.

Autocalibration Methods

A series of autocalibration approaches have been developed and are actively researched in the literature [153, 8, 162, 51], with the common element that a specific calibration target is not required. The methods instead derive calibration information from one or more views of a static scene. Such an approach provides significant flexibility by avoiding the need for specific targets or calibration configuration, but

adds complexity and uncertainty through lack of a known target geometry. In applications where reconstruction accuracy is required, autocalibration approaches are not yet reliable [9].

Avoiding Feature Extraction

A recent advance in camera calibration has been reported by Douxchamps et al. [34]. They observe that design of calibration features involves a tradeoff between large size for decreased noise sensitivity and good localization accuracy, versus small size for reduction of optical distortions. They then propose a state of the art calibration scheme that does not involve feature extraction, but rather uses an iterative estimate of the camera parameters to model the planar calibration target using raytracing. The raytraced image is compared with the real data, and iterative optimization used to refine the model parameters. Various sources of noise are modelled and tested, with detailed results and analysis. They further propose that this method is applicable not only to camera calibration, but also to general 3D reconstruction tasks. With further research and extension to optically accurate raytracing of DOF blurred images, this approach could potentially be applied to close-range imaging applications.

Bundle Adjustment

Bundle adjustment is a fundamental optimization process commonly used as a final stage in 3D reconstruction and camera calibration algorithms. The idea is that the world feature point locations and camera parameters are simultaneously adjusted to minimize image reprojection error. Triggs et al. [165] provide a survey and good description of the approaches for bundle adjustment, including formulation to exploit sparse structure in the underlying data relations. Further description is provided by Hartley [60].

6.1.2 Comparison of Calibration Techniques

With the large number of camera calibration techniques that have been proposed in the literature, an obvious question is how the techniques compare with each another, and which methods are best for specific applications. Key points of comparison relate

to accuracy and stability of the individual calibration parameters, and also resilience to noise in both the image feature detection and physical calibration target geometry.

Salvi et al. [135] describe the Tsai, Hall [56], Faugeras [38], and Weng [173] calibration algorithms using a common notation, and then compare them in the context of 3D reconstruction accuracy and image reprojection error using a common data set. A conclusion is drawn that nonlinear techniques are required to provide reconstruction error less than 0.1 mm (for the system tested), and furthermore that radial distortion modelling is sufficient when high accuracy is required. The Tsai method performed slightly better in terms of accuracy than the Wang method, and significantly better than the other algorithms tested.

Zollner and Sablatnig [188] compare the Tsai, Zhang, and Heikkila calibration methods. They concluded that all provide reliable results, but that the Tsai method is superior when using only a single view of a calibration plane with radial distortion. For calibration from multiple views, they conclude that the Zhang method exhibits superior convergence properties.

Gonzalez et al. [53] compare eight calibration methods, including the Tsai, Zhang, and Heikkila methods. They conclude that Batista's method [7] provides the most stable results, while Tsai's method produces stable extrinsic paramaters, but poor focal length estimates. They further conclude that the Heikkila algorithm estimates target distance effectively, but performs poorly for the remaining parameters. An important observation is that the reprojection error magnitudes produced by most tested algorithms after nonlinear optimization were comparable, but such a result does not imply that the camera parameters are correct. Comparable reprojection errors were obtained for significantly different calibration parameter results across the methods. This is likely a result of local minima in the optimization formulations, and also the existence of multiple solutions that similarly reduce the error. It is well known that initialization of a nonlinear optimization for calibration must be close to the true parameter values.

Sun and Cooperstock [155] studied the Tsai, Zhang, and Heikkila methods in the context of input data quantity and noise. For real world data, they find the Zhang method to be superior in the sense that it produces good results without meticulous attention paid to the experimental setup. The alternative methods considered were

found to be much more sensitive to measurement noise. Results are reported from the Zhang method at a target range of $\approx$ 40 cm, producing reprojection error on the order of pixel quantization error. Finally, Sun and Cooperstock conclude that a second order radial distortion term is sufficient for practical modelling, but that for lenses with totally unknown distortion characteristics, a decentering term is worth including in the formulation.

Swapna et al. [158] consider accuracy of calibration based on the Heikkila approach in the presence of various error sources. The camera intrinsic parameters are perturbed, and the effect on reprojection error analyzed. They draw the conclusion that interactions between the intrinsic parameters may prevent calibration from converging to a true minimum in the presence of image measurement error. Further simulation then leads to the conclusion that focal length and radial distortion errors lead to insignificant changes in reprojection error, making it difficult to calibrate these parameters. The paper then draws the conclusion that radial distortion need not be considered in camera modelling, a result contrary to a much of the published literature. A final experiment perturbs the calibration plane feature locations using Gaussian noise, and considers the sensitivity of the intrinsic parameters to this error. Resultant errors in the principal point were more than 4% of the nominal value (for errors of 0.1mm on the calibration plane), leading to the conclusion that the Heikkila algorithm is sensitive to error in the calibration target, and that printed targets should not be used for high accuracy calibration. In light of previous comparison work (such as [53]) which conclude that the Heikkila method does not produce stable results in the presence of noise, the conclusions from Swapna et al. are not surprising.

6.1.3 Accuracy of Calibration Patterns

Albarelli et al. [4] consider planar calibration patterns for which the physical feature locations are not exact, such as in laser printed grid approximations. The calibration object geometry is optimized alternately with the camera parameters to minimizing image reprojection errors, and calibration is based on the Zhang approach. This approach generalizes the calibration plane uncertainties proposed by Strobl et al. [152], and generates a number of important results. Of interest to this thesis are two primary results. First, by optimizing the calibration target model there is a

noticeable improvement in calibration accuracy ($\approx$ 60% reduction in metric error) for laser-printed calibration grids, which are commonly used in modern calibration. This result indicates that unless a high quality professional calibration target is used, calibration target errors should be considered in the formulation.

A second important result follows from the inclusion of two different images sets in the testing. The first set was obtained using camera imaging angles up to 15° from the plane surface normal, while the second set allowed the imaging angle to increase up to 30°. The 30° images produced better convergence, and less measurement error ($\approx$ 55% reduction in metric error) compared with the 15° data set. More importantly, at close imaging distances ($<$ 30 cm), there was noticeable gradient in the reconstruction output as the camera range changed. It is expected that with increased camera vergence angles, the reconstruction error should decrease because the problem is better constrained by the data set. In the close-range imaging application of this thesis, DOF blur prevents large camera vergence angles, although 30° imaging is reasonable. The Albarelli et al. results show that calibration from these angles can be stable, and that larger angles are preferred at close range. The improved accuracy from larger angles must be considered in concert with the reduction in grid intersection measurement accuracy caused by increasing DOF blur. Imaging angles on the order of 30° are a reasonable compromise.

Lavest et al. [84] examine whether an accurate physical 3D calibration pattern is required to obtain valid camera calibration results. For a single image of the calibration target and a least squares linearized approach, they find that a physical 3D target error larger than 2 mm^3 causes non-convergence of the algorithm. A multiple view calibration technique that adjusts the estimated feature points is then described, with the conclusion that subpixel accuracy of image point detection becomes more important than 3D target accuracy when multiple views are available. Their results imply that with image feature detection accurate to 0.01 pixels, calibration results are stable and accurate.

Mallon and Whelan [104] study the effect of calibration pattern choice in the presence of projective imaging and lens distortions, considering resultant biases from circular and checkerboard feature patterns. They find that only corner features (checkerboard) are free from lens distortion bias, and recommend that if circular dot patterns

are used, then each circle diameter should be less than 10 pixels as imaged.

6.1.4 Choice of Camera Calibration Technique

Within this work, a 2D calibration target is required because of narrow camera DOF. Multiple images of a planar target from a variety of poses can be obtained using the same data capture approach as in part scanning. For flexibility it is preferable to use a calibration target that does not have precisely pre-measured feature points (better than 1 part in 1000 measurement [152]), but rather has regular features that are approximately measured using accessible techniques such as an optical comparator. The calibration algorithm must then refine the model of the calibration plane feature locations to account for measurement errors. To be consistent in feature detection methodology, an orthogonal line-based grid is chosen for the planar calibration target, with measurement using the implementation of Chapter 4. The approach proposed by Strobl and Hirzinger [152] is therefore selected. It is based on the technique, which as demonstrated by multiple sources described previously, exhibits better convergence properties in the presence of noise compared to other well known algorithms. A consequence of the multiple calibration plane views, however, is the requirement for pre-processing of the close-range image data to prepare it for calibration use.

6.1.5 Hand-eye Calibration

Hand-eye calibration involves the determination of a rigid transform between an end effector coordinate system (robot or CMM), and the internal coordinate system of a camera or sensor mounted to the end effector tool point. Significant literature is available on this subject, such as [66, 32, 24], but a thorough review is beyond the scope of this thesis. Of relevance is recent work by Strobl and Hirzinger [151, 152]. The first publication ([151]) reviews the hand-eye calibration problem and existing literature, and then describes a technique for hand-eye calibration that aims to be optimal in the presence of Gaussian measurement errors. The second publication ([152]) addresses the issue of camera calibration target scaling uncertainty, and develops a calibration formulation based on the Zhang [185] approach. Target scale and aspect ratio are included in the non-linear optimization, with scaling information obtained from the

CMM/robot motion information that is not available in classic camera calibration. Two approaches for combined camera (Zhang-based) and hand-eye calibration are then described.

A significant portion of the modern hand-eye calibration work relies on quaternion [81, 57] representations of rotations, which provide advantages when rotations must be adjusted within non-linear optimizations.

The work by Strobl and Hirzinger [152] provides a direct approach that solves both the camera and hand-eye calibration parameters in the system described by this thesis, so is selected for implementation. The approach is based upon the Zhang calibration method, which is the preferred camera calibration approach (regardless of the Strobl work), as described previously. A prerequisite requirement for calibration, therefore, is a collection of image sequence pre-processing algorithms that prepare the image data for use by a Zhang-based calibration procedure.

6.2 Pre-processing for Camera Calibration

Zhang based calibration requires a homography to be computed for each image. The homography relates the image feature point coordinates to global coordinates of a model calibration plane. In typical applications of the Zhang algorithm, the entire calibration target is in view including boundaries. The plane boundaries can then be used to identify the unique calibration plane address of each imaged feature. In close-range imaging only a portion of the grid is visible, so global plane directions and coordinates must be coherently assigned to enable the calibration process. Chapter 5 described a novel algorithm to track interframe motion across a video sequence, which can be applied to tracking of the interframe calibration grid motion. This section describes two algorithms, one for coherent assignment of coordinates to the calibration plane using interframe transforms from the topological tracking approach, and the second to handle sequences of highly blurred frames without loss of grid assignment consistency.

6.2.1 Model Grid Coordinate Assignment

The calibration plane feature points must be assigned unique model coordinates, and although the scaling and orientation can be arbitrary, the assignment must be consistent across frames. Directions used as the axes in coordinate assignment should correspond to the principal directions of the target grid lines. Coordinate assignments to the calibration grid features are named the "model grid coordinates" in this work, with the "model grid" referring to features on the ideal calibration grid surface. To obtain views of the calibration plane from a variety of poses without losing coherence in the coordinate assignment, video sequences are captured, and motion between frames tracked using the algorithm described in Chapter 5.

Output from grid line intersection measurement is a set of feature points from the gridded surface, with no relative positioning in the grid structure available. Noise and damage to grid lines causes some intersections to be rejected, therefore leaving an incomplete set relative to a completed grid. Algorithm 6.1 has been developed to robustly assign model coordinates to the measured line intersections in an image, and is a required pre-processing step before homography computation and subsequent calibration.

Considering a set of grid line intersection features, there are multiple directions that encounter line intersections at regular intervals. Two of these directions correspond to the grid line axes, while the remainder form diagonals across the grid. To avoid ambiguity in projective imaging conditions, estimates of the grid line directions are output by the scale-space ridge detection algorithm. The matched ridge segments used to interpolate intersection locations are directly derived from the imaged grid lines, so provide a good estimate of the principal directions. A flat plane is used for calibration, so in practice the principal grid directions remain relatively consistent across an image.

