HIGH PERFORMANCE DIGITAL VOLUME CORRELATION

BY

MARK RALPH GATES

DISSEYATION
Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:
Professor Michael T. Heath, Chair and Co-Director of Research
Professor John Lambros, Co-Director of Research
Professor Michael Sutton, University of South Carolina
Professor William Gropp
Assistant Professor Luke Olson
Abstract

We develop speed, efficiency, and accuracy improvements to a three-dimensional (3D) digital volume correlation (DVC) algorithm, which measures displacement and strain fields throughout the interior of a material. Our goal is to perform DVC with resolution comparable to that achieved in 2D digital image correlation, in time that is commensurate with the image acquisition time. This represents a significant improvement over the current state-of-the-art available in the literature. Using an X-ray micro-CT scanner, we can resolve features at the 5 micron scale, generating 3D images with up to 36 billion voxels. We utilize linear and quadratic shape functions with tricubic spline interpolation to achieve high accuracy. We improve the algorithm’s speed and robustness through an improved coarse search, efficient implementation of spline interpolation, and using smoothing splines to address noisy image data. For DVC, the volume of data, number of correlation points, and work to solve each correlation point all grow cubically. We therefore employ parallel computing to handle this tremendous increase in computational and memory requirements. We study how various parameters affect the accuracy of the solution, and how to refine the solution to achieve improved accuracy at reduced computational cost. We demonstrate the effectiveness of our improved DVC implementation using simulated deformations of 3D micro-CT scans of polymer and ceramic foam samples.
Acknowledgements

My graduate studies and research at the University of Illinois have been influenced, guided, and supported by many people. First, I am grateful to my advisors, Prof. Michael Heath and Prof. John Lambros, who guided me in my graduate studies and through the development of DVC. It has been a pleasure to work with them on this project. I am also grateful to my other committee members, Prof. Michael Sutton, Prof. William Gropp, and Prof. Luke Olson, who provided valuable feedback for my research.

Numerous people provided technical assistance. I am grateful to Charles Mark Bee, Leilei Yin, and the Imaging Technology Group at the Beckman Institute for use of the Skyscan MicroCT and Xradia MicroCT scanners. I thank Brett Beiermann and Prof. Nancy Sottos of the Autonomic Materials Research Group at the Beckman Institute for help in preparing PDMS samples, Christian Espinoza and Prof. W. Kriven of the Department of Materials Science and Engineering for providing ceramic foam samples, Dr. Jay Carroll of the Department of Mechanical Science and Engineering for providing 2D images for testing, and Emily Zavala and Rebecca Mudrock for scanning images using the Xradia scanner.

I gratefully acknowledge the use of the Turing cluster maintained and operated by the Computational Science and Engineering Program at the University of Illinois. This work was supported by the Center for Simulation of Advanced Rockets under contract number B523819 funded by the U.S. Department of Energy, by the
Institute for Advanced Computing Applications and Technologies, and by award number 09084 from the University of Illinois Campus Research Board.

My colleagues in scientific computing have been very helpful in discussing ideas about optimization, image processing, GPU computing, Python, and numerous other topics. Particularly, I am appreciative of conversations with Russ Hewett, Steven Dalton, Peng Jiang, Van Bui, James Lai, Jehanzeb Chaudhry, Adam Reichert, Jacob Schroder, Nana Arizumi, and Hannah Neradt.

Many friends have supported me personally through this endeavor. I thank my friends in the International Friendship Link for encouraging me, praying with me, and standing by me through life’s difficulties. Bob and Debbie, Eriko and Yuji, Bonnie and Craig, Esther, Pedro, Pradeep and Gloria, Mike and Teng-Lin, Arica, Steven, Jenna, Dan, Vera, Eric and Joy — your love has meant everything to me. To Chris, Dudley, and family and to Scott and Christina, thank you for a home. Thanks also to my friends in Graduate Christian Fellowship who encouraged me through the writing process.

I am deeply indebted to my parents, sisters and their families, who have encouraged me through these years of graduate school.
# Table of Contents

Notation ................................................................. vii

Chapter 1 Introduction ............................................. 1
  1.1 Motivation ......................................................... 1
  1.2 Problem Statement ............................................... 2
  1.3 Prior Work ...................................................... 7
  1.4 Objectives ...................................................... 10

Chapter 2 Experimental Setup ..................................... 13
  2.1 Sample preparation ............................................. 13
  2.2 Artificial test problems ....................................... 15
  2.3 Computer resources .......................................... 21

Chapter 3 Optimization Algorithm ................................ 22
  3.1 Coarse search .................................................. 22
  3.2 Final optimization ............................................. 35

Chapter 4 Interpolation and Approximation .................... 49
  4.1 Basis functions ................................................ 49
  4.2 Interpolating splines ......................................... 57
  4.3 Smoothing splines ............................................. 58
  4.4 Tensor-product splines ....................................... 61
  4.5 Comparison of methods ...................................... 62

Chapter 5 Scalability and Parallelism ........................... 65
  5.1 Storing 3D images ............................................ 65
  5.2 Splines over subset ........................................... 69
  5.3 Coarse-grained parallel computing ......................... 71
  5.4 Fine-grained parallel computing with GPU ............... 79

Chapter 6 Refinement and Adaptivity ............................. 87
  6.1 Error estimates ............................................... 87
  6.2 2D quadratic displacement model problem ................. 91
  6.3 3D quadratic displacement model problem ................. 99
  6.4 Determining refinement region ............................ 102
Chapter 7  Results and Discussion  ........................................... 107
  7.1  Baseline test  ................................................................. 108
  7.2  Translation test .............................................................. 111
  7.3  Spherical inclusion test ..................................................... 114

Chapter 8  Conclusions  ............................................................. 143

Appendix A  PDMS sample preparation  ................................. 147

Appendix B  Skyscan MicroCT procedure  .............................. 149
  B.1  Scanning ........................................................................ 149
  B.2  Reconstruction ............................................................. 150

References .............................................................................. 151
Notation

Lowercase bold type indicates a vector, such as \( \mathbf{q} \).

Uppercase bold type indicates a matrix or tensor, such as \( \mathbf{M} \), except as noted below.

Italic type indicates a scalar, such as \( c \). For components of a vector or matrix, the indices appear as subscripts, such as \( q_i \) and \( M_{ij} \).

\( \mathbf{x} \) is vector of coordinates \((x, y, z)\).

\( \mathbf{u} \) is vector of displacements \((u, v, w)\) in the \((x, y, z)\) directions, respectively, or vector of displacements and their derivatives, \((u, v, w, \frac{\partial u}{\partial x}, \ldots, \frac{\partial w}{\partial z})\).

\( \mathbf{X}, \mathbf{Y}, \mathbf{Z} \) are vectors of the \(x, y, z\) coordinates, respectively, of all correlation points.

\( \mathbf{U}, \mathbf{V}, \mathbf{W} \) are vectors of the \(u, v, w\) displacements, respectively, at all correlation points.
Chapter 1

Introduction

1.1 Motivation

The development of three-dimensional (3D) digital volume correlation (DVC), coupled with 3D imaging techniques such as X-ray CT scanners and confocal microscopy, has enabled the measurement of full 3D displacement and strain fields throughout the interior of a material undergoing motion or deformation. DVC is an extension of two-dimensional (2D) digital image correlation (DIC), which measures in-plane surface displacements only. DVC is also distinct from 3D DIC, which uses stereo 2D images to measure both in-plane and out-of-plane displacements, but only on the surface of a material. In contrast to 2D and stereo DIC, DVC measures 3D displacements inside a material.

DVC provides a useful experimental complement to 3D numerical simulations such as finite element analysis (FEA). For example, DVC can generate the full-field 3D experimental results required to compare with 3D finite element simulations for validation purposes, results that are otherwise difficult or impossible to obtain. In addition to validation, DVC can also play a vital role in determining input parameters for simulations. For instance, it can be used to determine a representative
volume element—the smallest material element that still reveals global material response—to be used as a basis for material properties in homogenized simulations. Previously, representative volume elements have been determined by using 2D DIC to extract a representative surface element, and then assuming some type of extension into the third dimension [17, 45, 53]; DVC now offers the opportunity to attain a true 3D representative volume element. DVC is also useful to analyze complex materials, such as bone [5, 46, 73, 79], rock [42], synthetic foams [67], wood [21], sugar [20] and sand grains [35]; as well as complex behaviors, such as material fatigue after repeated loading cycles [10, 62], which are difficult to simulate. Thus, although only a relatively small number of researchers have studied DVC, it is a powerful tool with significant future potential.

1.2 Problem Statement

The DVC process starts with a sample possessing a random internal pattern of features that are detectable by the 3D imaging device. Such an internal “speckle” pattern can arise either from inherent internal material microstructure, such as in bone, or by specifically manufacturing samples with embedded particles [22, 28, 34]. As depicted in Figure 1.1, a reference 3D image of the undeformed sample is captured, then a motion or deformation is applied, and a 3D image of the resulting deformed sample is captured either under load (in situ) or after unloading (ex situ). In Figure 1.1, the 3D images captured are indicated by three horizontal slices, each one revealing internal pattern features. The 3D data, which are a collection of many such slices, represent intensity values at voxel locations of a 3D grid. We define a grid of correlation points on the reference image, with a 3D subset (subvolume) surrounding each point. For each correlation point, DVC determines the deformation that maps the subset in the reference image (Figure 1.1a) to the subset in the
Figure 1.1: DVC applied to 3D image with 15° rotation about vertical z axis. Subset in reference image (a) is mapped to subset in deformed image (b). Displacement field (c) computed on $4 \times 4 \times 3$ grid. Showing slices $z = 0$, $z = 25$, and $z = 50$ voxels.

deformed image (Figure 1.1b) with the best correlation. In Figure 1.1 this happens to be a rigid rotation in the horizontal plane.

This mapping between the reference image and the deformed image is given by a shape function which defines the degrees-of-freedom (DOF) to be determined at each point. The linear shape function [12] defines twelve DOF, namely the displacements in each dimension and their first derivatives,

$$
\mathbf{u} = \left[ \begin{array}{c}
u \\
v \\
w \\
\frac{\partial u}{\partial x} \\
\frac{\partial u}{\partial y} \\
\frac{\partial u}{\partial z} \\
\frac{\partial v}{\partial x} \\
\frac{\partial v}{\partial y} \\
\frac{\partial v}{\partial z} \\
\frac{\partial w}{\partial x} \\
\frac{\partial w}{\partial y} \\
\frac{\partial w}{\partial z} \end{array} \right]^T.
$$

As illustrated in Figure 1.2, for a correlation point $p = [p_x, p_y, p_z]$, we let $f(x)$ be the subset in the reference image and $g(\hat{x}(u))$ be the subset in the deformed image, with a point $x = [x, y, z]$ in the reference subset related to the corresponding point
\[ x(u) \]

in the deformed subset by the linear shape function

\[
\begin{bmatrix}
\hat{x}(u) \\
\hat{y}(u) \\
\hat{z}(u)
\end{bmatrix}
= \begin{bmatrix}
x + u + \frac{\partial u}{\partial x} \Delta x + \frac{\partial u}{\partial y} \Delta y + \frac{\partial u}{\partial z} \Delta z \\
y + v + \frac{\partial v}{\partial x} \Delta x + \frac{\partial v}{\partial y} \Delta y + \frac{\partial v}{\partial z} \Delta z \\
z + w + \frac{\partial w}{\partial x} \Delta x + \frac{\partial w}{\partial y} \Delta y + \frac{\partial w}{\partial z} \Delta z
\end{bmatrix}, \quad (1.1)
\]

with

\[
\Delta x = x - p_x, \quad \Delta y = y - p_y, \quad \Delta z = z - p_z
\]

defining a local coordinate system within each subset.

Alternatively, the quadratic shape function results in greater accuracy in matching an underlying nonlinear deformation [48, 70]. It adds second derivative terms, for a total of thirty DOF at each point. As illustrated in Figure 1.3, points in the
deformed subset are related to points in the reference subset by

\[ \hat{x}(u) = x + u + \frac{\partial u}{\partial x} \Delta x + \frac{\partial u}{\partial y} \Delta y + \frac{\partial u}{\partial z} \Delta z \]
\[ + \frac{\partial^2 u}{\partial x^2} \Delta x^2 + \frac{\partial^2 u}{\partial y^2} \Delta y^2 + \frac{\partial^2 u}{\partial z^2} \Delta z^2 + \frac{\partial^2 u}{\partial x \partial y} \Delta x \Delta y + \frac{\partial^2 u}{\partial x \partial z} \Delta x \Delta z + \frac{\partial^2 u}{\partial y \partial z} \Delta y \Delta z, \]

\[ \hat{y}(u) = y + v + \frac{\partial v}{\partial x} \Delta x + \frac{\partial v}{\partial y} \Delta y + \frac{\partial v}{\partial z} \Delta z \]
\[ + \frac{\partial^2 v}{\partial x^2} \Delta x^2 + \frac{\partial^2 v}{\partial y^2} \Delta y^2 + \frac{\partial^2 v}{\partial z^2} \Delta z^2 + \frac{\partial^2 v}{\partial x \partial y} \Delta x \Delta y + \frac{\partial^2 v}{\partial x \partial z} \Delta x \Delta z + \frac{\partial^2 v}{\partial y \partial z} \Delta y \Delta z, \]

\[ \hat{z}(u) = z + w + \frac{\partial w}{\partial x} \Delta x + \frac{\partial w}{\partial y} \Delta y + \frac{\partial w}{\partial z} \Delta z \]
\[ + \frac{\partial^2 w}{\partial x^2} \Delta x^2 + \frac{\partial^2 w}{\partial y^2} \Delta y^2 + \frac{\partial^2 w}{\partial z^2} \Delta z^2 + \frac{\partial^2 w}{\partial x \partial y} \Delta x \Delta y + \frac{\partial^2 w}{\partial x \partial z} \Delta x \Delta z + \frac{\partial^2 w}{\partial y \partial z} \Delta y \Delta z. \]

To determine the deformation \( u \), DVC seeks the best match between \( f \) and \( g \) by minimizing a function that measures their difference, computed as a summation over all points \( x \) in the subset centered at \( p \). Two different objective functions are
commonly used, the least-squares correlation function

\[ c(u) = \frac{\sum_x (f(x) - \hat{g}(\hat{x}(u)))^2}{\sum_x f(x)^2} = \frac{\|f - g\|^2}{\|f\|^2}, \tag{1.3} \]

and the normalized cross-correlation function [12],

\[ c(u) = 1 - \frac{\sum_x f(x) \hat{g}(\hat{x}(u))}{\left(\sum_x f(x)^2\right)^{1/2} \left(\sum_x \hat{g}(\hat{x}(u))^2\right)^{1/2}} = 1 - \frac{\langle f, g \rangle}{\|f\| \cdot \|g\|}. \tag{1.4} \]

The least-squares correlation function is clearly always non-negative. By the Cauchy-Schwarz inequality, the cross-correlation function is also always non-negative,

\[ |\langle f, g \rangle| \leq \|f\| \cdot \|g\| \Rightarrow \frac{|\langle f, g \rangle|}{\|f\| \cdot \|g\|} \leq 1 \Rightarrow 1 - \frac{\langle f, g \rangle}{\|f\| \cdot \|g\|} \geq 0. \]

If \( f \) and \( g \) exactly match, then \( c(u) = 0 \). In general, \( f \) and \( g \) do not exactly match, but we seek the best possible match by finding the deformation \( u \) that minimizes the objective function \( c(u) \).

Since the deformation \( u \) results in non-integer coordinates \( \hat{x}(u) \), some form of interpolation is required to evaluate the deformed image \( g(\hat{x}(u)) \) between voxels. Choices include \( C^0 \) trilinear interpolation [12, 28, 42], \( C^1 \) tricubic Hermite interpolation [5, 73, 79], and \( C^2 \) tricubic spline interpolation [8, 21]. In 2D DIC, \( C^4 \) biquintic spline interpolation has also been used [71]. Interpolation methods with higher continuity often result in more accurate DVC results, though for some patterns such as with a bimodal gray level distribution, the opposite is true [71].

For each correlation point, DVC first finds an initial guess using a coarse search that evaluates the correlation function at a grid of points. The result of the coarse search is then used as the initial guess for a minimization algorithm. Various mini-

---

\(^1\) Least-squares correlation function is also known as sum-of-squared differences (SSD), sum-of-squares correlation coefficient (SSCC) [73], and maximum likelihood estimation (MLE) [46] in the DIC and DVC literature.
mization algorithms have been used for DVC, including Levenberg-Marquardt [5] and BFGS [73], both of which use first derivative information to approximate the Hessian in Newton’s method, thereby reducing the cost per iteration. Steepest descent has also been used [28, 79], but it converges slowly unless the problem is very well conditioned, so is not recommended in general.

1.3 Prior Work

Digital volume correlation has been developed by various research groups since 1999, as summarized in Table 1.1. Bay et al. [5] first developed DVC using the three translation degrees-of-freedom (DOF) \(u, v, w\) and \(C^1\) tricubic Hermite interpolation. They studied CT scans of trabecular bone, using the internal microstructure of the bone as the pattern for DVC. For a \(580 \times 580 \times 520\) voxel image, they computed 5500 correlation points at 7 seconds per \(61^3\) subset using a Sparc 10. Smith et al. [73] extended this by adding three rotational DOF about the \(x, y,\) and \(z\) axes. Their code took 6–19 seconds per \(51^3\) subset on a Sparc 10. Franck et al. [22] instead added the three axial strain DOF \(\frac{\partial u}{\partial x}, \frac{\partial v}{\partial y},\) and \(\frac{\partial w}{\partial z}\), assuming rotations and shear strains are negligible, and used an FFT-based algorithm to minimize the correlation function. They studied soft agarose gel with embedded particles, using a confocal microscope to obtain 3D scans, targeted at extending DVC to biological applications. They computed 3375 points with \(64^3\) subsets. Roeder et al. [66] also used an FFT-based algorithm to examine a collagen matrix scanned with a confocal microscope, computing a small grid of 320 points. To attain a full linear transformation, Verhulp et al. [74] extended DVC to use twelve DOF. Their code took 40 seconds per \(17^3\) subset on an \(800\) MHz PC, and computed 2130 points. Germaneau et al. [28] also used twelve DOF, but with less expensive trilinear interpolation. They studied epoxy with embedded particles, using light scattering to produce a 3D scan. Lenoir et al. [42] stud-
<table>
<thead>
<tr>
<th>Authors</th>
<th>Year</th>
<th>Material</th>
<th>Image interpolation</th>
<th>Shape function</th>
<th>Optimization method</th>
<th>Correlation points</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subset-based DVC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bay et al. [5]</td>
<td>1999</td>
<td>bone</td>
<td>C^1 cubic</td>
<td>translation</td>
<td>Levenberg</td>
<td>5500</td>
</tr>
<tr>
<td>Smith et al. [73]</td>
<td>2002</td>
<td>bone</td>
<td>C^1 cubic</td>
<td>rotation</td>
<td>BFGS</td>
<td>125</td>
</tr>
<tr>
<td>Roeder et al. [66]</td>
<td>2004</td>
<td>collagen</td>
<td>n/a</td>
<td>translation</td>
<td>FFT</td>
<td>7569</td>
</tr>
<tr>
<td>Verhulp et al. [74]</td>
<td>2004</td>
<td>Al foam</td>
<td>C^1 cubic</td>
<td>linear</td>
<td>BFGS</td>
<td>2130</td>
</tr>
<tr>
<td>Zauel et al. [79]</td>
<td>2006</td>
<td>bone</td>
<td>C^1 cubic</td>
<td>translation</td>
<td>steepest descent</td>
<td>—</td>
</tr>
<tr>
<td>Franck et al. [22]</td>
<td>2007</td>
<td>agarose</td>
<td>n/a</td>
<td>axial strain</td>
<td>FFT</td>
<td>3375</td>
</tr>
<tr>
<td>Germeneau et al. [28]</td>
<td>2007</td>
<td>epoxy</td>
<td>C^0 linear</td>
<td>linear</td>
<td>steepest descent</td>
<td>—</td>
</tr>
<tr>
<td>Lenoir et al. [42]</td>
<td>2007</td>
<td>rock</td>
<td>C^0 linear</td>
<td>translation</td>
<td>steepest descent</td>
<td>60,000</td>
</tr>
<tr>
<td>Forsberg and Siviour [20]</td>
<td>2009</td>
<td>sugar</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>6000</td>
</tr>
<tr>
<td>Forsberg et al. [21]</td>
<td>2010</td>
<td>wood</td>
<td>C^2 cubic spline</td>
<td>linear</td>
<td>Newton</td>
<td>960</td>
</tr>
<tr>
<td>Hall et al. [35]</td>
<td>2010</td>
<td>sand</td>
<td>C^0 linear</td>
<td>rotation</td>
<td>—</td>
<td>50,000</td>
</tr>
<tr>
<td>Gates et al. [24, 25]</td>
<td>2010</td>
<td>PDMS</td>
<td>C^2 cubic spline</td>
<td>linear</td>
<td>Newton, BFGS</td>
<td>59,000</td>
</tr>
<tr>
<td>Gates et al. [26, 27]</td>
<td>in preparation</td>
<td>foam</td>
<td>C^2 cubic spline</td>
<td>linear,</td>
<td>BFGS</td>
<td>1,000,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>quadratic</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Finite Element-based DVC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roux et al. [67]</td>
<td>2008</td>
<td>solid foam</td>
<td>—</td>
<td>linear</td>
<td>FEM</td>
<td>1331</td>
</tr>
<tr>
<td>Hild et al. [37]</td>
<td>2009</td>
<td>stone wool</td>
<td>—</td>
<td>linear</td>
<td>FEM</td>
<td>512</td>
</tr>
<tr>
<td>Réthoré et al. [64]</td>
<td>2008</td>
<td>cast iron</td>
<td>—</td>
<td>linear + crack</td>
<td>FEM</td>
<td>539</td>
</tr>
<tr>
<td>Rannou et al. [62]</td>
<td>2010</td>
<td>cast iron</td>
<td>—</td>
<td>linear + crack</td>
<td>FEM</td>
<td>729</td>
</tr>
<tr>
<td>Leclerc et al. [41]</td>
<td>2010</td>
<td>cast iron</td>
<td>—</td>
<td>translation</td>
<td>voxel scale</td>
<td>243 voxels</td>
</tr>
<tr>
<td>Limodin et al. [44]</td>
<td>2010</td>
<td>cast iron</td>
<td>C^2 cubic spline</td>
<td>linear</td>
<td>FEM</td>
<td>—</td>
</tr>
<tr>
<td><strong>Particle Tracking</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Haldrup et al. [34]</td>
<td>2006</td>
<td>Al</td>
<td>n/a</td>
<td>translation</td>
<td>tracking</td>
<td>15,000</td>
</tr>
<tr>
<td>Kobayashi et al. [40]</td>
<td>2008</td>
<td>Cu</td>
<td>n/a</td>
<td>translation</td>
<td>tracking</td>
<td>188</td>
</tr>
</tbody>
</table>

— denotes entries not reported in papers.
* denotes materials with particles or markers added to form a pattern.

**Table 1.1:** Summary of prior work on DVC to date.
ied triaxial compression of argillaceous rock, computing 60,000 points, but using only three DOF and trilinear interpolation. Zauel et al. [79] used three DOF and $C^1$ tricubic interpolation, with steepest descent and a quadratic line search. Their code was optimized by using SIMD instructions to achieve 0.47 seconds per point on a 1.8 GHz Pentium 4. They used spherical subsets with a radius of 30 voxels. They also compared DVC results with a finite-element model. Bay [4] used DVC to examine indentation of metallic foam. Forsberg and Siviour [20] studied compacted sugar, using $32^3$ subsets and computing about 6000 points over a $342 \times 342 \times 380$ voxel image. Forsberg et al. [21] investigated a three-point bending problem with wood, using tricubic spline interpolation, an FFT for a coarse search, and Newton’s method for the fine search. They reduced the $x$ and $y$ resolution by $2 \times 2$ binning to fit their computational resources, and used $64^3$ subsets with a $128 \times 768 \times 320$ voxel image. Hall et al. [35] studied compaction of sand grains. Instead of defining a regular Cartesian grid, they segment the image to identify individual sand grains and center a subset on each sand grain. They used six translation and rotational DOF, assuming the grains were rigid, and computed DVC for 50,000 grains at 1 second per $27^3$ subset, plus approximately 0.55 second per grain to segment the image, on a 2.3 GHz Xeon. Their correlation process required more than 8 GB of memory for an approximately $785 \times 785 \times 1570$ voxel image.

Instead of the subset based DVC presented here, Roux et al. introduced a finite-element based DVC [37, 44, 67]. This approach defines a finite-element mesh on the image and computes the deformation of the entire mesh at once, instead of computing each subset independently. It enforces global continuity of displacements, but has also been extended to allow for crack discontinuities [62]. They studied small $100^3$ to $288^3$ voxel regions using elements from $6^3$ to $18^3$ voxels. Using this finite-element DVC, Limodin et al. [44] studied the effects of CT scanner artifacts and the spurious strain induced by thermal expansion caused by the X-rays in the
scanner. Recently, Leclerc et al. [41] presented a voxel-scale DVC where the displacement of each voxel is determined (i.e., a subset of size 1) with the addition of the regularization that local behavior is elastic, to make the problem well-posed. As with the finite-element DVC, the entire solution is computed as a single system.

Another alternative method that has been used is tracking individual particles between successive CT scans of a material. Haldrup et al. [34] used features such as particle size, moment, and relative position to other particles to track 15,000 individual tungsten marker particles in an aluminum sample. Kobayashi et al. [40] tracked 188 micro-pores in copper alloy samples, using volume, surface area, and distance between markers to identify markers.

1.4 Objectives

In this work our goals are to (i) perform DVC with resolution (i.e., density of correlation points in each dimension) and accuracy comparable to that achieved in 2D DIC, and (ii) do so in a “real-time” fashion, defined here as a correlation time that is commensurate with the image acquisition time. Due to the vastly increased volume of data associated with the undeformed and deformed images, DVC requires substantially more computation and storage than DIC to achieve similar resolution. We approach this problem on multiple fronts, which may not have been critical for 2D DIC, but are essential to effective high performance and high resolution 3D DVC because of its substantially greater complexity. In this thesis we describe the steps we have taken in improving the DVC algorithm with an eye to fulfilling the two goals above.

We suggest a refinement of the coarse search in Section 3.1 that improves both speed and robustness. 2D DIC has typically used $C^2$ bicubic spline interpolation [8] or $C^4$ biquintic spline interpolation [71], which has been shown to often give more...
accurate results than $C^0$ bilinear and $C^1$ bicubic interpolation [71]. We therefore enhance DVC by using $C^2$ tricubic spline interpolation, and discuss how to implement it efficiently for DVC in Chapter 4. Images generated by many 3D imaging techniques, such as X-ray CT scanners, have higher noise levels than typical images captured by 2D digital cameras. We propose using a smoothing spline to address noisy image data.

Several DVC studies have commented on the advantages of having a large computer memory in which to load an entire 3D data set [5, 35, 42]. The size of the 3D images grows cubically, however, so a modest increase in image resolution generates a substantially larger image. For our current CT scanner, loading a pair of medium resolution images into memory requires up to 8 GB of RAM, while at the maximum resolution, a pair of images would require up to 68 GB of RAM, far exceeding the memory available on most computers. We observe that only a small portion of this image data is in use at any one time, however, so by careful memory management we have developed an algorithm that scales to much larger images, described in Section 5.1.

Most DVC studies have been relatively modest in size, computing up to 6000 points, with a couple of larger studies up to 60,000 points. In contrast, a typical 2D grid of correlation points might be $40^2$ to $100^2$, so a comparable $40^3$ to $100^3$ resolution in 3D requires a grid with 64,000 to 1,000,000 points. Such high resolution is required to resolve high strain gradients within a material effectively. To reach our goal of a $100^3$ point correlation grid, at 3 seconds per correlation point, would require 35 days of computing time. A more modest $40^3$ grid of 64,000 points would still require over two days. Faster processors would reduce that time, but ultimately the cubic growth of 3D image data will exceed what a single processor can process in a reasonable amount of time. We therefore turn to parallel computing to gain sufficient processing power to scale to these large problem sizes, as discussed.
in Chapter 5. This also allows us to fully utilize the processing power of modern multicore processors.

The error in displacements measured by DVC is a function of the subset size, shape function, image pattern, and deformation field being measured. Because the image pattern and deformation field vary throughout the sample, no single choice of subset size and shape function will be optimal. This suggests that refinement of the DVC solution is necessary, which we discuss in Chapters 6 and 7.
Chapter 2

Experimental Setup

3D data sets of internal contrast in materials can be generated using techniques such as X-ray CT scanning or confocal microscopy. A wide variety of materials such as bone [5, 46, 73, 79], rock [42], synthetic foams [67], and metals [62, 34] have been studied using these techniques. Faithfully duplicating realistic imaging conditions, such as image size, resolution, and particularly noise, is an inherent part of the robustness of the DVC technique. Therefore, although this effort is primarily computational in nature, we will use images acquired through actual CT scanning. However, as the issue here is the algorithmic development of DVC to fulfill our goals, we will use artificial deformations of real images to develop and verify our implementation. The artificial deformation gives us a known solution to evaluate the accuracy of our method, apart from additional experimental noise induced by actual load tests.

2.1 Sample preparation

We use several materials in our experiments. For one material, we manufactured PDMS (polydimethylsiloxane, commonly known as silicone rubber) samples with
embedded silica particles to form a random pattern suitable for DVC, shown in Figure 2.1a. Samples had particles ranging in size 50–250 μm, totaling between 13.6%–25.5% by mass of the sample (5.5%–11.1% by volume). The PDMS was obtained from Gelest, Inc., and has viscosity 700–800 cSt. We used a fast reaction catalyst that solidifies the PDMS in one minute, with full cure in one day. The PDMS base and linker are mixed with the particles, degassed, mixed with the catalyst, poured into a cylindrical mold and left to cure at room temperature for one day. Cylindrical samples of 11mm diameter and 14mm height were obtained in this way. Samples were made with the assistance of Brett Beiermann and Prof. N. Sottos of the Department of Materials Science and Engineering at the University of Illinois.

The second material is a ceramic foam, shown in Figure 2.1b, manufactured using a colloidal suspension of 35 vol% alumina powder in 0.03 M valeric acid as a fatty acid stabilizer. The solution was foamed for 2–3 minutes using a kitchen hand mixer, dried at 60–120 °C for 30 minutes, then sintered at 1550 °C for 4 hours. The foam was provided by Christian Espinoza and Prof. W. Kriven of the Department of Materials Science and Engineering at the University of Illinois.

To generate 3D scans, we used a Skyscan 1172A MicroCT scanner and an Xradia MicroXCT-200 scanner, located at the Beckman Institute of the University of Illinois at Urbana-Champaign. The Skyscan has a maximum resolution of 5 μm, generating a 4000 × 4000 × 2300 voxel image. For the Skyscan results discussed in this thesis, we scanned samples at the medium resolution of 10 μm and 2000 × 2000 × 1150 voxels, or the coarse resolution of 20 μm and 1000 × 1000 × 575 voxels. The sample was affixed to a stage, which the scanner rotates 360° about the vertical axis with fixed angular steps (Figure 2.2a). During a single scan, the Skyscan takes multiple X-ray images at each angular step and averages them to reduce noise. We found that averaging 5 images yielded significant noise in the reconstructed im-
age. By increasing this to averaging 20 images at each step, the noise was reduced to show a clear distinction between the particles and the background. With averaging 20 images, a single scan with X-ray images at 900 angular steps took 110 minutes at 10 μm resolution. Note that different CT scanners have different noise characteristics; the large number of images being averaged may just be a function of the Skyscan. Skyscan’s Nrecon software reconstructed the 3D image, using a cone-beam reconstruction algorithm. Care was taken to remove artifacts using the maximum ring artifact reduction and beam hardening compensation. As shown recently in Limodin et al. [44], ring artifacts degrade the accuracy of DVC, while beam hardening and its correction had negligible effects. However, more investigation is warranted to assess the impact of the ring artifact reduction on accuracy. The reconstruction output is a sequence of 2D 8-bit images, each being a slice of the 3D image, as shown pictorially in Figure 2.1a, where only 5 slices out of the 1150 are shown.

We also scanned samples using an Xradia MicroXCT-200 at the Beckman Institute, pictured in Figure 2.2b. It has a maximum resolution of 1 μm and generates a $992 \times 1013 \times 994$ voxel image. The Xradia scanner appeared to have significantly less image noise than the Skyscan scanner, thus producing better DVC results. We used a three second exposure time, resulting in a scan time of 70 minutes. The ceramic foam was scanned at a resolution of 5 μm, as shown in Figure 2.1b, where 5 slices out of the 994 are shown. Emily Zavala of the Department of Aerospace Engineering at the University of Illinois scanned the ceramic foam samples.

### 2.2 Artificial test problems

For baseline tests, we took two consecutive scans, using the same settings and without moving (other than the tomograph rotation) or deforming the sample. Ideally,
(a) PDMS at 10 μm resolution  (b) Ceramic foam at 5 μm resolution

Figure 2.1: Slices of reconstructed 3D images. Blue inset boxes show typical \(41^3\) subset size.

(a) Diagram of CT scanner  (b) Xradia CT scanner interior with sample mounted inside load frame

Figure 2.2: CT scanner takes X-ray images as sample rotates about vertical axis.
Figure 2.3: Interpolation bias error using cubic spline, quintic spline, and Fourier series interpolation to generate images, and cubic spline interpolation to compute DVC. Deformed images are translated by $u^*$ voxels in $x$ direction; $u$ is displacement measured by DVC.

the displacement field computed by DVC for a pair of baseline images is zero everywhere, but mechanical perturbations in the scanner may introduce some rigid body motion, and scanner noise also affects the results. Consequently, baseline DVC runs provide us with a quantitative measure of these two effects.