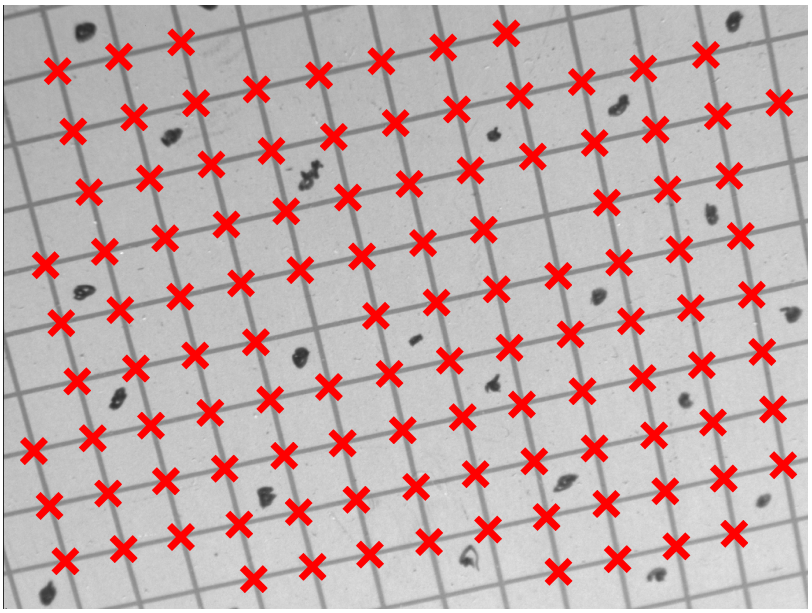
The direction vectors $D_1 = [x_1 \ y_1]^T$ and $D_2 = [x_2 \ y_2]^T$ encode not only the direction of the two principal grid directions, but also the expected distance between intersections in the corresponding direction. A model point $M_i = [x \ y]^T$ is assigned to the ith detected intersection point in an image, and corresponds to the "model grid coordinate" used in later calibration. These coordinates must be consistent across all images of the calibration plane. The model coordinates $M_j = [a \ b]^T$ are assigned

to the image coordinates of a detected feature j, with the image feature coordinates denoted by $I_j = [u\ v]^T$ (in the image coordinate system).

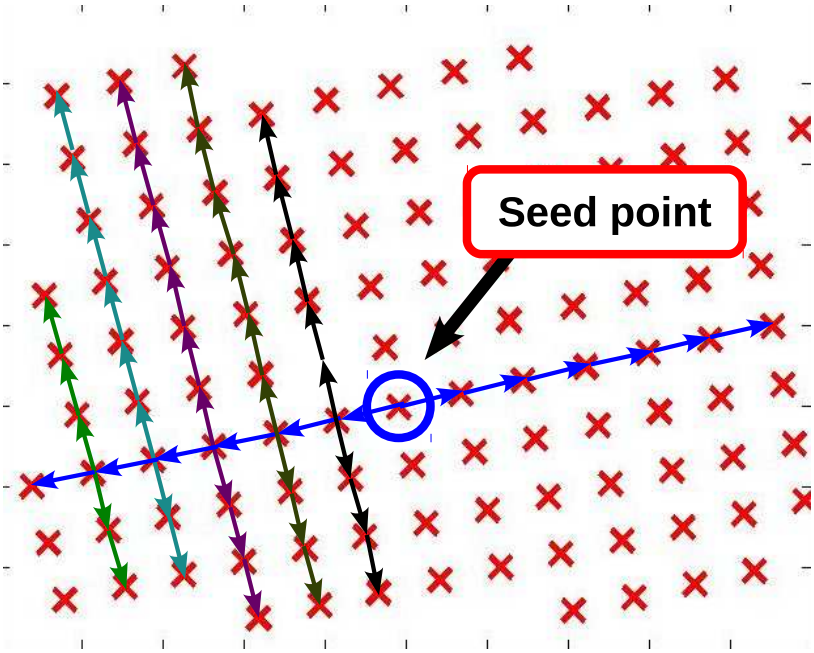
Algorithm 6.1 Model Grid Coordinate Assignment - Recursive Seeding

while Frames to process **do**
  if First frame **then**
    Find intersection with four nearly equidistant surrounding points (pairwise)
    Assign this intersection to I_{seed}
    $M_{seed} = [0\ 0]^T$
    Choose two neighbour points whose vectors correspond to principal grid directions estimated during scale-space ridge measurement
    if Chosen directions do not form a right handed system **then**
      Perturb order of directions to form right handed coordinate system
    end if
    Assign estimated principal grid directions to D_1, D_2
    AssignCoordinates(1, I_{seed}, M_{seed}) ▷ Algorithm 6.2
  else
    Match a current frame intersection to previous frame using inverse inter-frame transform (H^{-1})
    Assign matched intersection location and previously assigned model coordinates to I_{seed}, M_{seed}
    Correlate current frame principal grid directions to previous frame directions and assign as D_1, D_2
    AssignCoordinates(1, I_{seed}, M_{seed}) ▷ Algorithm 6.2
  end if
end while

Key features of the algorithm are that missed intersection points are tolerated, and the expected distance between grid feature points d_{local} adapts to local conditions as the algorithm progresses further from the seed point. Figure 6.1 shows the progression of the algorithm on a sample image, with colours representing different instances of the procedure. Figure 6.2 shows sample output after model coordinates have been assigned to detected grid line intersections.

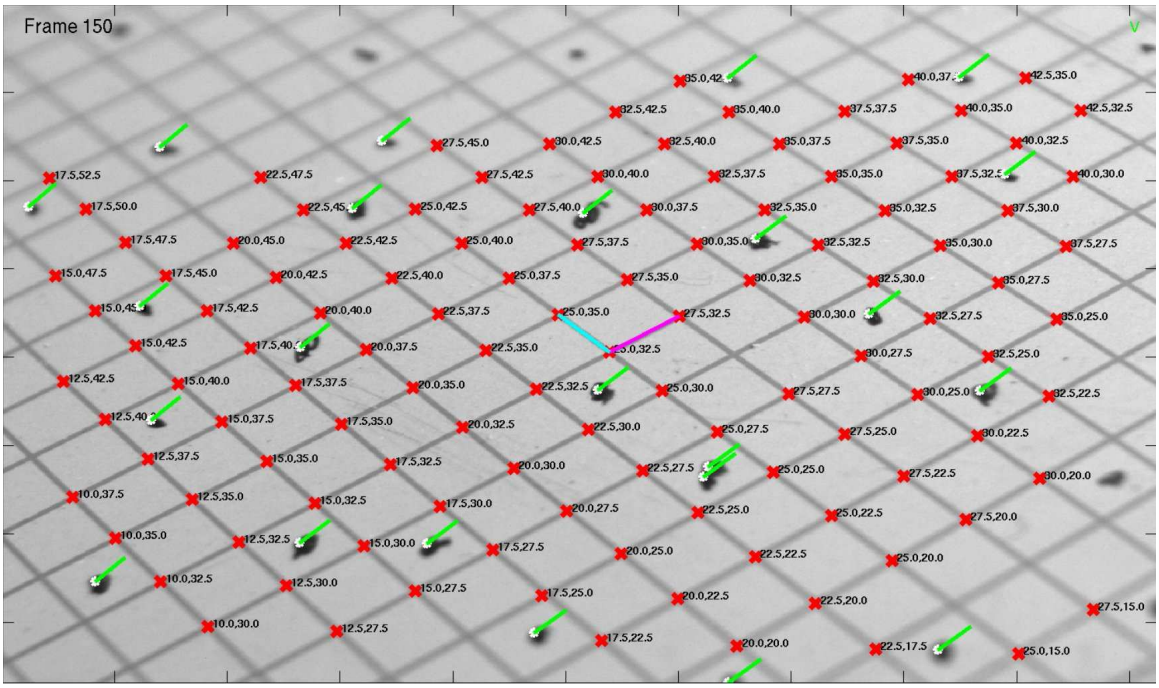


(a)

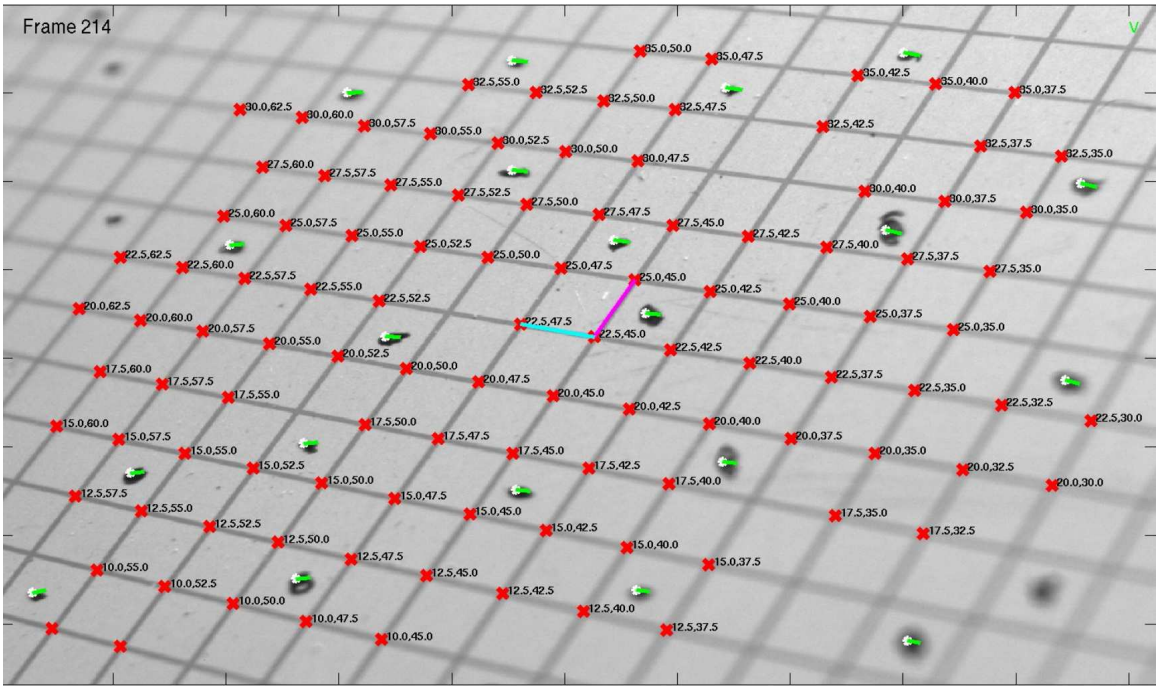


(b)

Figure 6.1: Intersection traversal example during model grid assignment. (a) Original image with detected intersections and, (b) arrows indicating path of model coordinate assignment traversal, with colours indicating different instances of the recursive assignment function.



(a) Sample frame 150.



(b) Sample frame 214.

Figure 6.2: Sample model grid coordinates assigned to detected feature points.

Algorithm 6.2 Recursive Coordinate Assignment

Require: Detected image feature set I, principal grid direction estimates D_1, D_2

procedure AssignCoordinates(n, I_{seed}, M_{seed})
  if $n > 2$ **then**
    return
  end if
  Search for detected intersections in direction $\pm D_n$**:**
  $d_{local} = D_n$
  while Searching inside image boundary **do**
    Image point $[u\ v]^T = [u\ v]^T \pm d_{local}$
    Update model coordinate M based on the direction n and grid unit steps taken from the seed point M_{seed}
    Find nearest intersection I_j to $[u\ v]^T$
    if Intersection within threshold distance **then**
      Assign model address M to intersection I_j
    end if
    AssignCoordinates(n+1, $[u\ v]^T$, M)
    Update d_{local} as vector from previous point $[u_{t-1}\ v_{t-1}]^T$ to current point $I = [u_t\ v_t]^T$
  end while
end procedure

6.2.2 Hough Approximation for Strongly Blurred Frames

Coherent assignment of coordinates requires an assigned intersection in a previous frame to be correlated with an intersection in the current frame. This correlation provides a translational grid registration between frames, and leads to the coordinate seed from which recursive assignment across an image begins. When frames are highly blurred, the scale-space measurement process may reject most or all intersections due to uncertainty in parabola fitting. These highly blurred situations often occur in a consecutive sequence of frames as the camera operator reacquires reasonable focus. Registering an intersection correspondence across the poor frames requires that the individual interframe transforms across the blurred sequence be compounded to form a composite interframe transform, bridging the severely blurred set.