Strain inducing deformations are generated artificially based on a particular 3D elastic solution. We applied the artificial deformation to one of the two baseline images by computing the analytical displacement field at every voxel and evaluating the image at those points using spline interpolation. We then performed DVC between the artificially deformed first image and the second, undeformed, image. By performing DVC between two separate scans in this way, we include the effects of image noise in our tests. It is important to use this process to investigate the robustness of DVC, since using the same image for both the undeformed image and to generate the deformed image always produces excellent results, as the exact same noise is present in both deformed and undeformed images. Since this
deformation is imposed artificially, there is concern that the imposed deformation will suffer from artificial error that would not be present in an actual experiment. In fact, as discussed in Schreier et al. [71], interpolation can introduce a phase shift in the displacement, causing an artificial systematic error in the subsequent DVC results. For translation tests, we use the FFT and apply the Fourier transform shift theorem to translate the image without introducing phase error [65, 71]. Fourier series interpolation can also be used for arbitrary deformations, but the FFT algorithm no longer applies. For a 3D image, the computational complexity increases from $O(n^3 \log(n))$ with the FFT to $O(n^6)$ without the FFT, making Fourier interpolation prohibitively expensive. Therefore, to generate deformed images with arbitrary deformations, we use triquintic spline interpolation, which has complexity $O(n^3)$. Higher order triquintic splines reduce the bias error compared to tricubic splines. Figure 2.3 compares the phase error for translation tests using cubic spline, quintic spline, and Fourier series interpolation to generate the deformed image, and cubic spline interpolation in the subsequent DVC process to measure the displacement. The error for quintic spline interpolation is within 0.005 pixels of the error for Fourier series interpolation, showing that quintic spline interpolation is an acceptable means to generate the deformed image.

In addition to tests with constant displacement and constant axial strain (i.e., linear displacement), we generated two problems with nonlinear displacement fields. The first problem is a quadratic displacement test. We define three regions in the image, as shown in Figure 2.4 for one dimension. In the left region, where $x < x_1$, a constant strain $\varepsilon_1$ is applied; in the right region, where $x_2 < x$, a constant strain $\varepsilon_2$ is applied; in the middle region, where $x_1 < x < x_2$, the strain varies linearly from
Figure 2.4: (a) Displacement field that is linear for $x < x_1$, quadratic for $x_1 < x < x_2$, and linear for $x_2 < x$. (b) Corresponding strain field.

$\varepsilon_1$ to $\varepsilon_2$, creating a quadratic displacement field. The displacement field is given by

$$u(x) = \begin{cases} 
  u_0 + \varepsilon_1 x, & x < x_1, \\
  u_0 + \varepsilon_1 x + \frac{\varepsilon_2 - \varepsilon_1}{2(x_2 - x_1)}(x - x_1)^2, & x_1 < x < x_2, \\
  u_0 + \varepsilon_1 x_1 + \frac{\varepsilon_2 - \varepsilon_1}{2(x_2 - x_1)}(x_2 - x_1)^2 + \varepsilon_2 (x - x_2), & x_2 < x,
\end{cases} \quad (2.1)$$

with the corresponding strain field

$$\varepsilon(x) = \begin{cases} 
  \varepsilon_1, & x < x_1, \\
  \varepsilon_1 + \frac{\varepsilon_2 - \varepsilon_1}{x_2 - x_1} (x - x_1), & x_1 < x < x_2, \\
  \varepsilon_2, & x < x_1.
\end{cases} \quad (2.2)$$

The constants $x_1$, $x_2$, $\varepsilon_1$, $\varepsilon_2$ can be varied to obtain quadratic regions of different widths and strain gradients. This can be extended by defining similar regions in the $y$ and $z$ dimensions for the $v$ and $w$ displacements to make regions that are triquadratic.
The second test problem is that of a rigid spherical inclusion of radius $R$ in a linear elastic matrix subjected to uniaxial far-field tension $T$, as depicted in Figure 2.5. The displacement field for this problem has been obtained by Goodier [33]. In the medium outside the sphere, the displacements in polar coordinates are

$$u_r = -\frac{A}{r^2} - \frac{3B}{r^4} + \left[\frac{(5 - 4\nu)C}{(1 - 2\nu)r^2} - \frac{9B}{r^4}\cos 2\theta\right] + \frac{TR}{2E} [1 - \nu + (1 + \nu)] \cos 2\theta,$$

$$u_\theta = -\left[\frac{2C}{r^2} + \frac{6B}{r^4}\right] \sin 2\theta - \frac{T\nu}{2E} (1 + \nu) \sin 2\theta,$$

$$A = -\frac{TR^3}{4\mu} \left[\frac{1 - \nu}{1 + \nu} + \frac{3}{16 - 20\nu}\right],$$

$$B = -\frac{8\mu (8 - 10\nu)}{TR^5},$$

$$C = -\frac{5TR^3 (1 - 2\nu)}{8\mu (8 - 10\nu)}.$$  

with distance $r$ from the center of the sphere, angle $\theta$ from the positive $x$ axis, tension $T$ applied parallel to the $x$ axis, Poisson’s ratio $\nu$, Young’s modulus $E$, and shear modulus $\mu$. The origin is at the center of the sphere. We selected $\nu = 0.49$, $E = 870$ KPa, and $T = 30$ KPa, as being typical for the PDMS used here [47] in order to have realistic displacement values for realistic load levels, although in this computational analysis the particular properties are not significant.
2.3 Computer resources

We computed DVC results on Turing, an Apple X-serve cluster with 768 nodes, each with two 2 GHz PowerPC G5 processors and 4 GB RAM, running Fedora Core 8. For GPU tests, we computed results on a PC with a 2.66 GHz Intel Core 2 Quad core CPU and Tesla C2050 GPU.
Chapter 3

Optimization Algorithm

For each correlation point, we optimize the correlation objective function, either the least-squares function (1.3) or the cross-correlation function (1.4). We start by performing a coarse search to find an initial guess that is near the global minimum, then use a standard nonlinear optimization method such as Newton’s method, BFGS, or Levenberg-Marquardt to refine the solution to sub-voxel accuracy. Having an improved initial guess increases both the speed and reliability of the subsequent final optimization. There is a trade-off between time spent doing a precise coarse search and time spent in the final optimization, so our goal is to find an initial guess quickly that is sufficiently good for our final optimization. We start by discussing the coarse search in Section 3.1, then discuss the final optimization in Section 3.2. Portions of this chapter previously appeared in Gates et al. [25].

3.1 Coarse search

3.1.1 Exhaustive search within area

The coarse search evaluates the correlation objective function on a regular grid and chooses the \( u \) with the minimum function value. The original 2D DIC coarse-fine
method [12] defines a search range \([u_{\text{min}}, u_{\text{max}}] \times [v_{\text{min}}, v_{\text{max}}]\) with step sizes \(\Delta u, \Delta v\) and evaluates the objective function at every candidate displacement \((u, v)\) in the search range. The displacement with minimum function value is chosen. Typically, step sizes \(\Delta u = 1, \Delta v = 1\) are chosen so the objective function is evaluated at integer pixel displacements. The derivative terms \(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y}\) are initially set to 0. After searching the initial range, the process is repeated with a smaller search range and finer step sizes, and also with search ranges for the derivative terms. This converges slowly and is computationally expensive, so Bruck et al. [8] used a single iteration of coarse-fine to generate an initial guess for Newton’s method, which then minimizes the objective function with respect to all six DOF simultaneously.

This coarse search, which we refer to as exhaustive search, is easily extended to 3D by adding a search range \([w_{\text{min}}, w_{\text{max}}]\) for \(w\), as depicted in Figure 3.1.

### 3.1.2 Expanding search from initial guess

One drawback of the exhaustive coarse search is that the user must provide a search range sufficiently large to accommodate any expected displacement. A large search
range dramatically increases the cost of doing a coarse search, but several improvements can be made to mitigate this increase. First, as neighboring correlation points have been computed, a starting point can be extrapolated from their results and the search region centered on that starting point. Bay et al. [5] suggest averaging the results of neighboring points. We improve on this by employing a first-order extrapolation. In each axial direction, if results for two neighboring points have been previously computed, we use a finite difference formula,

\[ u(x_i) = 2u(x_{i-1}) - u(x_{i-2}) , \]

to approximate the displacements at the current point \( x_i \). This results in up to three values, from extrapolating in each of the \( x, y, \) and \( z \) directions, which we average to obtain the starting point. If only one neighboring point in each direction has been computed, we revert to averaging neighboring results, while if no neighboring points have been computed, we use a user-supplied starting point, often zero. The coarse search region is then centered on this starting point. Often, however, this starting point itself is a good initial guess for the final optimization, in which case minimal additional coarse search is necessary.

Second, instead of defining a fixed search range, we expand from the starting point outward until a “good enough” initial guess is found. Here the challenge is determining a threshold for the objective function value that determines whether a point is close enough to the global minimum for Newton’s method or BFGS to converge; we discuss this in Section 3.1.3. When the solution is varying nearly linearly, only a few evaluations near the starting point are required, while for highly nonlinear or discontinuous regions, the search can expand outward as required to find a good initial guess. Figure 3.2a depicts an example expanding search, which first evaluates the starting point extrapolated from previously computed correla-
Figure 3.2: (a) Expanding search evaluates points in inner dark region around starting point, then points in light region, and stops when it finds blue circled point with objective function value below threshold. (b) Bracketing phase evaluates points around blue circled point; finds new point with smaller value (red diamond). (c) Bracketing phase continues by evaluating points around red diamond, and stops when point in red diamond is verified to be less than its neighbors.

We further require the result to have the minimum objective function value among its immediate neighbors in the search grid. As a consequence, if the point with minimum function value is on the boundary of the search region, we locally expand the search region to evaluate all its neighbors and check that its value is less than all its neighbors’ values. If a neighbor has a smaller function value, we choose that point as the new minimum point and repeat the check on its neighbors. This ensures that, at least in the axial directions, we have bracketed a minimum. To avoid re-computing function evaluations, the coarse search stores previously computed function evaluations. Because the grid on which points are evaluated can grow in an irregular fashion, instead of storing values in a fixed array, we store them in a hash table (map in the C++ standard template library [38]), using the tuple \((i, j, k)\) of integer coarse search grid indices as the hash key. This provides ef-
ficient, amortized constant time access to previously computed values and is simple to program. In Figure 3.2b and c, this bracketing phase evaluates an additional 9 points, finds a new minimum denoted by the red diamond, then evaluates another 9 points. It stops when it verifies this point is the minimum of its neighbors. For this example, the exhaustive search makes $7^3 = 343$ evaluations, while the expanding search makes only $5^3 = 125$ evaluations, and the bracketing phase adds 18 evaluations.

### 3.1.3 Correlation threshold

To determine a suitable threshold for the correlation objective function, we perform a calibration step by evaluating the objective function for sample correlation points at multiple trial displacements in the deformed image. Assuming a normal distribution, picking a value several standard deviations $\sigma$ below the mean $\mu$ provides a reasonably good threshold. Figure 3.3 shows distributions of objective function values for several subset sizes. The solid line indicates the mean value and dashed lines indicate standard deviations below mean.

![Histogram of objective function values](image)

**Figure 3.3:** Histogram of objective function $c(u)$ values for various subset sizes, evaluated at 1681 trial displacements. Solid line indicates mean value, dashed lines indicate standard deviations below mean. For subset size $35^3$, $\mu = 0.30$, $\sigma = 0.037$, and minimum is 0.071, so global minimum is more than six standard deviations below mean. Data for PDMS sample.
lines indicate standard deviations below the mean. If the coarse expanding search finds a function value below, say, \( \mu - 3\sigma \), then we have high confidence that the point is near the global minimum. For the subset size \( 35^3 \) in Figure 3.3, the mean is 0.30, the standard deviation is 0.037, and the global minimum is 0.071, so the minimum is more than six standard deviations below the mean. In the worst case, if the smallest objective function value in the coarse search is above this threshold, it will search the maximum search range, and then pick the smallest observed function value, even though it is above the threshold. The distribution of function values depends on the subset size. As the subset size increases, the distribution becomes narrower and the global minimum more well defined, as shown in Figure 3.3. Therefore selecting a good subset size is important to having a reliable coarse search.

### 3.1.4 FFT search algorithm

Over a small search area, computing the cross-correlation function directly in the spatial domain is efficient, while over a large search area it is more efficient to compute the cross-correlation function in the frequency domain using an FFT-based algorithm. For a subset size \( s \) and search region size \( n \), the computational complexity for searching in the spatial domain is \( O(s^3n^3) \), while the search using the FFT is \( O((s + n)^3 \log(s + n)) \). For small \( n \), these become \( O(s^3) \) and \( O(s^3 \log s) \), respectively, favoring the spatial algorithm. For \( n \approx s \), these become \( O(n^6) \) and \( O(n^3 \log n) \), respectively, clearly favoring the FFT algorithm. This is confirmed by our results in Figure 3.4, where for \( n < 4 \), computing in the spatial domain (dashed blue curve) is faster, while for larger \( n \), computing with the FFT (solid red line) is faster.

We suggest a hybrid algorithm that combines the benefits of the expanding and
Figure 3.4: Time to compute cross-correlation function with $31^3$ subset in spatial domain and using FFT. For large search region, FFT is faster than spatial algorithm. FFT is most efficient for sizes that are product of small primes, causing jagged line.

FFT searches. Our hybrid algorithm initially uses the expanding search in the spatial domain to search a small area, but if no suitable objective function values are found, we switch to the FFT algorithm to search a large region. This hybrid algorithm is useful for computing the first correlation point, since no good starting point is available. It is also useful when discontinuities such as a crack exist, requiring a search of a large area. While the FFT algorithm computes the cross-correlation function, even when we use the least-squares correlation function for the final optimization, we have found that using the FFT in the coarse search still provides a good initial guess.

Over a search region $[u_{\text{min}}, u_{\text{max}}] \times [v_{\text{min}}, v_{\text{max}}] \times [w_{\text{min}}, w_{\text{max}}]$, the numerator of the cross-correlation function (1.4) is computed as

$$\langle f, g \rangle = \text{IFFT} \left( \text{FFT}(f_{\text{pad}}) \circ \text{FFT}(g_{\text{pad}}) \right),$$
where FFT and IFFT are the forward and inverse FFT, the line over FFT denotes complex conjugation, and \( f_{\text{pad}} \) and \( g_{\text{pad}} \) are zero-padded copies of regions of \( f \) and \( g \). Zero padding is required because the FFT implicitly assumes that \( f \) and \( g \) are periodic. The subset \( f \) of size \( s \) and region \( g \) of size \( s + n \) must both be zero-padded to a size \( 2(s - 1) + n \) in each dimension. Also, \( f \) is flipped in each dimension, left-to-right, top-to-bottom, and front-to-back, in order to compute the correlation \( \langle f, g \rangle \) instead of the convolution \( f * g \). This flip can be done when copying data into the padded array, or by taking the complex conjugate of the FFT of \( f \), as denoted above. The FFTs of \( f \) and \( g \) are multiplied point-wise in the frequency domain, then an inverse FFT completes the process. The result has size \( (2(s - 1) + n)^3 \), but only the middle \( n^3 \) portion has valid results.

The \( \| g \| \) term in the denominator of (1.4) can also be computed with an FFT, by observing that it is the square root of the cross-correlation between the unit function over a single subset and \( g^2 \),

\[
\| g \| = \left( \langle 1, g^2 \rangle \right)^{1/2} = \left( \text{IFFT} \left( \text{FFT}(1_{\text{pad}}) \circ \text{FFT}(g^2_{\text{pad}}) \right) \right)^{1/2}.
\]

For each subset, the \( \| f \| \) term in (1.4) is a constant independent of \( u \), so can be efficiently computed directly without using the FFT.

We use the FFTW library [23], which implements a multi-dimensional FFT. The FFT is most efficiently implemented for sizes that are a product of small primes (e.g., 2, 3, 5, 7), and particularly for powers of 2. This accounts for the jagged line showing irregular performance in Figure 3.4. Hence, for a given FFT size, we increase the size to the next larger product of small primes.

This FFT algorithm has several limitations. First, transformations are limited to integer pixel translations, with no rotation or strain. This may cause the search to fail if large rotations or large strains are present. The exhaustive or expanding
search also normally searches whole-voxel translations, with derivatives set to zero, but can be modified easily to accommodate rotation or strain. Second, the FFT computes the entire search at once, whereas the expanding search computes points incrementally and can terminate as soon as it finds a good starting point.

### 3.1.5 Comparison of methods

#### 3D spherical inclusion problem

We demonstrate the effectiveness of the coarse search algorithms on two problems, the spherical inclusion problem (2.3) and a 2D crack problem. For the spherical inclusion, we use the ceramic foam sample and define a $14^3$ grid of 2744 points to cover one octant of the problem, with a grid spacing of 20 voxels and a $31^3$ subset size. We use BFGS for the subsequent minimization process. Results are computed in parallel using four processors on Turing (see Chapter 5 regarding parallelization). We use the static domain decomposition, so that all tests are decomposed identically, and exclude processor idle time at the end of the computation to avoid polluting results with load imbalance overhead.

With no computed starting point, the exhaustive search uses a fixed search range of $[0, 10] \times [-5, 0] \times [-5, 0]$, which is sufficiently large to encompass all deformations in the problem. As shown in Table 3.1, it takes the longest of all the methods, 1.17 seconds per correlation point. Two factors affect the performance. First, the exhaustive search makes a large number of objective function evaluations during the coarse search, 397 compared to 36 with the expanding search. Second, with no computed starting point, the exhaustive search generates an initial guess that is on the coarse search grid, i.e., $u, v, w$ have integer values. This results in more BFGS iterations in the subsequent optimization, compared to using a more accurate starting point.
Table 3.1: DVC results for spherical inclusion problem, using various starting points and coarse search algorithms, showing total number of failed points and average function evaluations, BFGS iterations, and time per correlation point. Hybrid search uses expanding and FFT searches.

When computing a starting point by averaging or extrapolating from neighbors’ results, the starting point already has sub-voxel accuracy. Using just the computed starting point for subsequent minimization, with no further coarse search, performs well for this problem. Averaging neighboring results reduces the subsequent BFGS minimization from 13.9 iterations to 11.5 iterations, while using our first-order extrapolation further reduced it to 9.3 iterations (Table 3.1). The time per correlation point decreases accordingly. These results are consistent across all the coarse search methods using these starting points. However, with no further coarse search, three points here failed to converge because the starting point was not sufficiently close to the global minimum. For more complex problems, e.g., with discontinuities or larger nonlinearities than the inclusion problem here, we expect even more failures without a coarse search, making this an unattractive solution.

When using a starting point with the exhaustive search, we narrow the search range to ±2 voxels from the starting point, which improves the performance by reducing the number of evaluations in the coarse search. Note, however, that for
problems with discontinuities, this search range may be inadequate. Frequently, the starting point has the smallest objective function value in the coarse search, so we see similar performance to using the starting point alone. The number of BFGS iterations is slightly reduced and all points succeed, due to instances when a point on the coarse search grid is a better initial guess than the starting point.

For the expanding search, we use a maximum search range of $\pm 5$ voxels from the starting point. On average, it requires only 36 or 37 evaluations per correlation point, resulting in an overall time 41% faster than the exhaustive search with no starting point, 0.69 versus 1.17 seconds per point, and slightly faster than the exhaustive search with a starting point.

The FFT coarse search is competitive with the exhaustive search, taking 1.08 and 1.17 seconds per point, respectively, even though the FFT here is searching a 23 times larger area. Combining the expanding and FFT searches into a hybrid search is 36% faster than the FFT search alone, and competitive with our plain expanding algorithm, both taking 0.69 seconds per point, but can efficiently search a much larger area.

2D crack problem

To investigate the robustness of the methods when there are discontinuities in the solution, our second problem studied a 2D crack using our 2D DIC implementation with 2D versions of the coarse search algorithms. A titanium sample was notched and then cyclically loaded to grow a crack. Images were then taken in the unloaded state (Figure 3.5a) and with the crack fully open (Figure 3.5b). We defined a $53 \times 31$ grid of 1643 correlation points, using a 25 voxel grid spacing and $51^2$ subset size. Subsets that overlap the crack itself are omitted from the results. The images were provided courtesy of Dr. Jay Carroll of the Department of Mechanical Science and Engineering at the University of Illinois [10].
(a) Unloaded state

(b) Crack fully open

Figure 3.5: Titanium sample before and after loading. Images courtesy of Dr. Jay Carroll [10].
<table>
<thead>
<tr>
<th>Starting point</th>
<th>Coarse search</th>
<th>Search area</th>
<th>Failed points</th>
<th>Function evaluations</th>
<th>BFGS iterations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>exhaustive</td>
<td>$41 \times 41$</td>
<td>0</td>
<td>1682</td>
<td>5.6</td>
<td>111.4</td>
</tr>
<tr>
<td>none</td>
<td>exhaustive</td>
<td>$11 \times 41$</td>
<td>0</td>
<td>452</td>
<td>5.6</td>
<td>52.0</td>
</tr>
<tr>
<td>extrapolate</td>
<td>exhaustive</td>
<td>$11 \times 21$</td>
<td>0</td>
<td>232</td>
<td>5.0</td>
<td>37.6</td>
</tr>
<tr>
<td>extrapolate</td>
<td>exhaustive</td>
<td>$5 \times 5$</td>
<td>71</td>
<td>26</td>
<td>4.9</td>
<td>27.4</td>
</tr>
<tr>
<td>extrapolate</td>
<td>none</td>
<td>n/a</td>
<td>124</td>
<td>0</td>
<td>4.9</td>
<td>26.2</td>
</tr>
<tr>
<td>extrapolate</td>
<td>expanding</td>
<td>$21 \times 21$</td>
<td>0</td>
<td>42</td>
<td>5.0</td>
<td>30.2</td>
</tr>
<tr>
<td>none</td>
<td>FFT</td>
<td>$41 \times 41$</td>
<td>0</td>
<td>1</td>
<td>5.6</td>
<td>33.4</td>
</tr>
<tr>
<td>extrapolate</td>
<td>hybrid</td>
<td>$5 \times 5$ and</td>
<td>0</td>
<td>12</td>
<td>5.0</td>
<td>27.1</td>
</tr>
</tbody>
</table>

**Table 3.2:** 2D DIC results for crack problem, using various starting points and coarse search algorithms, showing total number of failed points, average function evaluations per point, average BFGS iterations per point, and total time. Location of failed points is shown in Figure 3.6. Subsets that overlap crack and thus fail for all methods are excluded from count of failed points.

![Figure 3.6: Correlation points that failed to converge. DIC processes points from top of image down. Failures occur when subset minimization progression goes over crack and has no good starting point.](image)

Searching a large $41 \times 41$ area, the exhaustive search is slow, taking 111.4 seconds, or 0.068 seconds per point, as shown in Table 3.2. Since the vertical displacements are much larger than the horizontal displacements, we can search a smaller $11 \times 41$ area, cutting the time by 53% to 52 seconds. Centering the search region on a point extrapolated from neighbors allows us to shrink the search region further to $11 \times 21$, reducing the time to 37.6 seconds. Searching a smaller $5 \times 5$ area was faster still at 27.4 seconds, but had 71 points that failed because the search area was too small. Using only the extrapolated starting point, with no further coarse
search, again was fast but had 124 points that failed. These failures occurred as DIC processed correlation points across the crack, where there is no good starting point because neighboring points fail due to overlapping the crack.

The expanding search, in contrast, is able to have a large $21 \times 21$ maximum search area, but most of the time search just a small area, as seen by an average of only 42 function evaluations per point. It takes 30.2 seconds, and has no points that failed. The FFT search on a $41 \times 41$ area is 70% faster than the exhaustive search on the same area, taking 33.4 and 111.4 seconds respectively. Combining the expanding and FFT search methods into a hybrid search resulted in the fastest search at 27.1 seconds, which is 10% faster than the expanding search and 19% faster than the FFT search alone.

### 3.2 Final optimization

After an initial guess is found using a coarse search algorithm, we apply a standard nonlinear optimization method to find the minimum of the correlation objective function with sub-voxel accuracy. This also computes the derivative terms in $u$ to allow strain and rotation in the subset. Optimization techniques seek a zero of the gradient $\nabla c$, indicating a local critical point. If the Hessian matrix $H_c$ is symmetric positive definite (SPD) at the critical point, then it is a local minimum.

We provide an overview here of several optimization algorithms that have been used for DVC. None of these optimization methods guarantee convergence to the global minimum. If started too far from the global minimum, they may converge to a local minimum or may fail to converge at all. This underscores the importance of the coarse search to find an initial guess that is near the global minimum. Also, there is no property to test whether a particular minimum is the global minimum. We rely on the threshold derived in Section 3.1.3 to give us confidence that a func-
tion value is the global minimum. Points with objective function values above this threshold we mark as failed. This can occur if the initial guess is not close enough to the global minimum, the underlying material deformation is not well approximated by the shape function, or the image pattern has insufficient texture, such as in voids in the sample.

### 3.2.1 Newton’s method

Newton’s method solves a nonlinear optimization problem by using a series of quadratic optimization problems that approximate the function locally with the second-order Taylor expansion

\[
c(u + s) \approx c(u) + \nabla c(u)^T s + \frac{1}{2} s^T H_c(u) s,
\]

where \( u \) is the current point, \( s \) is the search direction to be determined, and \( H_c \) is the Hessian matrix of second derivatives. Taking the gradient of (3.1) with respect to \( s \) and finding its zero, assuming the Hessian is SPD, we find that the unique minimum of this quadratic problem occurs where

\[
H_c(u) s = -\nabla c(u).
\]

Given an initial guess \( u_0 \), Newton’s method solves (3.2) for each iterate \( u_k \) until it converges, yielding the following algorithm.

Here we have included an optional line search, which minimizes the function along the line \( s \). Close to the minimum, a line search is unnecessary because Newton’s method implicitly determines the appropriate step size as well as the direction, but when started far from the minimum, a line search improves the robustness of Newton’s method. We discuss the line search algorithm later in Section 3.2.5.
Algorithm 1 Newton’s method for optimization

**input:** function $c(u)$ and initial guess $u_0$

for $k = 0, \ldots$ do

solve $H_c(u_k)s = -\nabla c(u_k)$

$\alpha = \text{line}\_\text{search}(u_k, s)$  // optional; else $\alpha = 1$

$u_{k+1} = u_k + \alpha s$

if converged then

return $u_{k+1}$

endif

end for

Convergence is determined when either the change in function value or change in iterates is less than user-supplied tolerances,

$$|c(u_{k+1}) - c(u_k)| < \epsilon_f,$$

$$\|u_{k+1} - u_k\| < \epsilon_u.$$  

When started within its convergence region, Newton’s method normally converges quadratically, meaning the error $e_k = u_k - u^*$ satisfies

$$\lim_{k \to \infty} \|e_{k+1}\| = C \|e_k\|^2$$

for some constant $C$. However, the convergence region is often fairly small, as shown in Figure 3.7a, requiring an initial guess less than a voxel from the minimum. Newton’s method fails to converge if not started close enough to the minimum, and may also fail if the Hessian is not SPD at the current iterate.

It can be expensive to compute all the second derivatives in the Hessian. For DVC, with a subset containing $m$ points, e.g., a subset size $m = 41^3 = 68921$ points, there are $m$ spline evaluations to evaluate the objective function, $3m$ spline evaluations for the gradient, and $6m$ spline evaluations for the Hessian, for a total of $10m$ spline evaluations per iteration. There are $n = 12$ DOF for the linear shape func-
tion, so $m \gg n$, hence the $O(n^3)$ work to solve for $s$ is fairly insignificant compared to the $O(10m)$ work to evaluate the splines and build the gradient and Hessian.

### 3.2.2 BFGS

To address the unreliability and computational expense of Newton’s method, the BFGS method (named for its inventors Broyden [7], Fletcher [18], Goldfarb [29], and Shanno [72]) iteratively builds an approximation $B$ to the Hessian $H$, using only the first derivative information in $\nabla c$. As shown in Algorithm 2, at each step it applies a rank-2 update to $B$. This update guarantees that $B$ is SPD, and thus that the search direction $s$ is a descent direction. For a quadratic function, BFGS with an exact line search converges to the minimum in at most $n$ steps and $B$ converges to the true Hessian in $n$ steps [19, theorem 3.4.1]. While this result does not hold for general functions or inexact line searches, BFGS often still converges superlinearly and is widely considered one of the best general-purpose optimization methods.
Algorithm 2 BFGS

\textbf{input:} function $c(u)$ and initial guess $u_0$
\textbf{\textcolor{red}{B}}
\textbf{0} = I \quad \text{or} \quad B_0 = A^2; \text{ see Section 3.2.6}
\textbf{for} k = 0, \ldots \textbf{do}
\begin{align*}
    &\text{solve } B_k s = -\nabla c(u_k) \\
    &\alpha = \text{line search}(u_k, s) \\
    &u_{k+1} = u_k + \alpha s \\
    &\text{if converged then} \\
    &\quad \text{return } u_{k+1} \\
    &\text{end if} \\
    &y = \nabla c(u_{k+1}) - \nabla c(u_k) \\
    &d = \alpha s \\
    &B_{k+1} = B_k + yy^T - \frac{(Bd)(Bd)^T}{d^T B d}
\end{align*}
\textbf{end for}

The convergence region for BFGS is much larger than that for Newton’s method, as shown in Figure 3.7b, allowing the initial guess to be up to several voxels from the minimum, but BFGS usually takes more iterations than Newton’s method to meet the same convergence tolerance. We also implement a hybrid method, where we first use Newton’s method, but if it fails, fall back to using BFGS. This combines the speed of Newton’s method when it converges with the robustness of BFGS.

The $O(n^3)$ cost of solving for $s$ can be reduced to $O(n^2)$ by updating a factorization of $B$ instead of $B$ itself [61]. For DVC there are $4m$ spline evaluations per iteration to compute the objective function and its gradient, which again dominates the computation since $m \gg n$. The line search incurs additional function and gradient evaluations.

3.2.3 Gauss-Newton and Levenberg-Marquardt

Nonlinear least-squares functions have additional structure that can be exploited in the optimization process. We let $r(u)$ be the vector of residuals $r_i(u) = f_i - g_i(u)$,
with \( i = 1, \ldots, m \) indexing all points in a subset. Then the least-squares correlation function (1.3) can be restated as

\[
c(u) = F \sum_{i=1}^{m} (f_i - g_i(u))^2 = Fr^T r,
\]

with gradient and Hessian

\[
\nabla c(u) = 2F J_r^T r,
\]
\[
H_c(u) = 2F (J_r^T J_r + \sum_i r_i H_{r_i}),
\]

where \( J_r \) is the Jacobian of \( r \), \( H_{r_i} \) is the Hessian of \( r_i \), and \( F = 1/\|f\|^2 \) is a normalizing constant. Since the \( r_i \) are expected to be small near the minimum, the Gauss-Newton method drops the second derivative terms \( \sum_i r_i H_{r_i} \). This may lead to an ill-conditioned problem, so the Levenberg-Marquardt method \([43, 49]\) instead replaces \( \sum_i r_i H_{r_i} \) with \( \mu I \) for some well-chosen scalar \( \mu \). The system (3.2) defining the search direction thus becomes

\[
(J_r^T J_r + \mu I)s = -J_r^T r,
\]

which we recognize as the normal equations for the linear least-squares problem

\[
\begin{bmatrix}
J_r \\
\sqrt{\mu} I
\end{bmatrix} s \cong \begin{bmatrix}
-r \\
0
\end{bmatrix}.
\]

This leads to Algorithm 3. It can be viewed as finding the minimum of the quadratic model subject to a constraint that \( \|s\| \leq h \) for some radius \( h \), within which we trust that the quadratic model approximates our problem well. Hence the name trust region or restricted step for this type of method. For the selection of \( h \) or \( \mu \), we refer
One advantage of Levenberg-Marquardt is that it does not require a line search, reducing the number of function evaluations per iteration. A disadvantage of this formulation is that, since for DVC $m \gg n$, the $m \times n$ Jacobian $J_r$ is much larger than the $n \times n$ Hessian $H_c$, requiring more memory and $O(mn^2)$ computation to solve using a Householder QR factorization, as compared to $O(n^3)$ or $O(n^2)$ for Newton’s method or BFGS, respectively. Also, as stated, it is applicable to only least-squares problems, so cannot be used to optimize the cross-correlation function. For general nonlinear problems, such as the cross-correlation function, the full Hessian $H_c$ can be used in place of $J_r^T J_r$ [19, 30]. We use the Levenberg-Marquardt implementation in MINPACK [52], which includes a feature for scaling variables to improve the conditioning of the problem.

### 3.2.4 Steepest Descent

Another method which we include for completeness is steepest descent, sometimes called gradient descent. It is motivated by the fact that a function is decreasing fastest in the negative gradient direction, so it chooses the search direction $s = -\nabla c(u_k)$. This essentially replaces the Hessian $H_c$ with the identity matrix. While reliable, since the search direction is always a descent direction, it unfortunately
converges only linearly. The convergence for the quadratic model problem is

\[
\lim_{k \to \infty} \|e_{k+1}\| = \frac{\lambda_{\text{max}} - \lambda_{\text{min}}}{\lambda_{\text{max}} + \lambda_{\text{min}}} \|e_k\|,
\]

where \(\lambda_{\text{max}}\) and \(\lambda_{\text{min}}\) are the maximum and minimum eigenvalues of the Hessian matrix [6]. If the problem is well conditioned, i.e., the ratio \(\lambda_{\text{max}}/\lambda_{\text{min}}\) is near 1, then steepest descent performs well, but for ill-conditioned problems it converges very slowly.

For DVC, the cost of steepest descent is \(4m\) spline evaluations to compute the correlation function and its gradient, as with BFGS. While steepest descent avoids \(O(n^2)\) work in updating and solving \(B\), this is an insignificant amount of work for DVC, so we recommend BFGS over steepest descent due to its faster convergence.