In practice, registering coordinates across blurred sequences using compounded interframe transforms is not reliable. The true interframe transform should ideally be computed from corresponding feature points at either end of the blurred sequence, but these correspondences are not available. The estimated transform (from individual interframe tracking) contains compounded errors, and in practice leads to incorrect feature correlation across sequences on the order of 20-30 blurred frames.

To resolve the registration problem across blurred sequences, coarse grid line intersection locations are extracted, and coordinates assigned to these points in the same manner as accurately measured intersections. Although the coarse estimates cannot be used for calibration purposes, they effectively correct the grid registration at each subsequent frame, preventing large compounding of errors and incorrect registrations.

The contributed coarse intersection location algorithm uses a Hough-based line detection approach, with a k-means clustering to classify the grid line directions. The process is summarized in Algorithm 6.3.

As described in Chapter 5.2.1, the Shafait [141] local adaptive thresholding method performs well when binarizing close-range images of a gridded target. Using the thresholded output, the well known Hough transform [35] is used to estimate the characteristics of lines in the image. Matlab implementations of the Hough transform and supporting functions were used for this work, as provided in the MathWorks Image Processing Toolbox. To provide robustness, a hierarchical back-off approach is used when selecting the Hough algorithm parameters. The image is first tested

Algorithm 6.3 Coarse Intersection Estimation

Image threshold operation (Shafait-based)
repeat
  Calculate Hough transform
  Extract peaks from Hough accumulator bins
  Extract line segments corresponding to Hough peaks and generate statistics
  Relax Hough parameters (to prepare for next iteration if it is required)
until More than threshold number of lines detected or iteration limit reached
Classify lines according to angle (batch 2-means)
Reject lines not belonging to the primary angle classes
Intersect lines from disparate angle classes
Record intersections that fall within the image boundaries

Table 6.1: Implemented Hough Parameters

Pass	Threshold (% max accumulator value)	Suppression radius	Max peaks
1	70%	$\frac{1}{60}$ image dimension	1000
2	60%	$\frac{1}{60}$ image dimension	1000
3	40%	$\frac{1}{40}$ image dimension	10000

with strict parameters, and if a suitable number of grid lines are not detected, then they are relaxed and the process repeated. The set of Hough parameters used for this work, which were successful on hundreds of image frames tested, are listed in Table 6.1.

Following the Hough transform and accumulator bin peak detections, line segments are extracted from the image corresponding to the accumulator bin peaks. Disconnected line segments falling into the same parameter bin are connected, and segments less than approximately half of the image dimensions are rejected. The remaining set of lines are summarized in terms of angle and perpendicular distance from origin in preparation for classification.

Perpendicular grid lines must be intersected to approximate the true line intersections, so the well known k-means clustering algorithm [36] is used to isolate the two primary modes of line direction, and to reject outlier lines corresponding to noise and surface artifacts. At a basic level, the k-means algorithm identifies clusters of points in a parameter space by minimizing the aggregate distances between cluster means and points across the data set. In the case of the grid line angle classification,

there should be two primary clusters corresponding to the two sets of lines that are perpendicular to each other. Under perspective imaging the lines may not appear to be orthogonal, but they still form disparate angle classes suitable for classification through cluster analysis. Batch 2-means clustering is therefore applied.

The angle of a line with respect to the primary image axis is the driving parameter behind clustering. The difference between angles is inherently a cyclic function, with an integral difference of 2π radians corresponding to the same angle. The grid lines to be detected are not directed, so integral differences of π radians in fact correspond to the same line direction. During the k-means clustering process, a distance function must be provided to calculate the distance between a specific point and the cluster means in the parameter space. The cyclic nature of angular distance precludes the use of typical metrics such as Euclidean or Hamming distances. Instead, a custom distance function was developed for this application, and can be expressed as:

$$d_k = 1 - |cos(\theta_1 - \theta_2)| \tag{6.1}$$

A plot of the distance function is shown in Figure 6.3, with all angles expressed in radians. A linear distance function approximating the shape of Equation 6.1 could be constructed, but the cosine based approach has an advantage of reduced distance sensitivity for small angular values.

In images that are highly blurred or that have large surface defects, erroneous lines may be extracted during the Hough process. These lines typically appear as outliers from the k-means clusters, so can be rejected based on distance from the parent cluster mean. For this work, lines with an angular distance of more than $\frac{\pi}{8}$ radians from the mean are rejected. This filtering removes most outlier lines, but any that survive form approximated intersections on the image plane. These are then rejected either during the model coordinate assignment process (which looks for integral grid spacing steps between points), or are rejected as outlier point correspondences during homography computation immediately prior to camera calibration.

Figures 6.4 and 6.5 show typical output from the intersection estimation process for blurred image frames. The detected Hough lines are shown in (a), followed by the k-means classified output in (b) with the class represented through colour. Lines detected through the Hough process but not considered to be inliers in the k-means

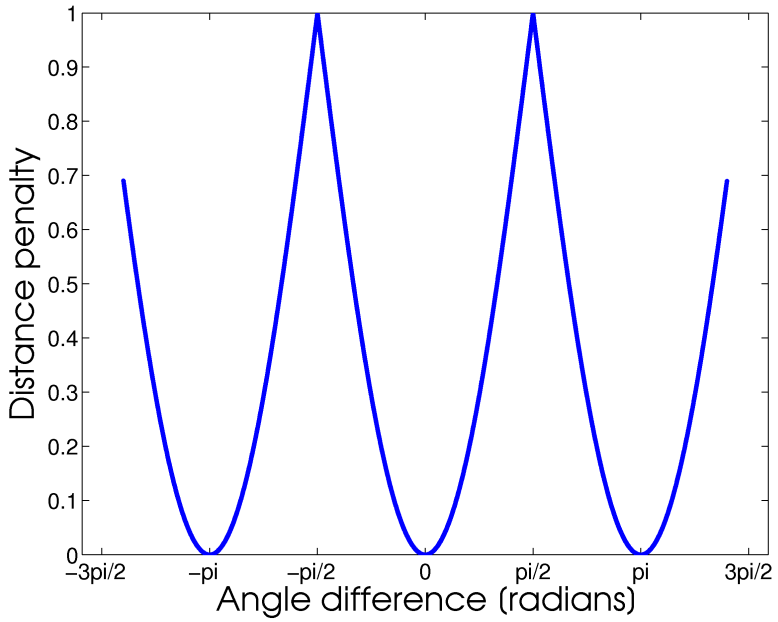
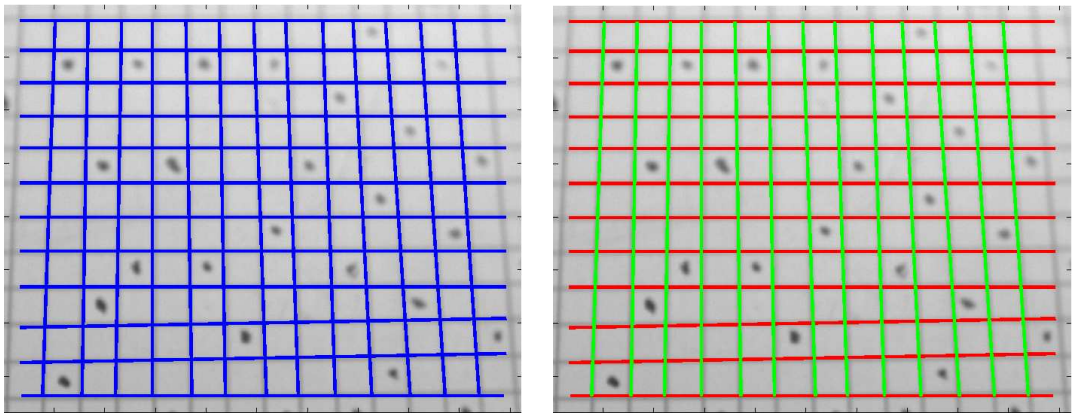


Figure 6.3: Distance function for k-means clustering of line angles.

clusters are rejected, as seen in Figure 6.5.

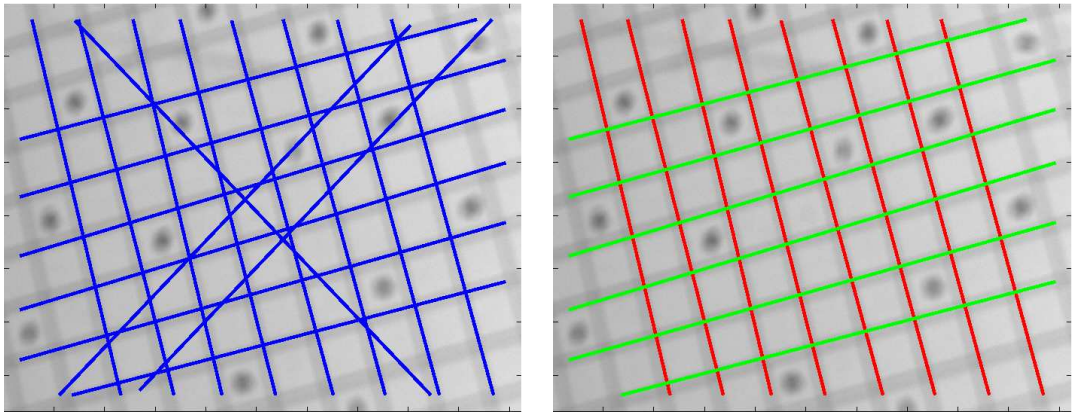
6.3 Summary of this Chapter

This chapter has reviewed state of the art literature on camera calibration and selected an approach for both camera and hand-eye calibration of the monocular CMM-based system developed by this thesis. Pre-processing algorithms have been contributed that enable the use of close-range planar image data in Zhang-based calibration techniques. Further description of the Zhang calibration algorithm is provided in Appendix B.



(a) (b)

Figure 6.4: Sample output from grid line approximation. (a) Detected Hough lines, (b) k-means classification shown by line brightness/colour.



(a) (b)

Figure 6.5: Sample output from grid line approximation, with outlier lines rejected by the k-means classification. (a) Detected Hough lines, (b) k-means classification shown by line brightness/colour.

Chapter 7

Conclusions

This thesis has presented a system design for close-range, gridded surface measurement, and has contributed algorithms that solve a number of challenges, principally related to narrow depth of field effects. The motivating sheet metal surface strain application was described by Chapter 1, and development of a monocular CMM-mounted camera system concept provided by Chapter 2 included reasoning behind selection of the close-range imaging approach. Human manipulated imaging of surfaces at close-range leads to a number of challenges that were addressed by the remainder of the thesis.

Chapter 3 examined the problem of accurate grid line intersection measurement in the presence of varying target range and DOF blur. Approaches using standard feature detectors were considered, followed by description of a scale-space ridge extraction based technique to retrieve dense grid line data in varying scale conditions. Computational intensity of the algorithm was considered, with the conclusion that the algorithm is not suited for video frame-rate implementation on a CPU.

Chapter 4 provided an introduction to GPU hardware in the context of general computational acceleration, followed by description of the algorithm design for GPGPU-based acceleration of the computationally intensive scale-space algorithm. Accuracy analysis using synthetic raytraced data has validated subpixel measurement performance in the presence of varying levels of DOF blur, producing error variance levels on the order of 0.05-0.1 pixels^2 for DOF blurred images. Speed and resource utilization results from GPGPU implementation were presented, demonstrating video

frame-rate processing speeds.

Finally, robustness of line intersection measurement in real images was experimentally shown using a series of video frames, demonstrating that missed intersection detections are not persistent, except in the presence of physical grid line damage.

From a triangulation error perspective, close-range imaging can provide an improved formulation for high accuracy measurement in the presence of noise. Multiple results have confirmed that grid line intersection measurements from close-range imaging, even when corrupted by DOF blur, can be performed robustly. It was further confirmed that scale-space ridge extraction is a suitable approach for close-range grid line measurement, and that the scale-space algorithm can be accelerated to video frame rates with state-of-the-art GPGPU hardware using the proposed architecture.

Chapter 5 described a novel approach for interframe motion tracking of gridded surfaces in the presence of varying DOF blur. The approach exploits topological structure of the grid to reduce dimensionality of the interframe registration problem, and intrinsically measures fiducial markers, removing the need for explicit feature detectors.

Implementation results were presented demonstrating robustness through statistical metrics on the optimization objective function, and through processing of multiple video sequences in which correct and accurate tracking has been manually verified. Contrast results between the objective global minimum and the second strongest minimum demonstrate algorithm robustness across thousands of video frames, including a variety of lighting, focus, and grid damage conditions.

System calibration was considered in Chapter 6, and a general approach put forward. Close-range imaging introduces challenges not encountered by classic calibration target views, so pre-processing algorithms were proposed and described such that the image sequences can be used as input for calibration algorithms. Sample results were presented from the model grid coordinate assignment and blurred grid approximation algorithms. Testing on image sequences with thousands of frames, including manual verification of correct grid coordinate assignment, has been performed. The techniques contributed by this chapter enable use of close-range image data for calibration using modern, state-of-the-art algorithms.

The techniques and algorithms contributed by this thesis make significant progress

toward enabling video frame rate, close-range optical computer vision for sheet metal surface strain analysis, and other applications where challenging image conditions impede conventional surface measurement.

7.1 Future Work

The contributions by this thesis solve a number of challenges imposed by close-range imaging conditions, and lead into a major subsequent research task, that of system calibration and calibrated parameter stability analysis. This thesis, while restricted to the machine vision aspects required to enable the calibration and part scanning processes, has provided the necessary techniques to enable calibration. Zhang-based [185] calibration using the modifications and extensions proposed by Strobl and Hirzinger [152] may provide the basis of future work.

In the context of calibration, future work can include a detailed study of the stability of parameters as affected by camera pose uniqueness, target distances, and spatial coverage of the calibration target. A trade-off analysis between the camera-to-target vergence angle and consequent DOF blur can be performed to suggest the preferred camera viewing angles that produce stable and reliable results in the presence of narrow DOF. It is expected that translating the camera to view a large portion of the calibration grid surface should produce improved calibration stability in combined camera/hand-eye calibration (reduced affect of CMM measurement errors), but this remains an open problem to be studied and verified.

The approach developed in this thesis requires human manipulation of a monocular camera to avoid robotic path programming, and to provide flexibility in rapid and adaptive scanning by an operator. Human manual manipulation of the close-range camera can lead to poor focus in many image frames and, potentially, regions of a scanned part that have insufficient data for accurate reconstruction. Some companies now offer articulated arm CMMs that include motorized joints. During normal measurement, a human can manipulate the arm while it passively measures position and pose. The motorized joints could then subsequently be activated to automatically follow the same path as taught by the human operator, or to perform other motions.

A future extension of the current system could include a motorized arm CMM, allowing a human operator to roughly scan the part surface to provide an approximated geometry, without sacrificing flexibility in industrial environments. The motorized arm could then automatically rescan the surface, correcting for range and speed related focus problems, and providing additional views of regions with insufficient data. The key enabler for this type of approach, in addition to the robotic CMM, would be the use of scale-space ridge detection to provide a measure of the level of blur in a specific portion of the image (scale of detection). The surface grid lines applied on a part surface are typically of uniform width, so detection of scale could be directly used as a measure of the blur. This information can provide feedback for automated optical guidance of the robotic CMM, allowing correction of focus without sacrificing system flexibility.

Within this thesis, a line based grid pattern was chosen for metal surface marking, as justified in Chapter 2. To obtain fine strain gradient measurement in large curvature parts, the marking pattern could be modified to a finer scale. A natural feature that can be tightly packed is a simple dot, so part surfaces could be covered in fine scale dots prior to deformation. Approaches for production of this pattern include photographic and inkjet printing. Optical surface scanning allows for a variety of feature patterns to be detected, and close-range imaging enables measurement of a fine feature scale. The resultant strain gradients should therefore be significantly finer than those obtainable from long range imaging, and provide a means to reconstruct surface strains on thinner materials and sharper curvatures that are increasingly common in manufacturing.

The future work proposed in this section would directly leverage and build upon the system design and algorithms developed throughout the thesis. The contributions therefore potentially enable a spectrum of future work in practical close-range surface scanning applications.

Appendix A

POV-Ray Scripts for Synthetic Grid Sequence

The following scripts were written to produce rendered grid images that emulate the real camera images of a gridded sheet metal surface. They execute in the POV-Ray environment [129].

A.1 GridGen.ini

```
Antialias=Off
Antialias_Threshold=0.1
Antialias_Depth=2
Input_File_Name="run1.pov"
Initial_Frame=1
Final_Frame=15
Initial_Clock=0
Final_Clock=2
Cyclic_Animation=off
Pause_when_Done=off
```

A.2 run1.pov

```

```

```
2 #include "colors.inc"
3   background { color Cyan }
4
5 #declare A = <0, 0, -1>;
6 #declare B = <0, 0,  0>;
7
8 // ANIMATION TRANSLATE!
9 #declare planeTranslate = <0,0 + clock*1,1>;
10
11 // ANIMATION ROTATE!
12 #declare planeRotateX = (0+clock*10)*x;
13 #declare planeRotateY = (5+clock*10)*y;
14 #declare planeRotateZ = (-50+clock*30)*z;
15
16 #declare extent = 10;
17 #declare halfLineWidth = 0.008;
18
19 #declare thePlate = box {
20         <-1*extent,-1*extent, 0>,
21         <extent,extent, 0>
22         pigment {
23                 color White
24                 //color rgb<0.999,0.999,0.999>
25         }
26 }
27
28 object{thePlate
29         rotate planeRotateX
30         rotate planeRotateY
31         rotate planeRotateZ
32         translate planeTranslate
33 }
34
35
36 #declare theLine = box {
37     <-1*halfLineWidth, -1*extent,   0>, // Near lower left corner
38     <halfLineWidth, extent,  -0.0000001>   // Far upper right corner
39         pigment {
40                 //color rgb<0.9,0.9,0.9>
```

```
41                  color Gray75
42          }
43 }
44
45 #declare halfNumHorizLines = 30;
46 #declare halfHorizLineSpacing = 0.1;
47 #declare halfNumVertLines = 30;
48 #declare halfVertLineSpacing = 0.1;
49
50 // Horizontal grid lines
51 #declare IndY = 0;
52 #while(IndY < halfNumHorizLines)
53          object{theLine
54                  translate <((1+2*IndY)*halfHorizLineSpacing), 0,0>
55                  rotate planeRotateX
56                  rotate planeRotateY
57                  rotate planeRotateZ
58                  translate planeTranslate
59          }
60          object{theLine
61                  translate <(-1*(1+2*IndY)*halfHorizLineSpacing), 0,0>
62
63                  rotate planeRotateX
64                  rotate planeRotateY
65                  rotate planeRotateZ
66                  translate planeTranslate
67          }
68        #declare IndY = IndY+1;
69 #end
70
71 // Vertical grid lines
72 #declare IndX = 0;
73 #while(IndX < halfNumHorizLines)
74          object{theLine
75                  rotate z*90
76                  translate <0, ((1+2*IndX)*halfHorizLineSpacing), 0>
77
78                  rotate planeRotateX
79                  rotate planeRotateY
```

```
80                  rotate planeRotateZ
81                  translate planeTranslate
82          }
83          object{theLine
84                  rotate z*90
85                  translate <0, (-1*(1+2*IndX)*halfHorizLineSpacing), 0>
86
87                  rotate planeRotateX
88                  rotate planeRotateY
89                  rotate planeRotateZ
90                  translate planeTranslate
91          }
92        #declare IndX = IndX+1;
93 #end
94
95 #declare cyl1 = cylinder {
96     <0, 0, 1>, <0, 0, 1.000001>, 0.0000001
97     pigment { color Blue }
98 }
99
100 #declare thisFocalPoint = trace(thePlate, A, B-A);
101
102 camera {
103         location A
104         look_at B
105         //location <-.1, .2, -.1>
106         //look_at  <2, -1, 2>
107         focal_point <0,0,1>
108         aperture 0.06    // almost everything is in focus
109         blur_samples 100      // more samples, higher quality image
110   }
111
112 #declare ImageWidth = 1024;
113 #declare ImageHeight = 768;
114
115 #declare lightPosition = <0,0,-1>;
116 #declare areaLightDim1 = <2*extent,0,0>;
117 #declare areaLightDim2 = <0,2*extent,0>;
118
```

```
119 light_source { lightPosition
120 color White
121 }
122 global_settings { ambient_light White }
```

Appendix B

Derivation of Zhang Calibration

The Zhang calibration technique [185] is common in the literature, but a full derivation is not typically provided. This appendix contains a derivation that follows the process developed and published by Zhang. The more complete development provided here is useful when implementing and verifying associated algorithms. For detailed justifications of steps and further textual descriptions, the reader is referred to [185].

B.1 Constraints

Begin with the pinhole imaging equation:

$$s_c \tilde{\mathbf{m}} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{M}} \tag{B.1}$$

where s_c is an arbitrary scale factor, $\tilde{\mathbf{m}}$ a homogeneous image point, and $\tilde{\mathbf{M}}$ a homogeneous world point. Rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ define the camera extrinsic parameters, and the upper triangular matrix $\mathbf{A}$ contains the intrinsic parameters:

$$\mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.2}$$

u_0 and v_0 are principle point coordinates, α and β are the scaling factors, and γ

defines skew between axes.

Define the Z coordinate of the model plane to be $Z = 0$, leading to:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{B.3}$$

$$= \mathbf{A} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{B.4}$$

$$= \mathbf{A} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{B.5}$$

where $\mathbf{r}_i$ is the i-th column of the rotation matrix $\mathbf{R}$.

As a result of the model plane being defined on the $Z = 0$ plane, and redefining $\tilde{\mathbf{M}} = \begin{bmatrix} X & Y & 1 \end{bmatrix}^T$, the image and model points are related as:

$$s\tilde{\mathbf{m}} = \mathbf{A} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \tilde{\mathbf{M}} \tag{B.6}$$

$$s\tilde{\mathbf{m}} = \mathbf{H}\tilde{\mathbf{M}} \tag{B.7}$$

where $\tilde{\mathbf{m}}$ are the two-dimensional image coordinates, and $\tilde{\mathbf{M}}$ are the two-dimensional model coordinates obtained by defining $Z = 0$.

The homography between the calibration target model and image coordinates ($\mathbf{H}$) is defined up to scale, so defining λ as an arbitrary scaling factor, we have:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix} = \lambda \mathbf{A} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \tag{B.8}$$

Computation of the homography $\mathbf{H}$ for each image to be included in the calibration is obtained using standard techniques, such as those described by [60]. The calculation begins with a normalized linear least squares estimate, followed by non-linear

optimization.

By definition, the columns of a rotation matrix are orthonormal. Using the orthogonality we have:

$$\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \lambda\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{B.9}$$

$$\frac{1}{\lambda}\mathbf{A}^{-1}\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{B.10}$$

$$\mathbf{r}_1 \cdot \mathbf{r}_2 = 0 \Rightarrow \mathbf{r}_1^T\mathbf{r}_2 = 0 \tag{B.11}$$

$$(\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_1)^T(\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_2) = 0 \tag{B.12}$$

$$\mathbf{h}_1^T\frac{1}{\lambda}\mathbf{A}^{-T}(\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_2) = 0 \tag{B.13}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 = 0 \tag{B.14}$$

Similarly, using the unit norm of the rotation matrix columns, we obtain

$$\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \lambda\mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{B.15}$$

$$\frac{1}{\lambda}\mathbf{A}^{-1}\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{B.16}$$

$$\mathbf{r}_n \cdot \mathbf{r}_n = 1 \Rightarrow \mathbf{r}_n^T\mathbf{r}_n = 1 \tag{B.17}$$

$$(\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_1)^T(\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_1) = 1 \quad \text{and} \quad (\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_2)^T(\frac{1}{\lambda}\mathbf{A}^{-1}\mathbf{h}_2) = 1 \tag{B.18}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_1 = \frac{1}{\lambda^2} \quad \text{and} \quad \mathbf{h}_2^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 = \frac{1}{\lambda^2} \tag{B.19}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_1 = \mathbf{h}_2^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 \tag{B.20}$$

Resulting are the two constraints that Zhang's calibration approach leverages. These constraints are repeated for convenience as Equations (B.21) and (B.22).