---

**Algorithm 4 Steepest descent**

```
input: function \(c(u)\) and initial guess \(u_0\)
for \(k = 0, \ldots\) do
    \(s = -\nabla c(u_k)\) (\(or s = -A^2\nabla c\); see Section 3.2.6)
    \(a = \text{line-search}(u_k, s)\)
    \(u_{k+1} = u_k + as\)
    if converged then
        return \(u_{k+1}\)
    end if
end for
```

---

### 3.2.5 Line search

BFGS and steepest descent determine a search direction \(s\), but not the step size to take in that direction. To determine this step size, a line search performs a 1D optimization of the function along the search direction \(s\). Finding the exact minimum is expensive, so we instead use an inexact line search with the lesser goal of finding a point for which the function decrease is commensurate with the step size. We use a cubic line search as given by Fletcher [19, p.33], which we briefly summarize here.
Let \( c(a) = c(u + \alpha s) \) be the function to minimize and \( \bar{a} \) be the smallest positive \( \alpha \) such that \( c(\bar{a}) = c(0) \). As \( a \to 0 \) or \( \alpha \to \bar{a} \), the function \( c(\alpha) \) is reduced by a negligible amount, so we must exclude those areas. Goldstein [31] and Wolfe [77] suggest two conditions to achieve this, which are illustrated in Figure 3.8. Applying the condition

\[
c(\alpha) \leq c(0) + \rho c'(0)\alpha,
\]

for some constant \( \rho \in (0, \frac{1}{2}) \), ensures that \( \alpha \) is not too close to \( \bar{a} \). To satisfy this condition, \( c(\alpha) \) must be below the \( \rho \) line in Figure 3.8. Applying the condition

\[
c'(\alpha) \geq \sigma c'(0),
\]

for some constant \( \sigma \in (\rho, 1) \), ensures that \( \alpha \) is not too close to 0. To satisfy this condition, the derivative must be greater than the slope of the \( \sigma \) line in Figure 3.8. The line search proceeds in two phases. First, it finds a bracket \([\alpha_1, \alpha_2]\) that is guaranteed to contain an \( \alpha \) satisfying (3.3) and (3.4). It then finds an \( \alpha \) that minimizes the cubic polynomial interpolating \( c(\alpha_1), c'(\alpha_1), c(\alpha_2), c'(\alpha_2) \) and replaces one of \( \alpha_1 \) or \( \alpha_2 \) with \( \alpha \) to shrink the bracket. This process is repeated until it finds an \( \alpha \) satisfying (3.3) and (3.4).

### 3.2.6 Scaling

As discussed earlier for steepest descent, the conditioning of the optimization problem affects the convergence rate. Ill-conditioning may be the result of poor scaling between the variables. With the linear shape function (1.1), DVC computes two kinds of variables: displacements \( u, v, w \) and their derivatives \( \frac{\partial u}{\partial x}, \ldots, \frac{\partial u}{\partial z} \). With the quadratic shape function (2.1), DVC adds a third kind, the second derivatives. Dis-
placements are typically on the order of several voxels, while derivatives are typically less than a few percent, and second derivatives are in units of voxels$^{-1}$. Thus, the values may differ by several orders of magnitude. This difference in scaling between the variables causes an artificial ill-conditioning that can negatively affect the convergence of optimization methods. However, this poor scaling can be remedied by applying a linear transformation of the variables.

We define the new vector $\hat{u}$ by an affine transformation, $\hat{u} = Au + b$. For a function $c(u)$, we define the function with respect to the new vector as $\hat{c}(\hat{u}) = c(A^{-1}(\hat{u} - b))$. By the chain rule, the gradient and Hessian are

$$\nabla c = A^T \nabla \hat{c},$$

$$H_c = A^T \hat{H}_c A,$$

where $\nabla \hat{c}$ and $\hat{H}_c$ are with respect to the new vector $\hat{u}$.

**Theorem.** For a Newton-like iteration $u_{k+1} = u_k - B_k^{-1} \nabla c(u_k)$, if

$$B_k = A^T \hat{B}_k A$$

*Figure 3.8:* Line search of $c(\alpha)$ showing $\alpha$ that satisfy Goldstein-Wolfe conditions.
for all \( k \), then the method applied to \( c(u) \) and \( \hat{c}(\hat{u}) \) produces the sequence of iterates \( \{u_0, \ldots, u_k\} \) in the original variables and the iterates \( \{\hat{u}_0, \ldots, \hat{u}_k\} \) in the transformed variables, with \( \hat{u}_k = Au_k + b \) relating the two sequences.

**Proof.** The proof follows Fletcher [19, Theorem 3.3.1]. Assume \( \hat{u}_k = Au_k + b \). This is true for \( k = 0 \) by choice of the initial guess \( \hat{u}_0 \). Now we show it holds for \( \hat{u}_{k+1} \):

\[
\hat{u}_{k+1} = \hat{u}_k - \hat{B}_k^{-1}\hat{\nabla}_k
\]

\[
= (Au_k + b) - (A^{-T}B_kA^{-1})^{-1}(A^{-T}\nabla c_k)
\]

\[
= A(u_k - B_k^{-1}\nabla c_k) + b
\]

\[
= Au_{k+1} + b,
\]

where we applied the inductive hypothesis, (3.5), and (3.7).

The significance of this theorem is that, if (3.7) holds, we can compute the optimization using the original vector \( u \) with the same convergence rate as the transformed vector \( \hat{u} \). We say a method is *invariant* with respect to affine transformations if (3.7) holds for all affine transformations. An invariant method is not affected by poor scaling because it implicitly changes to the transformed variables \( \hat{u} \) with the ideal \( A \) defined implicitly by the method. For Newton’s method, \( B_k \) is the Hessian \( H_c \), which satisfies (3.7) for all affine transformations by the chain rule (3.6), so it is invariant.

BFGS started with \( B_0 = I \) is not invariant, but if the initial \( B_0 \) satisfies (3.7), then every subsequent \( B_k \) also satisfies (3.7). This has two implications. First, if we start with the exact Hessian, \( B_0 = H_c(u_0) \), which by the chain rule satisfies (3.7) for all affine transformations, then BFGS is invariant. Second, if we instead start with \( B_0 = A^TIA \) for a specific \( A \), then the transformation \( A \) is implicitly propagated through the computation. In this case, BFGS computes \( \{u_0, \ldots, u_k\} \), which
is implicitly the same sequence in the transformed variables, \( \{ \hat{u}_0, \ldots \hat{u}_k \} \), as would be computed for \( \hat{c} \) starting with \( \hat{B}_0 = I \). Hence, we do not need to scale variables explicitly, but simply replace \( I \) with a suitable matrix \( A^T A \). Similarly, in steepest descent and Levenberg-Marquardt, replacing \( I \) with \( A^T A \) is sufficient. However, MINPACK’s implementation of Levenberg-Marquardt includes a feature to scale variables automatically, so we do not explicitly set the scaling. The goal, then, is to choose \( A \) such that the conditioning of the problem in the new variables \( \hat{u} \) is better, and hence convergence is faster. Typically \( A \) is chosen to be diagonal, such that \( A^2 \approx H_c \).

For DVC, by examining the diagonal of the Hessian at the minimum for various problems, we found that a good choice is to set \( B_0 \) with diagonal entries \( B_{ii} = 0.01 \) corresponding to the displacement variables \( u, v, w \); \( B_{ii} = 1 \) corresponding to first derivative variables; and \( B_{ii} = 100 \) corresponding to second derivative variables. For instance, if variables are ordered

\[
\mathbf{u} = \left[ u \ v \ w \ \frac{\partial u}{\partial x} \ \cdots \ \frac{\partial w}{\partial x} \ \frac{\partial^2 u}{\partial x^2} \ \cdots \ \frac{\partial^2 w}{\partial y \partial z} \right]^T,
\]

then we set

\[
B_0 = \begin{bmatrix}
0.01 & 0.01 & 0.01 & 1 & \cdots & 1 & 100 \\
& 0.01 & 1 & \cdots & 1 & 100 \\
& & \ddots & \ddots & \ddots \\
& & & \ddots & \ddots & \ddots \\
& & & & \ddots & \ddots & \ddots \\
& & & & & \ddots & \ddots & \ddots \\
& & & & & & 100 & \\
\end{bmatrix}.
\]
3.2.7 Comparison of Methods

We compare the performance of various optimization methods using the spherical inclusion problem used earlier in Section 3.1.5. To ensure a fair comparison, we use the same initial guess for all methods, as computed previously by the extrapolated starting point and expanding coarse search. This also excludes the coarse search time, since we read the initial guess from a file. As shown in Table 3.3, Newton’s method is the fastest method at 0.26 seconds per point, but has a significant number of points that fail to converge. Adding a cubic line search improves the robustness, reducing the number of failed points from 936 to 620 out of 2744. For points where Newton’s method fails, we can fall back to using BFGS. This hybrid method eliminates the failed points, but despite taking fewer iterations, is about the same performance as BFGS with scaling, taking 0.43 seconds per point.

BFGS with no line search takes the same time as Newton’s method with a line search, 0.32 seconds per point, and reduces the number of failed points to 10. Adding a cubic line search, as is common with BFGS, eliminated all the failed points, but doubled the time to 0.66 seconds per point. Scaling the variables reduces BFGS from 9.3 to 5.3 iterations, reducing the time by 38% from 0.66 to 0.41 seconds per point. For a similar number of iterations, Levenberg-Marquardt is somewhat more expensive, taking 0.83 seconds per point compared to 0.66 for BFGS (with no scaling). With scaling, BFGS was twice as fast as Levenberg-Marquardt. Initially, steepest descent was quite slow, taking 48.9 iterations and 3.01 seconds per point. Appropriately scaling the variables dramatically reduced this to 6.7 iterations and 0.50 seconds per point, but it was still 20% slower than BFGS with scaling.
<table>
<thead>
<tr>
<th>Method</th>
<th>Line search</th>
<th>Failed points</th>
<th>Iterations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton</td>
<td>none</td>
<td>936</td>
<td>2.8</td>
<td>0.26</td>
</tr>
<tr>
<td>Newton</td>
<td>cubic</td>
<td>620</td>
<td>2.6</td>
<td>0.32</td>
</tr>
<tr>
<td>Newton + BFGS</td>
<td>cubic</td>
<td>0</td>
<td>4.0</td>
<td>0.43</td>
</tr>
<tr>
<td>BFGS</td>
<td>none</td>
<td>10</td>
<td>11.6</td>
<td>0.32</td>
</tr>
<tr>
<td>BFGS</td>
<td>cubic</td>
<td>0</td>
<td>9.3</td>
<td>0.66</td>
</tr>
<tr>
<td>BFGS, scaled</td>
<td>cubic</td>
<td>0</td>
<td>5.3</td>
<td>0.41</td>
</tr>
<tr>
<td>Levenberg-Marquardt</td>
<td>n/a</td>
<td>0</td>
<td>9.8</td>
<td>0.83</td>
</tr>
<tr>
<td>Steepest descent</td>
<td>cubic</td>
<td>0</td>
<td>48.9</td>
<td>3.01</td>
</tr>
<tr>
<td>Steepest descent, scaled</td>
<td>cubic</td>
<td>12</td>
<td>6.7</td>
<td>0.50</td>
</tr>
</tbody>
</table>

**Table 3.3:** DVC results using various optimization methods, showing total number of failed points, average iterations and average time per correlation point. Initial guess from expanding search with extrapolated starting point is used for all tests; coarse search time is not included here.

### 3.2.8 Summary

To summarize our results related to the optimization algorithms, we made several refinements to improve their speed and efficiency for application to digital volume correlation. We improved the coarse search by using previously computed correlation points to extrapolate an initial guess and use an expanding search to minimize the region to search. This is effective for a small region, but to search a large region efficiently, we switch to using an FFT-based coarse search, which improves the robustness of the coarse search. For the subsequent sub-voxel final optimization, we found BFGS with diagonal scaling to be efficient and reliable. Levenberg-Marquardt and steepest descent were also reliable but not as efficient in our tests.
Chapter 4

Interpolation and Approximation

The core computation of DVC is evaluating the correlation function and its derivatives, which in turn requires interpolating $g(\hat{x})$ between voxels. We use cubic spline interpolation [69], which provides the $C^2$ differentiability required for Newton’s method. Dierckx [16] and de Boor [15] provide good overviews of spline algorithms. Here we summarize the different schemes for constructing and evaluating splines, comparing their computational complexity and memory requirements, both of which are especially important in 3D DVC. We start with a description of one-dimensional splines, which we extend to multiple dimensions using a tensor product in Section 4.4. This chapter was previously published in condensed form in Gates et al. [25].

4.1 Basis functions

A cubic spline $\phi(x)$ is a piecewise polynomial of degree three that is $C^2$ differentiable. Let $x_0, \ldots, x_n$ be knots (data points) with associated data values $f_0, \ldots, f_n$. Adjacent knots define an interval in which the spline is a polynomial. The spline may have jump discontinuities in the third derivative at knots. Let $x \in [x_i, x_{i+1})$ be
a point in the $i$th interval, and $\tilde{x} = (x - x_i)/(x_{i+1} - x_i)$ be the local coordinate in the reference interval $[0, 1]$.

Three different sets of basis functions are commonly used for splines: the monomial basis, Hermite basis, and B-spline basis. Which basis is most efficient depends on the particular computer hardware, including the size of caches and relative speed of floating-point computations to memory bandwidth. Within each interval, a cubic polynomial requires four coefficients. For the monomial basis, shown in Figure 4.1, the polynomial in each interval is represented separately as

$$\phi(x) = \sum_{r=0}^{3} a_r \tilde{x}^r = a_0 + a_1 (\tilde{x} + a_2 (\tilde{x} + a_3 \tilde{x})),$$

with coefficients $a_0, \ldots, a_3$. The monomial basis requires the least computation of the spline representations, but the most memory. Using the monomial basis, a spline with $n + 1$ knots in each dimension requires $4n$ coefficients in 1D, $16n^2$ coefficients in 2D, and $64n^3$ coefficients in 3D.

Hermite basis functions, shown in Figure 4.2, have the property that either the
Figure 4.2: Hermite basis functions. Dashed red lines indicate derivative at endpoint. For each basis function, one of four function and derivative values at endpoints is one: $h_{00}(0) = 1$, $\frac{d}{dx}h_{01}(0) = 1$, $h_{10}(1) = 1$, $\frac{d}{dx}h_{11}(1) = 1$. Other three function and derivative values at endpoints are zero.
function or its derivative is 1 at one endpoint, and the other three function and 
derivative values at the endpoints are 0. The spline coefficients are simply the func-
tion and derivative values at each node,

$$\phi(x) = f_i h_{00}(\tilde{x}) + f'_i h_{01}(\tilde{x}) + f_{i+1} h_{10}(\tilde{x}) + f'_{i+1} h_{11}(\tilde{x}) = \sum_{r=0}^{1} \sum_{a=0}^{1} \beta_{i+r,a} h_{ra}(\tilde{x}),$$

where

$$h_{00}(x) = 2x^3 - 3x^2 + 1,$$
$$h_{01}(x) = x^3 - 2x^2 + x,$$
$$h_{10}(x) = -2x^3 + 3x^2,$$
$$h_{11}(x) = x^3 - x^2,$$
$$\beta_{i,a} = \frac{\partial^a}{\partial x^a} f_i.$$

Each interval shares function and derivative values with neighboring intervals, so it requires fewer total coefficients than the monomial basis, $2(n + 1)$ in 1D, $4(n + 1)^2$ in 2D, and $8(n + 1)^3$ in 3D.

Cubic B-spline basis functions, shown in Figure 4.3, are each $C^2$ continuous 
across the entire domain and have support over four intervals, so each coefficient 
is shared across four intervals. Thus it requires the fewest coefficients, $n + 3$ in 
1D, $(n + 3)^2$ in 2D, and $(n + 3)^3$ in 3D, but computing the basis functions is more 
expensive than computing the Hermite basis functions. The spline is evaluated as

$$\phi(x) = \sum_{r=i-3}^{i} c_r N_{r,4}(x),$$
with coefficients $c_r$ and B-spline basis functions defined recursively by

$$N_{i,d+1}(x) = \frac{x - x_i}{x_{i+d} - x_i} N_{i,d}(x) + \frac{x_{i+d+1} - x}{x_{i+d+1} - x_{i+1}} N_{i+1,d}(x) \quad \text{for } d > 1,$$

$$N_{i,1}(x) = \begin{cases} 1, & \text{if } x \in [x_i, x_{i+1}), \\ 0, & \text{otherwise}. \end{cases}$$

This recursive definition can be implemented in a triangular scheme, shown in Figure 4.4. Each column depends on the entries in the previous column. The computation can be done in a single vector of length $d + 1$, representing a single column, by computing entries in each column from the top down, overwriting entries from
Figure 4.4: Triangular scheme for computing basis functions. Dependencies are shown by arrows. First three columns of derivative (b) are already computed in (a). By computing from top down, each column overwrites previous column.

the previous column. The derivative of the basis functions is given recursively by

\[
N_{i,d+1}' = d \left( \frac{N_{i,d}(x)}{x_{i+d} - x_i} - \frac{N_{i+1,d}(x)}{x_{i+d+1} - x_{i+1}} \right).
\]

Notice that this depends on the order \(d\) basis functions, that is, column 3 in Figure 4.4a since cubic splines have degree \(d = 3\). Hence, when both the value and the derivative of the spline are required, it is advantageous to evaluate them simultaneously to eliminate the redundant computation in the first three columns of Figure 4.4b. Evaluating the spline and its derivative simultaneously also improves cache performance since both will access the same spline coefficients. Therefore, we evaluate the correlation function and its gradient together in one function call.