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 = 0 \tag{B.21}$$

$$\mathbf{h}_1^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_1 = \mathbf{h}_2^T\mathbf{A}^{-T}\mathbf{A}^{-1}\mathbf{h}_2 \tag{B.22}$$

B.2 Analytic Initialization of Solution

The term $\mathbf{A}^{-T}\mathbf{A}^{-1}$ appears in constraints (B.21) and (B.22), so we define this term as $\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1}$. The first goal of this section is to explicitly write the matrix $\mathbf{B}$ in terms of the original expression for $\mathbf{A}$ (B.2). We start by deriving an expression for $\mathbf{A}^{-1}$. This can be accomplished in many ways including inspection, but here we use Gauss-Jordan elimination on an identity augmented copy of the matrix $\mathbf{A}$.

$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{I} \end{array}\right] = \left[\begin{array}{ccc|ccc} \alpha & \gamma & u_0 & 1 & 0 & 0 \\ 0 & \beta & v_0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{B.23}$$

$$= \left[\begin{array}{ccc|ccc} 1 & \frac{\gamma}{\alpha} & \frac{u_0}{\alpha} & \frac{1}{\alpha} & 0 & 0 \\ 0 & \beta & v_0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{B.24}$$

$$= \left[\begin{array}{ccc|ccc} 1 & \frac{\gamma}{\alpha} & \frac{u_0}{\alpha} & \frac{1}{\alpha} & 0 & 0 \\ 0 & 1 & \frac{v_0}{\beta} & 0 & \frac{1}{\beta} & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{B.25}$$

$$= \left[\begin{array}{ccc|ccc} 1 & 0 & \frac{u_0}{\alpha} - \frac{\gamma v_0}{\alpha\beta} & \frac{1}{\alpha} & -\frac{\gamma}{\alpha\beta} & 0 \\ 0 & 1 & \frac{v_0}{\beta} & 0 & \frac{1}{\beta} & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{B.26}$$

$$= \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{1}{\alpha} & -\frac{\gamma}{\alpha\beta} & \frac{\gamma v_0}{\alpha\beta} - \frac{u_0}{\alpha} \\ 0 & 1 & 0 & 0 & \frac{1}{\beta} & -\frac{v_0}{\beta} \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{B.27}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} \frac{1}{\alpha} & -\frac{\gamma}{\alpha\beta} & \frac{\gamma v_0}{\alpha\beta} - \frac{u_0}{\alpha} \\ 0 & \frac{1}{\beta} & -\frac{v_0}{\beta} \\ 0 & 0 & 1 \end{bmatrix} \tag{B.28}$$

We thus have

$$\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1} \tag{B.29}$$

$$= \begin{bmatrix} \frac{1}{\alpha} & 0 & 0 \\ -\frac{\gamma}{\alpha\beta} & \frac{1}{\beta} & 0 \\ \frac{\gamma v_0}{\alpha\beta} - \frac{u_0}{\alpha} & -\frac{v_0}{\beta} & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\alpha} & -\frac{\gamma}{\alpha\beta} & \frac{\gamma v_0}{\alpha\beta} - \frac{u_0}{\alpha} \\ 0 & \frac{1}{\beta} & -\frac{v_0}{\beta} \\ 0 & 0 & 1 \end{bmatrix} \tag{B.30}$$

$$= \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma v_0 - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{\gamma v_0 - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix} \tag{B.31}$$

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma v_0 - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{\gamma v_0 - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix} \tag{B.32}$$

The matrix $\mathbf{B}$ is symmetric, and is therefore represented by Zhang as the vector $\mathbf{b}$ with six degrees of freedom

$$\mathbf{b} = \begin{bmatrix} B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33} \end{bmatrix}^T \tag{B.33}$$

A least-squares estimate of $\mathbf{B}$ can be obtained from the model plane to image homographies, using the constraints defined in Equations (B.21) and (B.22). Both constraints contain terms of the form $\mathbf{h}_i^T\mathbf{B}\mathbf{h}_j$, so with the k-th column vector of $\mathbf{H}$ defined as $\mathbf{h_k} = \begin{bmatrix} h_{k1} & h_{k2} & h_{k3} \end{bmatrix}^T$, the constraints may be rewritten in a form that uses the vectorized $\mathbf{b}$ as:

$$\mathbf{h_i^T B h_j} \tag{B.34}$$

$$= \begin{bmatrix} h_{i1} & h_{i2} & h_{i3} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \begin{bmatrix} h_{j1} \\ h_{j2} \\ h_{j3} \end{bmatrix} \tag{B.35}$$

$$\begin{aligned} &= B_{11}(h_{i1}h_{j1}) + B_{12}(h_{i1}h_{j2} + h_{i2}h_{j1}) + \\ &\quad B_{22}(h_{i2}h_{j2}) + B_{13}(h_{i1}h_{j3} + h_{i3}h_{j1}) + \\ &\quad\quad B_{23}(h_{i2}h_{j3} + h_{i3}h_{j2}) + B_{33}(h_{i3}h_{j3}) \end{aligned} \tag{B.36}$$

$$= \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i1}h_{j3} + h_{i3}h_{j1} \\ h_{i2}h_{j3} + h_{i3}h_{j2} \\ h_{i3}h_{j3} \end{bmatrix}^T \begin{bmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{bmatrix} \tag{B.37}$$

The left hand vector in Equation B.37 is labelled as $\mathbf{v_{ij}}$, and is then used to establish a homogeneous system for least squares estimation as:

$$\begin{bmatrix} \mathbf{v_{12}^T} \\ \mathbf{v_{11}^T - v_{22}^T} \end{bmatrix} \mathbf{b} = \mathbf{0} \tag{B.38}$$

Each homography to be used in the least squares estimation contributes an additional two rows to the system, in the same form as Equation B.38. Standard least squares techniques (such as the SVD) can be used to estimate the vector $\mathbf{b}$, and equivalently the matrix $\mathbf{B}$.

To solve for the individual camera parameters, the elements of $\mathbf{B}$ can be combined and then simplified. Zhang provides the final equations in Appendix B of [185], but they are recorded and simplified here to show extraction of the relevant parameters. It is important to note that the matrix $\mathbf{B}$ is defined up to a scaling factor λ, which must be retrieved in addition to the camera parameters.

Following the same sequence of parameter extraction and using the combinations

proposed by Zhang:

v_0:

$$\frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \tag{B.39}$$

$$= \frac{(\frac{-\lambda\gamma}{\alpha^2\beta})(\frac{\lambda\gamma v_0 - \lambda u_0\beta}{\alpha^2\beta}) - (\frac{\lambda}{\alpha^2})(\frac{-\lambda\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{\lambda v_0}{\beta^2})}{(\frac{\lambda}{\alpha^2})(\frac{\lambda\gamma^2}{\alpha^2\beta^2} + \frac{\lambda}{\beta^2}) - (\frac{\lambda\gamma}{\alpha^2\beta})^2} \tag{B.40}$$

$$= \frac{(\frac{-\lambda^2 v_0}{(\alpha^2\beta)^2}) + (\frac{\lambda^2\gamma u_0\beta}{(\alpha^2\beta)^2}) + (\frac{\lambda^2\gamma(v_0\gamma - u_0\beta)}{\alpha^4\beta^2}) + (\frac{\lambda^2 v_0}{\alpha^2\beta^2})}{(\frac{\lambda^2\gamma^2}{\alpha^4\beta^2}) + (\frac{\lambda^2}{\alpha^2\beta^2}) - (\frac{\lambda^2\gamma^2}{\alpha^4\beta^2})} \tag{B.41}$$

$$= \frac{-\gamma v_0}{\alpha^2\beta} + \frac{\gamma u_0\beta}{\alpha^2\beta} + \frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta} + v_0 \tag{B.42}$$

$$= v_0 \tag{B.43}$$

λ:

$$B_{33} - \frac{B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})}{B_{11}} \tag{B.44}$$

$$= \frac{\lambda(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{\lambda v_0^2}{\beta^2} + \lambda - \frac{(\frac{\lambda(v_0\gamma - u_0\beta)}{\alpha^2\beta})^2 + v_0((\frac{-\lambda\gamma}{\alpha^2\beta})(\frac{\lambda(v_0\gamma - u_0\beta)}{\alpha^2\beta}) - (\frac{\lambda}{\alpha^2})(\frac{-\lambda\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{\lambda v_0}{\beta^2}))}{\frac{\lambda}{\alpha^2}} \tag{B.45}$$

$$= \frac{\lambda(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{\lambda v_0^2}{\beta^2} + \lambda - \frac{\lambda^2(v_0\gamma - u_0\beta)^2}{\alpha^4\beta^2}\frac{\alpha^2}{\lambda} + \frac{\lambda^2 v_0\gamma(v_0\gamma - u_0\beta)}{\alpha^4\beta^2}\frac{\alpha^2}{\lambda} - \frac{\lambda^2 v_0\gamma(v_0\gamma - u_0\beta)}{\alpha^4\beta^2}\frac{\alpha^2}{\lambda} - \frac{\lambda^2 v_0^2}{\alpha^2\beta^2}\frac{\alpha^2}{\lambda} \tag{B.46}$$

$$= \frac{\lambda(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{\lambda v_0^2}{\beta^2} + \lambda - \frac{\lambda(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{\lambda v_0\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{\lambda v_0\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{\lambda v_0^2}{\beta^2} \tag{B.47}$$

$$= \lambda \tag{B.48}$$

α:

$$B11 = \frac{\lambda}{\alpha^2} \tag{B.49}$$

$$\alpha = \sqrt{\frac{\lambda}{B_{11}}} \tag{B.50}$$

β:

$$\sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \tag{B.51}$$

$$= \sqrt{\frac{\lambda \frac{\lambda}{\alpha^2}}{\frac{\lambda}{\alpha^2}(\frac{\lambda\gamma^2}{\alpha^2\beta^2} + \frac{\lambda}{\beta^2}) - (\frac{-\lambda\gamma}{\alpha^2\beta})^2}} \tag{B.52}$$

$$= \sqrt{\frac{\frac{\lambda^2}{\alpha^2}}{\frac{\lambda^2\gamma^2}{\alpha^4\beta^2} + \frac{\lambda^2}{\alpha^2\beta^2} - \frac{\lambda^2\gamma^2}{\alpha^4\beta^2}}} = \sqrt{\frac{\frac{\lambda^2}{\alpha^2}}{\frac{\lambda^2}{\alpha^2\beta^2}}} \tag{B.53}$$

$$= \sqrt{\beta^2} = \beta \tag{B.54}$$

γ:

$$\frac{-B_{12}\alpha^2\beta}{\lambda} \tag{B.55}$$

$$= \frac{\frac{\lambda\gamma}{\alpha^2\beta}\alpha^2\beta}{\lambda} = \frac{\lambda\gamma}{\lambda} = \gamma \tag{B.56}$$

u_0:

$$\frac{\gamma v_0}{\beta} - \frac{B_{13}\alpha^2}{\lambda} \tag{B.57}$$

$$= \frac{\gamma v_0}{\beta} - \frac{\lambda(\gamma v_0 - u_0\beta)}{\alpha^2\beta}\frac{\alpha^2}{\lambda} \tag{B.58}$$

$$= \frac{\gamma v_0}{\beta} - \frac{\gamma v_0}{\beta} + \frac{u_0\beta}{\beta} = u_0 \tag{B.59}$$

Once the camera parameters, and thus the camera matrix $\mathbf{A}$ have been recovered, the camera extrinsic parameters can be determined using the following equations:

$$\mathbf{r_1} = \lambda \mathbf{A}^{-1} \mathbf{h_1} \tag{B.60}$$

$$\mathbf{r_2} = \lambda \mathbf{A}^{-1} \mathbf{h_2} \tag{B.61}$$

$$\mathbf{r_3} = \mathbf{r_1} \times \mathbf{r_2} \tag{B.62}$$

$$\mathbf{t} = \lambda \mathbf{A}^{-1} \mathbf{h_3} \tag{B.63}$$

The estimated columns $\mathbf{r_i}$ do not necessarily exhibit the properties of a rotation matrix, so a valid rotation matrix close to the recovered parameters should be estimated [185].

B.3 Homography-based Filtering

For the work described in this thesis, residual error from model to image homography calculation is used to filter image frames that have obtained poor grid feature localization. Such a condition occurs when the complete image is blurred close to the limit of the scale-space scale level threshold. In that case, blur may reduce localization information uniformly across the image, which can be detected through larger than normal residuals in the homography refinement optimization. Such frames are rejected from the calibration data set.

B.4 Non-linear Optimization

Following camera parameter initialization from the least squares solution of the vector $\mathbf{b}$, a non-linear optimization is required to refine the parameters and to recover lens distortion estimates. The least squares solution minimizes algebraic error, which has no geometric meaning in the context of projective imaging. Instead, a geometrically meaningful quantity such as image plane reprojection error should be minimized. Such techniques are described, for example, by Hartley [60].

The non-linear optimization should refine the camera intrinsic, extrinsic, and distortion parameter estimates. In some formulations, the minimization can further

adjust the estimated 3D coordinates of the model plane image points (bundle adjustment), and also scaling and aspect ratio estimates of the gridded target [152].

Radial distortion, modelled using two distortion coefficients, can be approximated as a Taylor polynomial:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = \begin{bmatrix} u_u \\ v_u \end{bmatrix} + [k_1((u_u - u_c)^2 + (v_u - v_c)^2) + k_2((u_u - u_c)^2 + (v_u - v_c)^2)^2] \begin{bmatrix} u_u - u_c \\ v_u - v_c \end{bmatrix} \tag{B.64}$$

$$= \begin{bmatrix} u_u \\ v_u \end{bmatrix} + [k_1((u_u - (u_c + r_u))^2 + (v_u - (v_c + r_v))^2) + \tag{B.65}$$

$$k_2((u_u - (u_c + r_u))^2 + (v_u - (v_c + r_v))^2)^2] \begin{bmatrix} u_u - (u_c + r_u) \\ v_u - (v_c + r_v) \end{bmatrix} \tag{B.66}$$

where k_1 and k_2 are the radial distortion coefficients, u_u and v_u are the undistorted image coordinates, u_d and v_d are the distorted image coordinates, and u_c and v_c are image coordinates of the centre of radial distortion. Distortion according to Equation B.66 can be used within the optimization objective function to refine the distortion coefficients k_1 and k_2 and distortion centre estimates.

Levenberg-Marquardt (LM) optimization is often selected for non-linear refinement in camera calibration and bundle adjustment [165, 60]. In the context of calibration or bundle adjustment, a sparse LM implementation should typically be chosen because of significant computational speedup. The optimization variable vector contains the independent extrinsic parameters of the camera at multiple viewpoints used for the calibration. Changes to the extrinsic parameter estimate for one view position does not affect the reconstruction error in other independent camera viewpoints, so the optimization normal equations have a sparse structure. Sparse LM optimization exhibits significant acceleration in terms of computation load, compared with naive (dense) application of the algorithm. Further details on sparse LM optimization, in the context of bundle adjustment, are available in [97, 60].

Table B.1: Calibration parameter scaling parameters

Parameter	Units	Approximate value from testing	Scaling Factor
α	Scaling factor	2.8538e+03	1E4
β	Scaling factor	2.8558e+03	1E4
u_0	Pixels	512.2039	1E3
v_0	Pixels	384.1091	1E3
k_1	First order multiplier	-6.6750e-06	1E10
k_2	Second order multiplier	-1.2285e-08	1E12
r_I	Pixels	-5.9379	1E3
r_J	Pixels	-7.7186	1E3
ν	Ratio	1.0043	1E6
Rotation	$u\theta$	0.1-10	1E4
Translation	Calibration grid units	1-100	1E3

B.5 Calibration Parameter Scaling

In the Zhang-based optimization proposed by Strobl and Hirzinger [152], termination of the LM optimization occurs when the iterative parameter changes become less than a set magnitude between iterations. The optimization variables are recorded in differing units with varying final value magnitudes, and requiring a variety of tolerances. Scaling is used to adjust the parameter magnitudes before input to optimization, and inverse scaling is used within the objective function before applying the variables in computation. The LM objective function termination criterion can be set within one or two orders of the magnitude of machine precision to cause optimization to terminate on parameter tolerance, rather than on objective value changes which have little meaning. Table B.1 shows a set of scaling factors based on the magnitude of typical values from calibration of the camera used in this work.

References

[1] Abdel-Aziz, Y. and Karara, H. (1971). Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proceedings of the Symposium on Close-Range Photogrammetry*, pages 1–18, Falls Church, VA. American Society of Photogrammetry.

[2] Aguilar, J. J., Torres, F., and Lope, M. A. (1996). Stereo vision for 3d measurement: accuracy analysis, calibration and industrial applications. *Measurement*, **18**(4), 193 – 200.

[3] Ahn, S. J., Rauh, W., and Warnecke, H.-J. (2001). Least-squares orthogonal distances fitting of circle, sphere, ellipse, hyperbola, and parabola. *Pattern Recognition*, **34**(12), 2283 – 2303.

[4] Albarelli, A., Rodolà, E., and Torsello, A. (2010). Robust camera calibration using inaccurate targets. In *Proceedings of the British Machine Vision Conference*, pages 16.1–16.10. BMVA Press.

[5] Babaud, J., Witkin, A. P., Baudin, M., and Duda, R. O. (1986). Uniqueness of the gaussian kernel for scale-space filtering. *IEEE Trans. Pattern Anal. Mach. Intell.*, **8**(1), 26–33.

[6] Badekas, E. and Papamarkos, N. (2005). Automatic evaluation of document binarization results. In A. Sanfeliu and M. Corts, editors, *Progress in Pattern Recognition, Image Analysis and Applications*, volume 3773 of *Lecture Notes in Computer Science*, pages 1005–1014. Springer Berlin / Heidelberg.

[7] Batista, J., Araujo, H., and De Almeida, A. (1998). Iterative multi-step explicit camera calibration. In *Computer Vision, 1998. Sixth International Conference on*, pages 709 –714.

[8] Beardsley, P. and Murray, D. (1992). Camera calibration using vanishing points. In *BMVC92*.

[9] Bougnoux, S. (1998). From projective to Euclidean space under any practical situation, a criticism of self-calibration. In *Computer Vision, 1998. Sixth International Conference on*, pages 790 –796.

[10] Bretzner, L. and Lindeberg, T. (1996). Feature tracking with automatic selection of spatial scales. *Computer Vision and Image Understanding*, **71**, 385–392.

[11] Brown, D. C. (1971). Close-range camera calibration. *PHOTOGRAMMETRIC ENGINEERING*, **37**(8), 855–866.

[12] Brown, M. Z., Burschka, D., and Hager, G. D. (2003). Advances in computational stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, **25**, 993–1008.

[13] Bruzzone, E. and Mangili, F. (1991). Calibration of a ccd camera on a hybrid coordinate measuring machine for industrial metrology. *Industrial Vision Metrology*, **1526**(1), 96–112.

[14] Canny, J. (1986). A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **PAMI-8**(6), 679 –698.

[15] Carter, J., Tucker, T., and Kurfess, T. (2008). 3-axis CNC path planning using depth buffer and fragment shader. *Computer Aided Design and Applications*, **5**(1-4), TBD.

[16] Chan, H.-L. (2005). *Laser Digitizer-Based Sheet Metal Strain and Surface Analysis*. Ph.D. thesis, McMaster University.

[17] Chan, H.-L., Spence, A., and Sklad, M. (2005). Parallel computing for sheet metal strain analysis. In *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*, pages 260 – 266.

[18] Chan, H.-L., Spence, A., and Sklad, M. (2007). Laser digitizer-based sheet metal strain and surface analysis. *International Journal of Machine Tools and Manufacture*, **47**(1), 191 – 203.

[19] Chen, D. and Zhang, G. (2005). A new sub-pixel detector for x-corners in camera calibration targets. In *WSCG (Short Papers)*, pages 97–100.

[20] Chinveeraphan, S., Takamatsu, R., and Sato, M. (1995). Understanding of ridge-valley lines on image-intensity surfaces in scale-space. In V. Hlavc and R. ra, editors, *Computer Analysis of Images and Patterns*, volume 970 of *Lecture Notes in Computer Science*, pages 661–667. Springer Berlin / Heidelberg.

[21] Colic, A., Kalva, H., and Furht, B. (2010). Exploring nvidia-cuda for video coding. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, MMSys '10, pages 13–22, New York, NY, USA. ACM.

[22] Cope, B., Cheung, P. Y., Luk, W., and Howes, L. (2010). Performance comparison of graphics processors to reconfigurable logic: A case study. *IEEE Transactions on Computers*, **59**, 433–448.

[23] Coutinho, B., Teodoro, G., Oliveira, R., Neto, D., and Ferreira, R. (2009). Profiling general purpose gpu applications. In *Computer Architecture and High Performance Computing, 2009. SBAC-PAD '09. 21st International Symposium on*, pages 11 –18.

[24] Daniilidis, K. (1999). Hand-eye calibration using dual quaternions. *International Journal of Robotic Research*, **18**, 286–298.

[25] Davies, E. R. (2004). *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[26] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition.

[27] de la Escalera, A. and Armingol, J. M. (2010). Automatic chessboard detection for intrinsic and extrinsic camera parameter calibration. *Sensors*, **10**(3), 2027–2044.

[28] Deriche, R. and Giraudon, G. (1991). On corner and vertex detection. In *CVPR91*, pages 650–655.

[29] Deriche, R. and Giraudon, G. (1992). A computational approach for corner and vertex detection. *International Journal of Computer Vision*, **10**, 101–124.

[30] Devernay, F., Devernay, F., Robotique, P., and Robotvis, P. (1995). A non-maxima suppression method for edge detection with sub-pixel accuracy. Technical report, INRIA Research Rep. 2724, SophiaAntipolis.

[31] di Baja, G. and Nystrom, I. (2004). 2d grey-level skeleton computation: a discrete 3d approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 455 – 458 Vol.2.

[32] Dornaika, F. and Horaud, R. (1998). Simultaneous robot-world and hand-eye calibration. *Robotics and Automation, IEEE Transactions on*, **14**(4), 617 –622.

[33] Douskos, V., Kalisperakis, I., and Karras, G. (2007). Automatic calibration of digital cameras using planar chess-board patterns. In *Optical 3-D Measurement Techniques VIII*, volume 1, pages 132–140. Wichman Verlag.

[34] Douxchamps, D. and Chihara, K. (2009). High-accuracy and robust localization of large control markers for geometric camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **31**(2), 376 –383.

[35] Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, **15**, 11–15.

[36] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition.

[37] Eberly, D., Gardner, R., Morse, B., Pizer, S., and Scharlach, C. (1994). Ridges for image analysis. *Journal of Mathematical Imaging and Vision*, **4**(4), 353–373.

[38] Faugeras, O. and Toscani, G. (1986). The calibration problem for stereo. In *CVPR86*, pages 15–20.

[39] Faugeras, O., Luong, Q.-T., and Papadopoulou, T. (2001). *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA.

[40] Fiala, M. and Shu, C. (2005). Fully automatic camera calibration using self-identifying calibration targets. Technical Report NRC/ERB-1130, NRC 48306, National Research Council of Canada.

[41] Fleck, M. (1992). Some defects in finite-difference edge finders. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**, 337–345.

[42] Florack, L. (2000). A spatio-frequency trade-off scale for scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(9), 1050–1055.

[43] Florack, L. and Kuijper, A. (2000). The topological structure of scale-space images. *Journal of Mathematical Imaging and Vision*, **12**, 65–79.

[44] Florack, L., Romeny, H., Koenderink, J., and Viergever, M. (1994). General intensity transformations and differential invariants. *Journal of Mathematics Imaging and Vision*, **4**, 171–187.