We optimize these formulas by assuming data at fixed width voxels, so the denominators can all be pre-computed. This eliminates divisions, which are an order-of-magnitude slower than multiplications, and reduces the overall number of floating-point operations, resulting in a 3.5 times speedup in evaluating all points in a subset, compared to a general purpose implementation.

We summarize in Table 4.1 the relative complexity and speed of the various spline representations to evaluate all the points in a subset for three DOF (i.e., displacements only), six DOF (displacements and axial strains), and general deformations (linear and quadratic shape functions). For three DOF, the basis functions are
Table 4.1: Floating-point operations (flops) per spline evaluation, time to evaluate all points in subset, theoretical model time, and memory requirements for various spline representations of $50^3$ subset. Bold entries indicate fastest time in row. Evaluations with translations (three DOF) and translations and axial strains (six DOF) can be optimized because grid is axis-aligned. Times in ms on 1.25 GHz PowerPC G4.

<table>
<thead>
<tr>
<th></th>
<th>Monomial basis</th>
<th>Hermite basis</th>
<th>B-spline basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>149 ms</td>
<td>54 ms</td>
<td>31 ms</td>
</tr>
<tr>
<td>model time</td>
<td>127 ms</td>
<td>33 ms</td>
<td>20 ms</td>
</tr>
<tr>
<td>flops/point</td>
<td>126 flops</td>
<td>147 flops</td>
<td>147 flops</td>
</tr>
<tr>
<td>Translations and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>axial strains</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>148 ms</td>
<td>60 ms</td>
<td>38 ms</td>
</tr>
<tr>
<td>model time</td>
<td>127 ms</td>
<td>34 ms</td>
<td>24 ms</td>
</tr>
<tr>
<td>flops/point</td>
<td>127 flops</td>
<td>160 flops</td>
<td>175 flops</td>
</tr>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>147 ms</td>
<td>82 ms</td>
<td>59 ms</td>
</tr>
<tr>
<td>model time</td>
<td>127 ms</td>
<td>38 ms</td>
<td>31 ms</td>
</tr>
<tr>
<td>flops/point</td>
<td>129 flops</td>
<td>186 flops</td>
<td>231 flops</td>
</tr>
<tr>
<td>memory</td>
<td>62500 KB</td>
<td>8291 KB</td>
<td>1163 KB</td>
</tr>
</tbody>
</table>

evaluated at the same point within every voxel, so can be evaluated just once at the beginning of evaluating a subset, creating a 47% improvement in time to evaluate a subset with the B-spline basis, from 59 ms with a general deformation to 31 ms with three DOF. The Hermite basis exhibits a smaller but significant 34% savings, from 82 ms to 54 ms. The three DOF case is important for use with the coarse search. For six DOF, the grid of points is axis-aligned; along a grid line parallel to the x axis, the y and z basis functions are constant, so can be evaluated just once per grid line. The six DOF case is useful in computing displacements and axial strains, either if rotations and shear strains are known to be negligible or as an intermediate step before doing a full minimization with linear shape functions. Again, there is a substantial 35% savings from 59 ms with a general deformation to 38 ms with six DOF for the B-spline basis. With the Hermite basis there is a 25% savings from 82 ms to 60 ms. Because the monomial basis is evaluated efficiently using Horner’s rule instead of explicitly computing the basis functions, these optimizations do not apply, so we see a fairly constant time of 147 ms.
Despite doing more computation, the smaller memory requirements of the B-spline basis, which uses 1163 KB of memory for the subset’s coefficients, cause it to be faster on this processor than the Hermite basis, which uses 8291 KB. Evaluation using the monomial basis is memory bandwidth limited, as it must transfer 62500 KB for the subset’s coefficients, so achieves poor performance. Each coefficient is used only once, so there is poor reuse of cache memory, as compared with the B-spline basis where each coefficient is reused up to 64 times. The coefficients for the B-spline basis are small enough to fit in cache, so can also avoid cache misses the next time the subset is evaluated.

The “model” times listed in Table 4.1 use a simple theoretical model to predict the evaluation time based on the number of floating point operations and the amount of memory transferred as

\[
\text{model time} = \frac{\text{total flops}}{\text{processor speed}} + \frac{\text{memory}}{\text{bandwidth}}.
\]

In this case, the processor speed is 1 Gflops/sec and the memory bandwidth is 550 MB/sec, as measured by the STREAM benchmark [50]. This model is useful to gauge how an algorithm can be expected to perform on given computer hardware. For instance, here the monomial basis might be improved, but cannot be expected to run faster than the model time of 127 ms due to memory bandwidth limitations. For the B-spline basis, there is also potential for improvement from the currently achieved performance of 59 ms to the theoretical model performance of 31 ms for twelve DOF. Similarly, the Hermite basis has potential to improve from 82 ms to 38 ms. However, as this simple model does not take into account many features of the computer architecture, such as instruction dependencies, being within a factor of two of the theoretical speed indicates our implementation is achieving reasonable performance. Note that the three spline bases represent the same interpolat-
ing spline function, and differ solely in their computational time performance and memory usage.

### 4.2 Interpolating splines

For an interpolating spline, \( \phi(x_i) = f_i \) at all knots \( x_i \) for \( i = 0, \ldots, n \). That is, the spline is constrained to pass through all data points. Appropriate boundary conditions must be applied to make the spline unique. We set the value of the first derivative at the boundary, which can be computed using a finite difference along the boundary of the subset.

To compute a spline with the Hermite basis requires solving the tridiagonal system [3]

\[
\begin{bmatrix}
1 & 4 & 1 \\
1 & 4 & 1 \\
\vdots & \vdots & \vdots \\
1 & 4 & 1 \\
1 & & \\
& 1 & \\
\end{bmatrix}
\begin{bmatrix}
f'_0 \\
f'_1 \\
\vdots \\
f'_{n-1} \\
f'_n \\
\end{bmatrix}
=
\begin{bmatrix}
f'_0 \\
3(f_2 - f_0) \\
\vdots \\
3(f_n - f_{n-2}) \\
f'_n \\
\end{bmatrix}.
\]

We use the LAPACK [2] tridiagonal solver to solve this system efficiently in \( O(n) \) time. To use the monomial basis, the remaining coefficients can be computed from the function and derivative values [15, p. 54–59].

The B-spline basis requires adding three boundary knots on each end of the domain. To maintain the assumption of intervals with a fixed unit width, we choose the boundary knots

\[
x_{-3} = x_0 - 3, \quad x_{-2} = x_0 - 2, \quad x_{-1} = x_0 - 1,
\]

\[
x_{n+1} = x_n + 1, \quad x_{n+2} = x_n + 2, \quad x_{n+3} = x_n + 3,
\]
rather than the more commonly used coincident boundary knots. Coefficients are computed by solving

\[
\begin{bmatrix}
N_{-3}'(x_0) & N_{-2}'(x_0) & N_{-1}'(x_0) & N_0'(x_0) \\
N_{-3}(x_0) & N_{-2}(x_0) & N_{-1}(x_0) & N_0(x_0) \\
\vdots & \vdots & \vdots & \vdots \\
N_{n-4}'(x_n) & N_{n-3}'(x_n) & N_{n-2}'(x_n) & N_{n-1}'(x_n) \\
N_{n-4}(x_n) & N_{n-3}(x_n) & N_{n-2}(x_n) & N_{n-1}(x_n)
\end{bmatrix}
\begin{bmatrix}
c_{-3} \\
c_{-2} \\
\vdots \\
c_{n-2} \\
c_{n-1}
\end{bmatrix}
= \begin{bmatrix}
f_0' \\
f_0 \\
\vdots \\
f_n' \\
f_n
\end{bmatrix},
\]

or

\[Ec = f. \quad (4.1)\]

Because of the finite support of the basis functions, this is a banded system and can be efficiently solved with a banded solver for a large image. However, as discussed later in Section 5.2, we compute splines over each subset rather than the whole 3D image. In this case, since the size of the matrix is modest, say, \(n = 31\), we found that using a dense solver was actually faster than using a banded solver.

### 4.3 Smoothing splines

When the data \(f_i\) are inherently noisy, exactly interpolating the data is not appropriate, because the interpolant would mimic the noise as well as the underlying signal. It is often better to use a smoothing spline, which specifies a tolerance for how close the spline is to the data, based on an error bound or standard deviation of errors. A cubic interpolating spline minimizes the integral of the second derivative, subject to interpolating the data. A cubic smoothing spline, in contrast, minimizes jumps in the third derivative, subject to the discrepancy at data points being less
than a smoothing parameter $S$,

$$\text{minimize } \sum_i \left( \phi^{(3)}(x^+_i) - \phi^{(3)}(x^-_i) \right)^2$$

subject to $\sum_i (f_i - \phi(x_i))^2 \leq S$. \hfill (4.2)

Reinsch [63] recommends picking $S$ in the range $\delta^2(n \pm \sqrt{2n})$, where $n$ is the number of voxels in the image and $\delta$ is an estimate of the error in intensity at each voxel. For convenience in using a parameter independent of the image size, we pick $S = \delta^2 n$ and use $\delta$ as a parameter that controls the amount of smoothing. This parameter can be used to some extent to compensate for the noise inherent in the tomography images. However, in 2D DIC applications such correction may not be as necessary since modern CCD cameras tend to have lower noise levels than X-ray CT scanners. We again use the B-spline basis, as it is computationally the fastest, and solve (4.2) using a least-squares system, as in Dierckx [16].

To solve (4.2), we introduce a Lagrange multiplier $p$, resulting in the least squares system

$$\begin{bmatrix} E & 1 \\ \frac{1}{\sqrt{p}} H \end{bmatrix} c \cong \begin{bmatrix} f \\ 0 \end{bmatrix},$$

with

$$H = \begin{bmatrix} a_{-3,1} & \cdots & a_{n-1,1} \\ \vdots & \ddots & \vdots \\ a_{-3,n-1} & \cdots & a_{n-1,n-1} \end{bmatrix},$$

$E$, $f$, and $c$ as defined in (4.1), and $a_{i,j}$ being jumps in the third derivative at interior knots [16, ch. 5]. This system can be solved using either a QR factorization or the
normal equations,
\[(E^T E + \frac{1}{p} H^T H) c = E^T f.\] (4.4)

The QR approach has the advantage of being more stable, but is more expensive. The normal equations approach generates a matrix with a bandwidth of nine for cubic splines, so can be solved efficiently with a banded solver. However, it squares the condition number, so is a poor choice if the matrix is not well conditioned. For a linear system with condition number \(\kappa\), the solution loses \(\log_{10}(\kappa)\) digits of accuracy. Double precision arithmetic has approximately 16 digits of precision, while single precision arithmetic has approximately 7 digits of precision. For a moderate amount of smoothing with \(\delta = 4\), we found the condition number for the normal equations (4.4) is less than 10, which is quite good. A larger amount of smoothing with \(\delta = 12\) resulted in a condition number of \(5 \times 10^5\). For double precision this leaves 10 digits of accuracy, which is still acceptable. For single precision it leaves only one digit of accuracy, so a large amount of smoothing requires double precision. In addition to depending on the smoothing parameter \(\delta\), the conditioning also depends on the image.

The multiplier \(p\) must be determined so that \(F(p) \equiv \sum(f_i - \phi(p)(x_i))^2 \leq S\). We use the rational interpolation scheme developed by Dierckx [16] to solve for \(p\). As discussed later in Section 5.2, we compute splines over each subset, rather than over the entire image. We found that the \(p\) value for different subsets was similar, so using the \(p\) from a previous subset as an initial guess improved the convergence of the method to determine \(p\). This optimization improved overall performance of the code by 15–30%, as shown in Table 4.2. For this example, the optimized version with \(\delta = 5\) reduced the time from 0.90 to 0.68 seconds per point.
4.4 Tensor-product splines

The previous discussion in Sections 4.1–4.3 was for the 1D case. We have implemented both interpolating and smoothing splines in a DVC code in three dimensions using a tensor product of 1D splines. For a 3D tricubic spline, we let \( \{ (x_i, y_j, z_k); i = 0, \ldots, l; j = 0, \ldots, m; k = 0, \ldots, n \} \) be knots with associated data values \( f_{ijk} \). At a point

\[
x \in [x_i, x_{i+1}] \times [y_j, y_{j+1}] \times [z_k, z_{k+1}]
\]

with local coordinates

\[
\tilde{x} = (x - x_i)/(x_{i+1} - x_i),
\]

\[
\tilde{y} = (y - y_j)/(y_{j+1} - y_j),
\]

\[
\tilde{z} = (z - z_k)/(z_{k+1} - z_k),
\]

a tricubic spline is represented as

\[
\phi(x, y, z) = \sum_{r=0}^{3} \sum_{s=0}^{3} \sum_{t=0}^{3} \alpha_{rst} \tilde{x}^r \tilde{y}^s \tilde{z}^t
\]

\[
= \sum_{r=0}^{1} \sum_{a=0}^{1} \sum_{s=0}^{1} \sum_{b=0}^{1} \sum_{t=0}^{1} \sum_{c=0}^{1} \beta_{i+r,j+s,k+t,a,b,c} h_{ra}(\tilde{x}) h_{sb}(\tilde{y}) h_{tc}(\tilde{z})
\]

\[
= \sum_{r=-3}^{i} \sum_{s=-3}^{j} \sum_{t=-3}^{k} c_{rst} N_{xr}(x) N_{ys}(y) N_{zt}(z)
\]

in the monomial, Hermite, and B-spline bases, respectively. The coefficients for the Hermite basis are the function value and partial derivatives at each node,

\[
\beta_{ijkabc} = \frac{\partial^{a+b+c}}{\partial x^a \partial y^b \partial z^c} f_{ijk},
\]
which are computed to enforce continuity of the spline at voxel boundaries [14].

To compute the coefficients for B-splines in 3D, we solve

\[ A_{ir} A_{js} A_{kt} C_{rst} = F_{ijk}, \]

where \( A_{ir}, A_{js}, A_{kt} \) are the 1D interpolation matrices from (4.1) or (4.3) in the \( x \), \( y \), and \( z \) directions, \( C \) is the tensor of B-spline coefficients, and \( F \) is the tensor of data values. As in 1D, the boundaries of \( F \) contain derivatives computed by finite differences. Solving this system involves solving in the \( x \) dimension, cyclically permuting \( ijk \) indices to \( jki \), solving in the \( y \) dimension, cyclically permuting to \( kij \), solving in the \( z \) dimension, and cyclically permuting back to \( ijk \) [9].

### 4.5 Comparison of methods

Previous DVC algorithms used \( C^0 \) trilinear interpolation [28, 42] or \( C^1 \) tricubic interpolation [5, 73, 79], which is easily implemented using the Hermite basis with derivatives computed by finite differences of the 3D image data. These derivative values will be inherently noisy due to the noise in the images, creating a noisy correlation function. Figure 4.5 shows 2D slices of correlation functions, where we vary \((u, v)\) and hold the other ten DOF fixed. The grid lines indicate integer voxel deformations. The correlation function with \( C^1 \) tricubic interpolation (Figure 4.5a) exhibits scallops between integer deformations, leading to many local minima. Interpolating splines smooth the derivatives by enforcing second derivative continuity, resulting in a smoother correlation function with fewer local minima (Figure 4.5b), which is therefore easier to minimize. Smoothing splines yield an even smoother correlation function (Figure 4.5c), resulting in improvements in convergence and accuracy. Unfortunately, computing the smoothing spline is more expensive than
computing an interpolating spline, but this may be offset by requiring fewer iterations in the subsequent optimization algorithm. Conceptually, smoothing splines perform a similar function to the image filtering suggested by Schreier [71], which smooths the image data prior to performing DIC using a low-pass filter. However, smoothing splines offer a tunable parameter \( \delta \) to adjust the amount of smoothing at run time.

For the spherical inclusion problem with a PDMS sample scanned by the Skyscan, using \( C^1 \) tricubic interpolation required an average of 18.2 BFGS iterations per correlation point and had a relative error of 4.11%; tricubic spline interpolation required 15.3 iterations, a 16% improvement, and reduced the error to 3.63%; smoothing spline approximation required 13.3 iterations, an additional 11% improvement, with an error of 3.57%.

With the ceramic foam sample scanned by the Xradia scanner, which has lower noise than the Skyscan scanner, the improvements were more modest. As shown in Table 4.2, using smoothing splines resulted in a 10% fewer iterations, from 9.47 iterations with an interpolating spline \((\delta = 0)\) down to 8.51 iterations with \(\delta = 5\). With the optimized version, which used previous \( p \) values as the initial guess for
<table>
<thead>
<tr>
<th>Noise ($\delta$)</th>
<th><strong>BFGS</strong> iterations</th>
<th><strong>Unoptimized time</strong> (sec)</th>
<th><strong>Optimized time</strong> (sec)</th>
<th><strong>Error</strong> (voxels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (interpolating)</td>
<td>9.47</td>
<td>0.68</td>
<td>0.68</td>
<td>0.086</td>
</tr>
<tr>
<td>1</td>
<td>9.15</td>
<td>1.27</td>
<td>0.99</td>
<td>0.081</td>
</tr>
<tr>
<td>2</td>
<td>8.93</td>
<td>1.13</td>
<td>0.96</td>
<td>0.077</td>
</tr>
<tr>
<td>3</td>
<td>8.75</td>
<td>1.16</td>
<td>0.90</td>
<td>0.075</td>
</tr>
<tr>
<td>4</td>
<td>8.65</td>
<td>1.15</td>
<td>0.89</td>
<td>0.073</td>
</tr>
<tr>
<td>5</td>
<td>8.51</td>
<td>0.90</td>
<td>0.68</td>
<td>0.072</td>
</tr>
</tbody>
</table>

Table 4.2: Effect of smoothing splines on average iterations and time per correlation point, and error for spherical inclusion problem with ceramic foam. Optimized version uses previous $p$ values to speed up computation of smoothing splines.

computing the smoothing splines, the overall time was unchanged at 0.68 seconds per point. The mean absolute error over all correlation points decreased slightly from 0.086 voxels to 0.072 voxels.
Chapter 5

Scalability and Parallelism

Thus far we have examined how to improve the serial efficiency and robustness of a DVC implementation. This is an important first step, but to achieve our goal of computing high-resolution DVC in a reasonable time, two additional, interrelated issues must be addressed. The first issue we will address is how to store data so that we can scale the problem to a large image size within the amount of memory available on a typical computer. The second issue is how to parallelize the DVC code to reduce the required wall clock time. Portions of Sections 5.1–5.3 appeared in Gates et al. [25]. Sections 5.3–5.4 will appear in Gates et al. [26].

5.1 Storing 3D images

For 2D DIC, storing two $1024 \times 1024$ images in memory is feasible; together they require 2 MB if stored one byte per pixel. The coefficients necessary for a bicubic spline of the deformed image require 8 MB, 32 MB, or 128 MB, depending on the representation with the B-spline basis, Hermite basis, or monomial basis, respectively, using double precision floating-point numbers. This easily fits in modern computers, and even a large $125 \times 125$ subset can fit into a small 512 KB L2 cache.
For 3D DVC, however, two $1024 \times 1024 \times 1024$ images together require 2 GB of memory. The coefficients necessary for a tricubic spline of the entire deformed image require 8 GB, 64 GB, or 512 GB, depending on the representation. This is more memory than most desktop computers have today, so loading the entire image will produce significant disk swapping. Further, the memory requirements grow cubically with the resolution, so loading the entire image becomes even more untenable with higher resolution scans. 3D images are typically stored as a series of 2D images (e.g., in TIFF or PNG format), each representing a single slice with a constant value of the vertical coordinate $z$ (see Figure 2.2). Each subset intersects only a small number of these slices, so rather than attempting to load the entire 3D image, we developed a data structure called Image3D that loads only those slices that are currently in use.

From a user’s perspective, accessing voxels in an Image3D object operates much like a 3D array. To access the voxel with indices $i, j, k$, we use:

$$g = \text{image}(i, j, k);$$

which returns the $i, j$ entry of slice $k$. There is an additional function to load a contiguous range of image slices:

$$\text{image.load}(k_{\text{begin}}, k_{\text{end}});$$

Internally, Image3D maintains a vector of pointers to the 2D image slices. At any time, one contiguous block of these slices are loaded into memory, while pointers for all other slices are null, marking them as not loaded.

When starting a new subset of the reference image $f$ or deformed image $g$, DVC first calls Image3D’s load method to load the necessary range of slices. There are several cases for a load, depending on the overlap of the new range of slices with the current range of slices, which are illustrated in Figure 5.1 and handled in Algorithm 5. If the new range is a subset of the current range, nothing needs to be done (case 1). This common case occurs for all subsets of $f$ that have the same $z$ coordinate and
Algorithm 5 Image3D load slices

```
input: begin, end
if current_begin \leq \begin{align*}
\text{and end} \leq \text{current_end} \text{ then}
\end{align*}
// case 1: new range is subset of current range
return
else if begin \leq current_end \leq end then
// case 2: new range overlaps current_end
begin = \min\left( \begin{align*}
\text{end, max( current_begin, end - max_slices )} \end{align*}\right)
else if begin \leq current_begin \leq end then
// case 3: new range overlaps current_begin
end = \max\left( \begin{align*}
\text{end, min( current_end, begin + max_slices )} \end{align*}\right)
else
// case 4: current and new ranges disjoint; don’t change begin or end
end if
delete slices in range [ current_begin, begin )
delete slices in range [ end, current_end )
read slices in range [ begin, end ) that are not already loaded
current_begin = begin
current_end = end
```

are processed consecutively on the same processor. If the new range overlaps with the previous range, Image3D expands the new range to include currently loaded slices up to some maximum number of slices, 100 by default, as shown in cases 2a–2c and 3a–3c. Caching these extra slices ensures that minor variations in the deformation \( u \) between neighboring correlation points does not cause slices of \( g \) to be deleted and later re-loaded. If the new range is more than 100 slices, the entire new range is loaded (case 2d or 3d). Cases 2a–2d are handled by the second if-then condition in Algorithm 5, while the symmetric cases 3a–3d are handled by the third if-then condition. If the new range is disjoint with the previous range, then all the current slices are deleted and new slices are loaded (case 4), which is the default behavior if cases 1–3 do not occur.

This data structure makes the DVC algorithm scalable to large problem sizes, even on a single processor, rather than being restricted by the amount of available RAM. For instance, to perform DVC on a scan of the ceramic foam required less
Figure 5.1: Various cases of overlap between previous range of slices (top box in each case) and new range of slices (middle boxes) when reading new slices. Bottom row of boxes indicates slices that are read, deleted, and slices outside new range that are cached. New range is \([\text{begin}, \text{end}]\); \(\text{max}\) is maximum number of slices to cache.
than 240 MB of memory per processor, even though the entire 3D image would take 1905 MB for two images (992 × 1013 × 994 voxels). This data structure also has immediate benefits for parallel computing, as each processor can manage its own set of images independently, effectively decoupling processors.

5.2 Splines over subset

In 2D DIC, the spline function is often computed across the whole image as an initialization prior to computing any correlations. This would be expensive in 3D for two reasons: first, it requires storing the entire image in memory at once, which we avoid for scalability reasons; second, it requires solving three linear systems with the entire image, one for each dimension, with a transpose between each solve. It reads and writes $O(n^3)$ data six times, so even with a fast $O(n)$ banded solver, computing the spline coefficients would be expensive. We estimate it would take 30 minutes to solve for the coefficients of a $1024^3$ image on a single processor of Turing, based on extrapolating from timings for smaller $50^3$ to $400^3$ arrays. The spline could be computed in parallel, with each processor storing a piece of the image. In that case, solving the linear systems can be done locally, but each transpose becomes an expensive all-to-all parallel communication.

Instead, we compute a 3D spline over just the current subset plus a small padding region around it to accommodate minor perturbations in position and size during the minimization, as illustrated in Figure 5.2a. This allows processors to compute splines independently, eliminating parallel communication. The amount of padding is user specified; we generally use 5 voxels on all sides, which for a $51^3$ subset accommodates up to a 20% strain or 13° rotation without the spline being recomputed. Before each correlation function evaluation, we check that the bounding box for the $g$ subset is still contained within this spline region (Figure 5.2b), and
re-compute the spline over a new region if necessary. For the linear shape function (1.1), a tight bounding box is given by

\[
\hat{h}_x = \max \left\{ h \left( 1 + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} - \frac{\partial u}{\partial z} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial u}{\partial x} - \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial u}{\partial x} - \frac{\partial u}{\partial y} - \frac{\partial u}{\partial z} \right) \right\},
\]

\[
\hat{h}_y = \max \left\{ h \left( 1 + \frac{\partial v}{\partial y} + \frac{\partial v}{\partial x} + \frac{\partial v}{\partial z} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial v}{\partial y} + \frac{\partial v}{\partial x} - \frac{\partial v}{\partial z} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial v}{\partial y} - \frac{\partial v}{\partial x} + \frac{\partial v}{\partial z} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial v}{\partial y} - \frac{\partial v}{\partial x} - \frac{\partial v}{\partial z} \right) \right\},
\]

\[
\hat{h}_z = \max \left\{ h \left( 1 + \frac{\partial w}{\partial z} + \frac{\partial w}{\partial x} + \frac{\partial w}{\partial y} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial w}{\partial z} + \frac{\partial w}{\partial x} - \frac{\partial w}{\partial y} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial w}{\partial z} - \frac{\partial w}{\partial x} + \frac{\partial w}{\partial y} \right), \right.
\]
\[
\left. h \left( 1 + \frac{\partial w}{\partial z} - \frac{\partial w}{\partial x} - \frac{\partial w}{\partial y} \right) \right\},
\]

where \( \hat{h}_x, \hat{h}_y, \hat{h}_z \) are the maximum displacement in each coordinate direction from the center of the subset and \( h \) is half the subset size, as illustrated in Figure 5.2. A simpler but loose bounding box is given by

\[
\hat{h}_x = h \left( 1 + \left| \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z} \right| \right),
\]
\[
\hat{h}_y = h \left( 1 + \left| \frac{\partial v}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial v}{\partial z} \right| \right),
\]
\[
\hat{h}_z = h \left( 1 + \left| \frac{\partial w}{\partial x} + \frac{\partial w}{\partial y} + \frac{\partial w}{\partial z} \right| \right).
\]
For the quadratic shape function (1.2), a loose bounding box is

\[
\hat{h}_x = h \left( 1 + \left| \frac{\partial u}{\partial x} \right| + \left| \frac{\partial u}{\partial y} \right| + \left| \frac{\partial u}{\partial z} \right| + h \left( \left| \frac{\partial^2 u}{\partial x^2} \right| + \left| \frac{\partial^2 u}{\partial y^2} \right| + \left| \frac{\partial^2 u}{\partial z^2} \right| + \left| \frac{\partial^2 u}{\partial x \partial y} \right| + \left| \frac{\partial^2 u}{\partial x \partial z} \right| + \left| \frac{\partial^2 u}{\partial y \partial z} \right| \right) \right),
\]

\[
\hat{h}_y = h \left( 1 + \left| \frac{\partial v}{\partial x} \right| + \left| \frac{\partial v}{\partial y} \right| + \left| \frac{\partial v}{\partial z} \right| + h \left( \left| \frac{\partial^2 v}{\partial x^2} \right| + \left| \frac{\partial^2 v}{\partial y^2} \right| + \left| \frac{\partial^2 v}{\partial z^2} \right| + \left| \frac{\partial^2 v}{\partial x \partial y} \right| + \left| \frac{\partial^2 v}{\partial x \partial z} \right| + \left| \frac{\partial^2 v}{\partial y \partial z} \right| \right) \right),
\]

\[
\hat{h}_z = h \left( 1 + \left| \frac{\partial w}{\partial x} \right| + \left| \frac{\partial w}{\partial y} \right| + \left| \frac{\partial w}{\partial z} \right| + h \left( \left| \frac{\partial^2 w}{\partial x^2} \right| + \left| \frac{\partial^2 w}{\partial y^2} \right| + \left| \frac{\partial^2 w}{\partial z^2} \right| + \left| \frac{\partial^2 w}{\partial x \partial y} \right| + \left| \frac{\partial^2 w}{\partial x \partial z} \right| + \left| \frac{\partial^2 w}{\partial y \partial z} \right| \right) \right).
\]

Using a loose bounding box may falsely indicate that the g subset is outside the current region, initiating an unnecessary re-computation of the spline. This will decrease the performance but is otherwise safe.

A variant on this approach would be to use a larger amount of padding to create a spline that covers multiple subsets. The padding would best be done asymmetrically, with more padding in the positive x direction, since we order correlation points in the x direction first, then y and z. This approach would be particularly advantageous if the correlation grid spacing is small, to avoid computing splines over regions that are offset by a small amount and hence overlap significantly. We have not yet explored this optimization, however.

5.3 Coarse-grained parallel computing

A computation can be decomposed into tasks to compute in parallel at several different granularities. Given a series of 3D images to analyze using DVC, perhaps the coarsest granularity for parallel computing would be to run multiple instances of DVC to analyze multiple pairs of images simultaneously. Within the DVC computation for a single pair of images, there are at least two additional levels of granularity: a coarse-grained decomposition computes multiple correlation points in parallel, while a fine-grained decomposition computes the correlation objective function for a single correlation point in parallel. We address the coarse-grained parallelism
Figure 5.2: Solid black lines indicate deformed $g$ subset after a linear transformation. (a) Derivatives in initial transformation $u_0$ are zero, so initial subset is square. Outer gray region is padding added before computing spline. (b) After optimization step, derivatives are nonzero, introducing strain and rotation. Dashed lines show bounding box, which here is still inside initial spline region.
in this section and discuss the fine-grained parallelism in Section 5.4.

To implement a coarse-grained parallel algorithm, we assign different correlation points to different processors. A key question is how to assign correlation points to specific processors. Every correlation point is computed independently, apart from extrapolating the starting point from neighboring correlation points. In our storage scheme for 3D images, each processor loads only the slices it needs for the current subset, as shown in Figure 5.3. Therefore, assigning correlation points in a plane with the same \( z \) coordinate to a single processor will maximize reuse of image data already in memory. This makes the algorithm scalable to large data sets because each processor reads only a fraction of the data, instead of every processor reading and storing the entire data set. Also, this provides the maximum benefit of using neighboring correlation points on the same processor to provide a good starting guess for the next correlation point.

However, this simple 1D decomposition may produce poor efficiency resulting from load imbalance. For \( p \) processors, we define the cost \( c_p \) in processor-hours as

\[
c_p = (\text{number of processors}) \times (\text{wall clock execution time}) = p t_p, \quad (5.1)
\]
and efficiency as

\[ e_p = \frac{\text{serial cost}}{\text{parallel cost}} = \frac{c_1}{c_p}. \]  

(5.2)

Consider computing a $5 \times 5 \times 5$ correlation grid with four processors, as shown in Figure 5.4. With a 1D decomposition, one processor gets twice as many points as the other three processors (Figure 5.4a). Assuming all points take the same time to compute and no parallel overhead other than load imbalance exists, the best efficiency achievable is 63%, given by

\[
\frac{\text{total points}}{(\text{number of processors}) \times (\text{maximum points per processor})} = \frac{125}{4 \times 50}.
\]

With more than 5 processors, some processors would get no points and would therefore be completely idle! A 2D decomposition yields a 69% efficiency (Figure 5.4b). We improve on these by considering two other decompositions. One option is to group points into rows, where all points in a row have the same $y$ and $z$ coordinates, and divide rows evenly among processors, yielding a maximum difference of one row between different processors (Figure 5.4c). In this example, decomposition by rows yields an 89% maximum efficiency. Another option is to divide the total number of points evenly among processors, yielding a maximum difference of one point between different processors (Figure 5.4d). Here, this results in an 98% maximum efficiency.

This static load balancing scheme assumes that each correlation point takes approximately the same amount of time to compute. However, variations in image quality and the underlying deformation field can cause correlation points to take different amounts of time to compute, leading to a load imbalance. We used MPE and Jumpshot [11] to profile and visualize the MPI communication and idle time for the static load balancing scheme, as shown in Figure 5.5a. Each horizontal line represents one of four processors. The orange blocks show idle time at the end of
the computation for processors 0, 1, and 3 as they wait for processor 2 to finish, demonstrating load imbalance. To correct this imbalance, we develop a master-worker dynamic load balancing scheme, described below.

For a dynamic load balancing scheme, we assign one MPI process to be the master, and all other processes to be workers. The master process creates a set of tasks, with each task being a single row of the correlation grid. For each worker process, the master sends a task and sets up a nonblocking receive to wait for results back from the worker. The master also assigns itself a task, since for a small number of processors we do not want to sacrifice an entire processor for the master. However, with a sufficiently large number of processors, we expect the management of tasks to be enough work to warrant a dedicated master process.

The master then enters its main loop. It does one subtask, which we define as a single correlation point, then tests the set of nonblocking receives for any incoming results from workers. For any results that come in, it records the results and sends a new task to that worker process. Each worker receives a task, processes

---

**Figure 5.4:** Decomposition of $5 \times 5 \times 5$ grid of correlation points onto 4 processors. (a) 1D decomposition yields subdomains with 50, 25, 25, and 25 points; (b) 2D decomposition yields subdomains with 45, 30, 30, and 20 points; (c) decomposition by rows yields subdomains with 35, 30, 30, and 30 points; (d) decomposition by points yields subdomains with 32, 31, 31, and 31 points.
Figure 5.5: Visualization of idle time and communication time for DVC using (a) static and (b) dynamic load balancing schemes. Orange blocks at end are idle time. Red lines for master process 0 are testing for incoming messages. Blue and green blocks are sends and receives.
the correlation points in it, sends results back to the master, and waits for a new assignment. When all tasks have been assigned, the master sends a flag telling the worker it is finished.