[45] Florack, L. M., ter Haar Romeny, B. M., Koenderink, J. J., and Viergever, M. A. (1992). Scale and the differential structure of images. *Image and Vision Computing*, **10**(6), 376 – 388. Information Processing in Medical Imaging.

[46] Forbes, K., Voigt, A., and Bodika, N. (2002). An inexpensive, automatic and accurate camera calibration method. In *In Proceedings of the Thirteenth Annual South African Workshop on Pattern Recognition. PRASA*.

[47] Fung, J. and Mann, S. (2004). Computer vision signal processing on graphics processing units. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, volume 5, pages V – 93–6 vol.5.

[48] Fung, J. and Mann, S. (2008). Using graphics devices in reverse: Gpu-based image processing and computer vision. In *International Conference on Multimedia*

Computing and Systems/International Conference on Multimedia and Expo, pages 9–12.

[49] Galanulis, K. (2005). Optical measuring technologies in sheet metal processing. *Advanced Materials Research*, **6-8**, 19–34.

[50] Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., and Rossi, F. (2004). *GSL-GNU Scientific Library: Reference manual*. Network Theory Ltd., third edition.

[51] Gherardi, R. and Fusiello, A. (2010). Practical autocalibration. In *Proceedings of the 11th European conference on Computer vision: Part I*, ECCV'10, pages 790–801, Berlin, Heidelberg. Springer-Verlag.

[52] Goldstein, M. S., Mitchell, J. P., Fleisig, R. V., and Spence, A. D. (July 18-20, 2005). Design of a close-up stereo vision based sheet metal inspection system. In *2nd CDEN International Conference on Design Education, Innovation, and Practice*, Kananaskis, AB.

[53] Gonzlez, J. I., Gmez, J. C., Artal, C. G., and Cabrera, A. M. N. (2005). Stability study of camera calibration methods. In *CI Workshop en Agentes F fisicos Spain WAF*.

[54] Gorthi, S. S. and Rastogi, P. (2010). Fringe projection techniques: Whither we are? *OPTICS AND LASERS IN ENGINEERING*, **48**(2), 133–140.

[55] Guo, F., Yang, Y., Chen, B., and Guo, L. (2010). A novel multi-scale edge detection technique based on wavelet analysis with application in multiphase flows. *Powder Technology*, **202**(1-3), 171 – 177.

[56] Hall, E., Tio, J., McPherson, C., and Sadjadi, F. (1982). Measuring curved surfaces for robot vision. *Computer*, **15**(12), 42 – 54.

[57] Hanson, A. J. (2006). *Visualizing quaternions*. Morgan Kaufmann.

[58] Haralick, R. M. (1983). Ridges and valleys on digital images. *Computer Vision, Graphics, and Image Processing*, **22**(1), 28 – 38.

[59] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–152.

[60] Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK.

[61] Hartley, R. I. (1994). Projective reconstruction and invariants from multiple images. *IEEE Trans. Pattern Anal. Mach. Intell.*, **16**, 1036–1041.

[62] Heikkila, J. and Silven, O. (1997). A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1106 –1112.

[63] Hennessy, J. L. and Patterson, D. A. (2006). *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann.

[64] Hoffmann, H. and Vogl, C. (2003). Determination of true stress-strain-curves and normal anisotropy in tensile tests with optical strain measurement. *CIRP Annals - Manufacturing Technology*, **52**(1), 217 – 220.

[65] Hopf, M. and Ertl, T. (1999). Hardware based wavelet transformations. In *Vision, Modeling, and Visualization '99*, pages 317–328.

[66] Horaud, R. and Dornaika, F. (1995). Hand-eye calibration. *Int. J. Rob. Res.*, **14**, 195–210.

[67] Horn, B. K. P. (2000). Tsai's camera calibration method revisited. Technical report, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[68] Hwu, W.-M., Rodrigues, C., Ryoo, S., and Stratton, J. (2009). Compute unified device architecture application suitability. *Computing in Science Engineering*, **11**(3), 16 –26.

[69] Inc., F. T. (2010). *FARO PowerGage Technical Sheet*.

[70] Johansson, G. and Carr, H. (2006). Accelerating marching cubes with graphics hardware. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, CASCON '06, New York, NY, USA. ACM.

[71] Kinsner, M., Capson, D., and Spencer, A. (2007). Scale-space feature detection for close range camera calibration. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 1464 –1467.

[72] Kinsner, M., Capson, D., and Spence, A. (2008). Scale-space ridge detection with gpu acceleration. In *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, pages 001527 –001530.

[73] Kinsner, M., Spence, A., and Capson, D. (2010a). GPU Accelerated Sheet Forming Grid Measurement. *Computer Aided Design and Applications*, **7**(5), 675–684.

[74] Kinsner, M., Capson, D., and Spence, A. (2010b). A modular cuda-based framework for scale-space feature detection in video streams. *Journal of Physics: Conference Series*, **256**(1), 012005.

[75] Kirk, D. B. and Hwu, W.-m. W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[76] Koenderink, J. (1984). The structure of images. *Biological Cybernetics*, **50**(5), 363–370.

[77] Kovacs, P. and Tisza, M. (2008). A complex measuring and evaluation system for determination of forming limit diagrams. *Materials Science Forum*, **589**, 233–238.

[78] Kruger, L. and Wohler, C. (2011). Accurate chequerboard corner localisation for camera calibration. *Pattern Recognition Letters*, **In Press, Accepted Manuscript**.

[79] Kuchnio, P. and Capson, D. (2009). A parallel mapping of optical flow to compute unified device architecture for motion-based image segmentation. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 2325 –2328.

[80] Kuijper, A. and Florack, L. (1999). Calculations on critical points under gaussian blurring. In *Scale-Space Theories in Computer Vision*, pages 318–329.

[81] Kuipers, J. B. (1999). *Quaternions and Rotation Sequences*. Princeton University Press, Princeton.

[82] Kurfess, T., T. Tucker, K. A., and Meghashyam, P. M. (2007). GPU for CAD. *Computer Aided Design and Applications*, **4**(6), 853–862.

[83] Lam, L., Lee, S.-W., and Suen, C. (1992). Thinning methodologies-a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **14**(9), 869 –885.

[84] Lavest, J.-M., Viala, M., and Michel, D. (1998). Do we really need an accurate calibration pattern to achieve a reliable camera calibration? In *Proceedings of the 5th European Conference on Computer Vision-Volume I - Volume I*, ECCV '98, pages 158–174, London, UK. Springer-Verlag.

[85] Lee, D., Dinov, I., Dong, B., Gutman, B., Yanovsky, I., and Toga, A. W. (2010a). Cuda optimization strategies for compute- and memory-bound neuroimaging algorithms. *Computer Methods and Programs in Biomedicine*, **In Press, Corrected Proof**.

[86] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., and Dubey, P. (2010b). Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. *SIGARCH Comput. Archit. News*, **38**(3), 451–460.

[87] Lengyel, J., Reichert, M., Donald, B. R., and Greenberg, D. P. (1990). Real-time robot motion planning using rasterizing computer graphics hardware. In *In Proc. SIGGRAPH*, pages 327–335.

[88] Lim, J.-Y. and Stiehl, H. (2003). A generalized discrete scale-space formulation for 2-d and 3-d signals. In *Scale Space Methods in Computer Vision*, pages 132–147. Springer Berlin / Heidelberg.

[89] Lindeberg, T. (1990). Scale-space for discrete signals. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, **12**(3), 234–254.

[90] Lindeberg, T. (1993). Discrete derivative approximations with scale-space properties: A basis for low-level feature extraction. *Journal of Mathematical Imaging and Vision*, **3**(4), 349–376.

[91] Lindeberg, T. (1994a). Junction detection with automatic selection of detection scales and localization scales. In *In Proc. 1st International Conference on Image Processing, volume I*, pages 924–928. IEEE Computer Society Press.

[92] Lindeberg, T. (1994b). *Scale-space Theory in Computer Vision*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[93] Lindeberg, T. (1996). Edge detection and ridge detection with automatic scale selection. In *Proceedings of Computer Vision and Pattern Recognition*, pages 465–470.

[94] Lindeberg, T. (1998). Feature detection with automatic scale selection. *International Journal of Computer Vision*, **30**, 79–116.

[95] Lindeberg, T. (1999). Principles for automatic scale selection. In *HCVA99*, pages II: 239–274.

[96] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA. ACM.

[97] Lourakis, M. A. and Argyros, A. (2009a). SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, **36**(1), 1–30.

[98] Lourakis, M. I. A. and Argyros, A. A. (2009b). Sba: A software package for generic sparse bundle adjustment. *ACM Trans. Math. Softw.*, **36**, 2:1–2:30.

[99] Lucchese, L. and Mitra, S. (2002). Using saddle points for subpixel feature detection in camera calibration targets. In *Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific Conference on*, volume 2, pages 191 – 195 vol.2.

[100] Luhmann, T., Robson, S., Kyle, S., and Harley, I. (2007). *Close Range Photogrammetry: Principles, Techniques and Applications*. Wiley.

[101] Luo, Y. and Duraiswami, R. (2008). Canny edge detection on nvidia cuda. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1 –8.

[102] MacLean, W. J. and Tsotsos, J. K. (2008). Fast pattern recognition using normalized grey-scale correlation in a pyramid image representation. *Mach. Vision Appl.*, **19**, 163–179.

[103] Mallat, S. (2008). *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition.

[104] Mallon, J. and Whelan, P. F. (2007). Which pattern? biasing aspects of planar calibration patterns and detection methods. *Pattern Recogn. Lett.*, **28**, 921–930.

[105] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London Series B*, **207**, 187–217.

[106] Mitchell, J. (2005). *Close-Up Stereo Triangulation with Application to Sheet Metal Strain Analysis*. Master's thesis, McMaster University.

[107] Mokhtarian, F. and Suomela, R. (1998). Robust image corner detection through curvature scale space. *IEEE Trans. Pattern Anal. Mach. Intell.*, **20**, 1376–1381.

[108] Moravec, H. P. (1980). *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Ph.D. thesis, Stanford University, Stanford, CA, USA. AAI8024717.

[109] Newman, T. S. and Yi, H. (2006). A survey of the marching cubes algorithm. *Computers & Graphics*, **30**(5), 854 – 879.

[110] NVIDIA (2009a). NVIDIA CUDA C Programming Best Practices Guide. `http://developer.nvidia.com/object/gpucomputing.html`.

[111] NVIDIA (2009b). NVIDIA's Next Generation CUDA Compute Architecture: Fermi. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.

[112] NVIDIA (2010a). NVIDIA CUDA C Programming Guide. http://developer.nvidia.com/object/gpucomputing.html.

[113] NVIDIA (2010b). NVIDIA GF100 White paper. http://www.nvidia.com/object/IO_89569.html.

[114] NVIDIA (2010c). PTX: Parallel Thread Execution ISA Version 2.1. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/ptx_isa_2.1.pdf.

[115] NVIDIA (2010d). Tuning CUDA Applications for Fermi. http://developer.nvidia.com/object/gpucomputing.html.

[116] O'Rourke, J. (1998). *Computational Geometry in C*. Cambridge University Press. Hardback ISBN: 0521640105; Paperback: ISBN 0521649765.

[117] Otsu, N. (1979). A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on*, **9**(1), 62 –66.

[118] Owens, John, D., Luebke, David, Govindaraju, Naga, Harris, Mark, Kruger, Jens, Lefohn, Aaron, E., Purcell, and Timothy, J. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, **26**(1), 80–113.

[119] Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., and Phillips, J. (2008). Gpu computing. *Proceedings of the IEEE*, **96**(5), 879 –899.

[120] Parida, L., Geiger, D., and Hummel, R. (1998). Junctions: detection, classification, and reconstruction. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **20**(7), 687 –698.

[121] Park, S. I., Ponce, S., Huang, J., Cao, Y., and Quek, F. (2008). Low-cost, high-speed computer vision using nvidia's cuda architecture. In *Applied Imagery Pattern Recognition Workshop, 2008. AIPR '08. 37th IEEE*, pages 1 –7.

[122] Pauwels, K. and Van Hulle, M. (2008). Realtime phase-based optical flow on the gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1 –8.

[123] Peitgen, H.-O., Jürgens, H., and Saupe, D. (2004). *Chaos and fractals - new frontiers of science*. Springer, 2nd edition.

[124] Peng, T. and Gupta, S. K. (2007). Model and algorithms for point cloud construction using digital projection patterns. *Journal of Computing and Information Science in Engineering*, **7**(4), 372–381.

[125] Perwass, C. (2005). Junction and corner detection through the extraction and analysis of line segments. In R. Klette and J. unic, editors, *Combinatorial Image Analysis*, volume 3322 of *Lecture Notes in Computer Science*, pages 568–582. Springer Berlin / Heidelberg.

[126] ping Wang, Y. and Lee, S. L. (1998). Scale-space derived from b-splines. *IEEE Trans. Pattern Anal. Machine Intell*, **20**, 1040–1055.

[127] Pirjan, A. (2010). Improving software performance in the compute unified device architecture. *Informatica Economica*, **14**(4), 30 – 47.

[128] Pock, T., Unger, M., Cremers, D., and Bischof, H. (2008). Fast and exact solution of total variation models on the gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1 –8.

[129] POV-Ray (2011). *Persistance of Vision Raytracer*.

[130] Ray, N. and Saha, B. (2007). Edge sensitive variational image thresholding. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 6.

[131] Remaki, L. and Cheriet, M. (2000). Kcs-new kernel family with compact support in scale space: formulation and impact. *IEEE Transactions on Image Processing*, **9**, 970–981.

[132] Research, P. G. (2004). Dragonfly: Technical reference manual. Technical report, Point Grey Research.

[133] Ricolfe-Viala, C. and Sanchez-Salmeron, A.-J. (2007). Improved camera calibration method based on a two-dimensional template. In *Proceedings of the 3rd Iberian conference on Pattern Recognition and Image Analysis, Part II*, IbPRIA '07, pages 420–427, Berlin, Heidelberg. Springer-Verlag.

[134] Romeny, B. M. (1996). Introduction to Scale-Space Theory: Multiscale Geometric Image Analysis. Technical report, Utrecht University.

[135] Salvi, J., Armangu, X., and Batlle, J. (2002). A comparative review of camera calibrating methods with accuracy evaluation. *Pattern Recognition*, **35**(7), 1617 – 1635.

[136] Sauvola, J. and Pietikinen, M. (2000). Adaptive document image binarization. *Pattern Recognition*, **33**(2), 225 – 236.

[137] Schneider, M., Friebe, H., and Galanulis, K. (2008). Validation and optimization of numerical simulations by optical measurements of tools and parts. In *International Deep Drawing Research Group, IDDRG 2008 International Conference*, Olofstrom, Sweden.

[138] Sebok, T. J., Roemer, L. E., and Jr., G. S. M. (1981). An algorithm for line intersection identification. *Pattern Recognition*, **13**(2), 159 – 166.

[139] Serrat, J., Lopez, A., and Lloret, D. (2000). On ridges and valleys. *Pattern Recognition, International Conference on*, **4**, 4059.

[140] Sezgin, M. and Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, **13**(1), 146–168.

[141] Shafait, F., Keysers, D., and Breuel, T. (2008). Efficient implementation of local adaptive thresholding techniques using integral images. In *Proceedings of the 15th Document Recognition and Retrieval Conference (DRR-2008), Part of the IS&TSPIE International Symposium on Electronic Imaging, January 26-31, San Jose, CA, USA*. SPIE.

[142] Shen, T.-S. and Menq, C.-H. (2001). Automatic camera calibration for a multiple-sensor integrated coordinate measurement system. *Robotics and Automation, IEEE Transactions on*, **17**(4), 502 –507.

[143] Shu, C., Brunton, A., and Fiala, M. (2003). Automatic grid finding in calibration patterns using delaunay triangulation. Technical report, NRC Institute for Information Technology. 46497.

[144] Shu, C., Brunton, A., and Fiala, M. (2010). A topological approach to finding grids in calibration patterns. *Machine Vision and Applications*, **21**, 949–957.

[145] Sklad, M. P. (2004). Aspects of automated measurement of proportional and non-proportional deformation in sheet metal forming. *Journal of Materials Processing Technology*, **145**(3), 377 – 384.

[146] Sowerby, R., Duncan, J., and Chu, E. (1986). The modelling of sheet metal stampings. *Int. J. Mech. Sci.*, **28**(7), 415–430.

[147] Spence, A. D., Capson, D. W., Sklad, M. P., Chan, H.-L., and Mitchell, J. P. (2008). Simultaneous large scale sheet metal geometry and strain measurement. *Journal of Manufacturing Science and Engineering*, **130**(5).

[148] Steger, C. (1998). Evaluation of subpixel line and edge detection precision and accuracy. In *In International Archives of Photogrammetry and Remote Sensing*, pages 256–264.

[149] Steger, C. (2000). Subpixel-precise extraction of lines and edges. In *International Archives of Photogrammetry and Remote Sensing*, volume XXXIII, part B3, pages 141–156.

[150] Steinmetz, C. (1990). Sub-micron position measurement and control on precision machine tools with laser interferometry. *Precision Engineering*, **12**(1), 12 – 24.

[151] Strobl, K. H. and Hirzinger, G. (2006). Optimal Hand-Eye Calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4647–4653, Beijing, China.

[152] Strobl, K. H. and Hirzinger, G. (2008). More Accurate Camera and Hand-Eye Calibrations with Unknown Grid Pattern Dimensions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398–1405, Pasadena, CA, USA.

[153] Sturm, P. and Maybank, S. (1999). On plane-based camera calibration: A general algorithm, singularities, applications. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages 2 vol. (xxiii+637+663).

[154] Sturm, P., Ramalingam, S., Tardif, J.-P., Gasparini, S., and Barreto, J. (2011). Camera models and fundamental concepts used in geometric computer vision. In *Foundations and Trends in Computer Graphics and Vision*, volume 6, pages 1–183. Now Publishers Inc., Hanover, MA, USA.

[155] Sun, W. and Cooperstock, R. (2006). An empirical evaluation of factors influencing camera calibration accuracy using three publicly available techniques. *Mach. Vision Appl.*, **17**, 51–67.

[156] Sutton, P. M., Hansen, C. D., wei Shen, H., and Schikore, D. (2000). A case study of isosurface extraction algorithm performance. In *Data Visualization 2000*, pages 259–268. Springer.

[157] Suzuki, K., Horiba, I., and Sugie, N. (2003). Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding*, **89**, 1–23.

[158] Swapna, P., Krouglicof, N., and Gosine, R. (2009). The question of accuracy with geometric camera calibration. In *Electrical and Computer Engineering, 2009. CCECE '09. Canadian Conference on*, pages 541 –546.

[159] Snchez, J., Destefanis, E., and Canali, L. (2006). Plane-based camera calibration without direct optimization algorithms. In *IV Jornadas Argentinas de Robtica*, Cordoba.

[160] Teixeira, L., Celes, W., and Gattass, M. (2008). Accelerated corner-detector algorithms. In *BMVC08*.

[161] Thirion, J.-P. and Gourdon, A. (1993). The Marching lines algorithm : new results and proofs. Research Report RR-1881, INRIA.

[162] Thormahlen, T., Broszio, H., and Mikulastik, P. (2006). Robust linear auto-calibration of a moving camera from image sequences. In *ACCV06*, pages II:71–80.

[163] Treibig, J., Hager, G., Hofmann, H. G., Hornegger, J., and Wellein, G. (2011). Pushing the limits for medical image reconstruction on recent standard multicore processors. *CoRR*, **abs/1104.5243**.

[164] Triggs, B. (1998). Autocalibration from planar scenes. In *Proceedings of the 5th European Conference on Computer Vision-Volume I - Volume I*, ECCV '98, pages 89–105, London, UK. Springer-Verlag.

[165] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment a modern synthesis. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 153–177. Springer Berlin / Heidelberg.

[166] Tsai, R. (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, **3**, 323–344.

[167] Tuzikov, A., Soille, P., Jeulin, D., Bruneel, H., and Vermeulen, M. (1992). Extraction of grid patterns on stamped metal sheets using mathematical morphology. In *Pattern Recognition, 1992. Vol.I. Conference A: Computer Vision and Applications, Proceedings., 11th IAPR International Conference on*, pages 425 –428.

[168] Vacher, P., Haddad, A., and Arrieux, R. (1999). Determination of the forming limit diagrams using image analysis by the corelation method. *CIRP Annals - Manufacturing Technology*, **48**(1), 227 – 230.

[169] Viola, P. and Jones, M. (2004). Robust real-time face detection. *International Journal of Computer Vision*, **57**, 137–154.

[170] Volkov, V. and Demmel, J. W. (2008). Benchmarking gpus to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 31:1–31:11, Piscataway, NJ, USA. IEEE Press.

[171] Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M., and Shringarpure, A. (2010). On the limits of gpu acceleration. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, HotPar'10, pages 13–13, Berkeley, CA, USA. USENIX Association.

[172] Weitkamp, C., editor (2005). *Lidar Range-Resolved Optical Remote Sensing of the Atmosphere*. Springer Series in Optical Sciences. Springer Berlin/Heidelberg.

[173] Weng, J., Cohen, P., and Herniou, M. (1992). Camera calibration with distortion models and accuracy evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, **14**, 965–980.

[174] Willson, R. (1994). *Modeling and Calibration of Automated Zoom Lenses*. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[175] Willson, R. and Shafer, S. (1991). Active lens control for high precision computer imaging. In *Proc. IEEE International Conference on Robotics and Automation (ICRA '91)*, volume 3, pages 2063 – 2070.

[176] Witkin, A. P. (1983). Scale-space filtering. In *Proceedings of the 8th International Joint Conference on Artificial Interlligence*, pages 1019–1022.

[177] Wong, H., Papadopoulou, M.-M., Sadooghi-Alvandi, M., and Moshovos, A. (2010). Demystifying gpu microarchitecture through microbenchmarking. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 235 –246.

[178] Xia, J., long Xiong, J., Xu, X., and Qin, H. (2010). A multiscale sub-pixel detector for corners in camera calibration targets. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, volume 1, pages 196 –199.

[179] Yang, Q. and Ma, S. D. (1997). Optical flow in the scale space. In *Computer Vision, ACCV'98*, pages 607–614. Springer Berlin / Heidelberg.

[180] Yu, Q. and Medioni, G. (2008). A gpu-based implementation of motion detection from a moving platform. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1 –6.

[181] Yuille, A. L. and Poggio, T. A. (1986). Scaling theorems for zero crossings. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **PAMI-8**(1), 15 –25.

[182] Zhang, S. (2005). *High-Resolution, Real-Time 3-D Shape Measurement*. Ph.D. thesis, Stony Brook University.

[183] Zhang, X., Lei, M., Yang, D., Wang, Y., and Ma, L. (2007). Multi-scale curvature product for robust image corner detection in curvature scale space. *Pattern Recognition Letters*, **28**(5), 545 – 554.

[184] Zhang, X.-F., Luo, A., Tao, W., and Burkhardt, H. (1997). Camera calibration based on 3d-point-grid. In *Proceedings of the 9th International Conference on Image Analysis and Processing-Volume I - Volume I*, ICIAP '97, pages 636–643, London, UK. Springer-Verlag.

[185] Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, **22**, 1330–1334.

[186] Zhong, B. and Liao, W. (2007). Direct curvature scale space: Theory and corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **29**(3), 508 –512.

[187] Zhuang, H. and Wu, W.-C. (1996). Camera calibration with a near-parallel (ill-conditioned) calibration board configuration. *Robotics and Automation, IEEE Transactions on*, **12**(6), 918 –921.

[188] Zollner, H. and Sablatnig, R. (2004). Comparision of methods for geometric camera calibration using planar calibration targets. In W. Burger and J. Scharinger,

editors, *Digital Imaging in Media and Education, Proc. of the 28th Workshop of the Austrian Association for Pattern Recognition (OAGM/AAPR)*, volume 179, pages 237–244. Schriftenreihe der OCG.