Figure 5.5b shows a profile of the communication and idle time for the dynamic load balancing scheme. The numerous red lines on the master process 0 denote it testing for incoming messages after it computes each correlation point. The blue and green blocks on worker processes 1–3 show them sending results to, and receiving a new task from, the master process after completing each task. The orange blocks at the end are again idle time, but are significantly less than the idle time for the static load balancing scheme in Figure 5.5a. The wall clock time is reduced 8% from 347 seconds to 319 seconds, showing that the added communication for dynamic load balancing is more than compensated by the improved load balancing.

We still want to be efficient with reusing the existing slices that have been loaded by each process. Therefore, rather than assigning tasks in a round-robin fashion, with task 0 going to process 0, task 1 to process 1, etc., we instead make an initial distribution of tasks identical to the static load balancing “by rows” (Figure 5.4c). If a process finishes all of its initial distribution of tasks, the master will re-assign it tasks from other processes. This is managed by creating a vector on the master of all the tasks, with a flag for each task indicating whether it is assigned. For \( n \) tasks and \( p \) processes, process \( i \) is initially assigned task number \( i \cdot \left\lfloor \frac{n}{p} \right\rfloor \). After finishing a task, a worker’s assigned task number is incremented to the next unassigned task number. If the task number reaches \( n \), it wraps around to zero.

We implemented these parallel algorithms using MPI [54], to run in a variety of parallel and distributed computing environments. To test the scalability of our algorithms, we perform DVC on the spherical inclusion problem with the ceramic foam sample using a \( 31^3 \) subset size, and vary the number of processors on Turing. We test three different problem sizes, all of which cover the same \( 500 \times 500 \times 500 \)
voxel region of the image. The small problem size uses a 20 voxel grid spacing, yielding a $26 \times 26 \times 26$ grid with 17,576 correlation points. The medium problem size uses a 10 voxel grid spacing, yielding a $51 \times 51 \times 51$ grid with 132,651 correlation points. The large problem size uses a 5 voxel grid spacing, yielding a $101 \times 101 \times 101$ grid with 1,030,301 correlation points.

In Figure 5.6a we plot in log-log scale the wall clock time versus number of processors. A slope of negative one, shown by the triangle and black lines, indicates linear speedup. Both the static (dashed blue lines) and dynamic (solid red lines) load balancing schemes come close to linear speedup. In all cases the dynamic scheme is faster than the static scheme. For the parallel efficiency $e_p$, defined in (5.2) and plotted in Figure 5.6b, we estimate the serial time by summing the computation time for the static load balancing scheme, excluding any idle time. The static scheme gradually loses efficiency with more processors, while the dynamic scheme is better able to maintain a high efficiency. For the small problem size (solid line with circles), the dynamic scheme starts to deviate from a straight line in Figure 5.6a, which is also seen in Figure 5.6b as a reduction in parallel efficiency. This occurs because there is insufficient work to keep all the processors busy: for 128 processors there are on average only 5.3 tasks per processor. For the medium and large problem sizes, the dynamic scheme maintains greater than 90% efficiency up to the maximum number of processors tested, while the efficiency with the static scheme drops to 70%. For the large problem with 128 processors, the dynamic scheme is 34% faster than the static scheme. The largest problem—which is our target million point correlation grid—would take an estimated 138 hours on a single CPU. The dynamic scheme solves it in 69 minutes with 128 processors; the static scheme in 93 minutes. Since a pair of CT scans typically takes 1–2 hours, this achieves our goal of computing high-resolution DVC in time commensurate with image acquisition time.
A complementary parallelization is to compute the correlation function itself in parallel. Recall that the correlation function, for example, the least-squares function

\[
c(u) = \frac{\sum_{x \in S} (f(x) - g(\hat{x}(u)))^2}{\sum_{x \in S} f(x)^2},
\]

(5.3)
is a summation over all points \(x\) in the subset \(S\). Computing the correlation function and its gradient in parallel entails computing the spline interpolation of \(g(\hat{x})\) at every point in parallel, then doing parallel reductions to sum the correlation function and its derivatives. While there is an insufficient computation to communication ratio to do this efficiently with loosely coupled distributed computing such as MPI, modern graphics processing units (GPUs) provide a lightweight thread model that is ideal for this type of computation.

Figure 5.6: (a) Parallel scaling in log-log scale for three fixed problem sizes. Solid black lines show linear speedup. (b) Parallel efficiency for same tests. Results for spherical inclusion problem on Turing.

5.4 Fine-grained parallel computing with GPU
CUDA [56, 57] is a parallel programming language developed by Nvidia Corporation to allow general purpose programming of GPUs. Its syntax is similar to C++, adding several CUDA-specific keywords. The CUDA programming model assumes a host CPU, which executes serial code, and a GPU device, which executes parallel code. The host CPU copies data from the CPU’s main memory to the GPU’s device memory, calls a kernel function that is executed on the GPU, then copies results from the GPU memory back to the CPU memory. Other languages such as OpenCL [55] also exist for programming GPUs, including GPUs by other manufacturers. How to best implement code on a GPU depends on details of the specific GPU hardware architecture. In this work, we focus on using CUDA with Nvidia GPU cards.

In CUDA, the parallel computation is decomposed into a 1D or 2D grid of blocks, which are executed asynchronously on the GPU. Each block is further decomposed into a 1D, 2D, or 3D grid of threads, all of which execute the same kernel function in lockstep on different pieces of data. There is no communication or synchronization between blocks in a single kernel call. Within a block, threads can communicate via shared memory and synchronize via a barrier.

A GPU consists of one or more streaming multiprocessors, each of which has multiple cores that execute threads in parallel. The Nvidia Tesla C2050, for example, has 14 multiprocessors with 32 cores each, for a total of 448 cores [58]. A set of 32 threads from one block, called a warp, is executed simultaneously on a multiprocessor. It is thus advantageous to make the number of threads in a block a multiple of 32 to use all cores fully. Up to 8 blocks can be assigned to a single multiprocessor, subject to constraints on the number of registers, threads, and amount of shared memory each block uses [56, p. 154].

A GPU uses multiple blocks to hide memory latency. While one block is waiting for a memory read to complete, another block can be executing. Unlike CPUs,
where context switching between threads is expensive, GPUs have zero-overhead context switching, so can execute two different blocks in consecutive cycles. Having a sufficient number of blocks and amount of computation within each block to hide memory latency is important to achieving high performance. For good efficiency, a computation on the GPU uses hundreds to thousands of blocks, each with hundreds of threads.

For DVC, we assign each thread to compute one point in the subset, and make blocks by tiling each slice of the subset with 2D tiles, as illustrated in Figure 5.7. Using $16 \times 16$ blocks to cover a $31^3$ subset results in a $2 \times 2 \times 31$ grid of blocks, for a total of 124 blocks, each containing 256 threads. Since 31 is not evenly divisible by 16, tiles along two edges will have one row or column that is outside the subset; for these points the results are set to zero.

We copy the subset of the reference image $f$ and the spline coefficients of the deformed image $g$ to 3D textures on the GPU to take advantage of texture caching. These do not need to be copied for each correlation function evaluation, but only when starting a new subset or if the spline is recomputed. The vector $u$ is copied to the GPU’s constant memory, which is also cached. Recall from Table 4.1 that a $50^3$ subset takes 1163 KB of memory and requires 231 flops per point, or 28.9 Mflops total. This yields an overall computation-to-memory ratio of 194 flops/word. Each point takes 231 flops and accesses 64 floating point coefficients, for a 3.6 flops/word ratio. However, neighboring points share 48 coefficients which are cached, leaving only 16 coefficients to transfer, for a 14.4 flops/word ratio. These large computation-to-memory ratios allow for overlapping computation and communication to hide memory latency, yielding excellent performance on the GPU.

After computing the spline $g(\hat{x}(u))$ and its derivatives at each point, each block does a series of standard parallel sum reductions [39] to compute partial sums of the correlation function $c(u)$ and its derivatives within the block. Because there is no
communication between blocks, the partial sums from each block are transferred back to the CPU, which does a final summation. While this GPU code parallelizes the computation of the correlation function for a single subset, we also combined it with our MPI-based DVC, making a hybrid implementation that simultaneously does both coarse-grained parallelism across multiple CPUs and, for each CPU, fine-grained parallelism using a GPU.

We use single precision floating point with CUDA, since it is supported on all GPU cards and has twice the performance of double precision floating point on recent cards such as the Nvidia C2050 [58]. The GPU algorithm also computes in a different order — the parallel sum reduction adds terms together in a hierarchical tree fashion, which tends to yield more accurate results since numbers of similar magnitude are summed at each step, as compared to a serial implementation that sums terms into a single accumulator. To assess the impact of single precision and algorithmic changes, we compare DVC solutions for the spherical inclusion test computed using single and double precision. Global error properties are identical between the solutions: both have 1.45% error and a standard deviation in displace-
Figure 5.8: Difference in displacement $u$ using double and single-precision, in linear and log scale.

ment of 0.042 voxels. Figure 5.8 shows a histogram of the difference between the computed displacements, $u_{\text{single}} - u_{\text{double}}$, in linear and log scale for 2744 correlation points. Differences are small; 95% of points are within 0.001 voxels, much less than the standard deviation of 0.042 voxels. We conclude that single precision is sufficient to compute DVC to within its experimental accuracy limits.

We first test the performance of the GPU for computing the correlation function itself, apart from other parts of our DVC code. To simulate solving multiple correlation points with several BFGS iterations per point, we call the correlation function 50 times, copying the $f$ and $g$ data from the CPU to the GPU every 5 iterations. We achieve up to a 39.8 times speedup using a GPU compared to a CPU alone (both using single precision), as shown in Figure 5.9, on a machine with a 2.66 GHz Intel Core 2 Quad core CPU and Tesla C2050 GPU. However, the speedup is variable and depends on the machine, tile size, and subset size. Generally, larger subset sizes have a larger speedup. A decrease in performance is seen when a multiple of the tile size is exceeded, requiring another row or column of tiles. For instance, $8 \times 8$, $8 \times 16$, and $16 \times 16$ tile sizes all show a decrease in speedup between subset
sizes of $47^3$ and $49^3$, when 48 is exceeded. Medium size tiles with 64 to 256 threads ($4 \times 32$, $8 \times 8$, $8 \times 16$, $16 \times 16$) have the best performance with speedups from 19.8 to 39.8, while small tiles ($4 \times 4$, $4 \times 8$) and large tiles ($16 \times 32$) show lesser speedups from 10.9 to 28.7. For subsequent tests, we choose the $8 \times 16$ tile size because it performed well consistently, being the first or second fastest for all subset sizes.

We then test performance of our entire DVC application with the GPU, using the spherical inclusion test with a $14^3$ grid of 2744 points and a $31^3$ subset size. We achieve a speedup of 5.9 using the GPU compared to using the CPU alone, from 0.228 seconds per point with only the CPU to 0.038 seconds per point with the GPU. The smaller 5.9 speedup for the whole application compared to the 39.8 speedup for just the correlation function is a consequence of Amdahl’s law [1]: we parallelize one portion of the application, but the maximum speedup achievable is limited by the amount of time spent in the remaining serial code. Some of this serial code is also amenable to computation with the GPU, for instance, Ruijters and Thevenaz [68] developed an algorithm to compute the B-spline coefficients on the GPU.
Figure 5.10: Parallel scaling in log-log scale for one to four CPU cores, all sharing one GPU, on spherical inclusion problem with 2744 points. Solid black lines show linear speedup.

Because we are not fully utilizing the GPU, multiple CPU cores can share a single GPU and all gain a speedup. We show the speedup for our hybrid GPU and MPI code in Figure 5.10, using one to four CPU cores all sharing one GPU, and compare with a CPU-only MPI code. Both the CPU-only and the GPU code show near linear speedup in the number of CPU cores, even though there is only one GPU. The overall speedup from using a single core to using four cores and the GPU is 20.3 times, from 624.6 seconds to 30.8 seconds. We expect that if more of the DVC code is written for the GPU, each core would show a larger speedup but the GPU would not be able to be shared by as many cores.

These initial tests are very promising for using our hybrid GPU and MPI code. With four CPU cores and one GPU, each core took on average 0.045 seconds per correlation point. To solve our target million point correlation grid at this rate would take an estimated 12.5 CPU-hours, or 3.1 hours with four CPU cores and
one GPU. Using a small cluster with 16 CPU cores and four GPUs, this could po-
tentially be performed in parallel in under an hour, achieving our goal of comput-
ing high-resolution DVC in time commensurate with CT scanning time, using far
fewer CPUs than our CPU-only MPI code.
Chapter 6

Refinement and Adaptivity

We showed in Chapter 5 that we can compute high-resolution DVC. To ensure these DVC results have high accuracy and can resolve strain gradients, various DVC parameters — subset size, correlation grid spacing, and shape function — must be chosen carefully. This choice is influenced by both the underlying deformation and the image pattern, e.g., size and density of speckles, image contrast, and image noise. Because of these influences, the optimal choice may be different in different parts of the image, so refinement is required to achieve the best accuracy. By refining the solution adaptively, we also aim to reduce the computational cost. To understand where and how to refine, we first investigate various error estimates, then examine refinement for a quadratic model problem. Results from this chapter are to appear in Gates et al. [27].

6.1 Error estimates

Several studies have examined the relationship between error, subset size, shape function, and image pattern for 2D DIC. Yaofeng and Pang [78] propose a subset
entropy measure,
\[
\delta = \frac{\sum_{x \in S} \sum_{\tilde{x} \in N(x)} |f(x) - f(\tilde{x})|}{2\beta s^2},
\]
where \( S \) is the subset, \( s \) is the subset size, \( N(x) \) are the eight pixels neighboring \( x \), and \( \beta \) is the image bit depth (e.g., 8 bit or 16 bit gray scale). Their tests show that the standard deviation \( \sigma_u \) of the error in displacement decreases as the mean subset entropy of the image increases. Further, for translation and tension tests, \( \sigma_u \) decreases with increasing subset size.

Subsequent studies derived specific estimates of \( \sigma_u \) based on the image noise \( \sigma \) and gradients of the image gray intensity level \([59, 75, 76]\). In particular, Wang et al. \([75]\) derive the estimate
\[
\sigma_u \approx \left( \frac{2\sigma^2}{\sum_{x \in S} \left( \frac{\partial}{\partial x} g(\hat{x}) \right)^2} \right)^{1/2}
\]
for a 1D translation in the \( x \) direction, and equations with similar form for 1D strain and 2D translation. Equation (6.1) is shown as the dashed line in Figures 6.2, 6.3, and 6.4. This error measure suggests that increasing the subset size, which includes more terms in the denominator, reduces the standard deviation of the error. A good distribution of speckles in the image, good image contrast, and low image noise likewise reduce \( \sigma_u \), but we will assume these have already been addressed in preparing the sample and acquiring a particular image and so are now fixed. Pan et al. \([59]\) use a similar estimate to devise an algorithm for automatic selection of the subset size. As a single number that measures the overall quality of an image, Pan et al. \([60]\) propose the mean intensity gradient,
\[
\delta_f = \frac{\sum_{i=1}^{W} \sum_{j=1}^{H} \| \nabla f(x_{ij}) \|_2}{WH},
\]
where \( W \) and \( H \) are the width and height of the image, respectively. The standard deviation then can be approximated by

\[
\sigma_u \approx \left( \frac{2\sigma^2}{\sum_{x \in S} \left( \frac{\partial^2 u}{\partial x^2} \right)^2} \right)^{1/2} \approx \left( \frac{2\sigma^2}{S^2\sigma_f^2} \right)^{1/2}.
\]

Again, these measures all suggest that the standard deviation of the error is reduced by making the subset size larger, under the assumption that the shape function can accurately represent the underlying displacement.

Wang et al. [75] also derive expressions for the expected value of the displacement \( u \), which show a bias using bilinear and \( C^1 \) bicubic image interpolation. This bias is addressed by Schreier et al. [71], who suggest using cubic or quintic splines to reduce the bias. We show the effects of this interpolation bias and further discuss how to mitigate it in Section 7.2.

When the shape function does not match the underlying displacement, a systematic bias error is measured by DIC, as discussed by Schreier and Sutton [70]. For a linear shape function fitting a quadratic displacement field, they derive the bias error estimate

\[
\Delta u = \frac{1}{3}aM(M + 1) = \frac{1}{24} \frac{\partial^2 u}{\partial x^2} (s^2 - 1),
\]  

(6.2)

where \( M = (s - 1)/2 \) is half the subset size and \( a = \frac{1}{2} \frac{\partial^2 u}{\partial x^2} \) is the coefficient of the quadratic term in the displacement field, \( u(x, y) = ax^2 \). Equation (6.2) is shown as the dotted line in Figures 6.2, 6.3, and 6.4. This result implies that smaller subsets are required to achieve accurate DIC results when using linear shape functions. Yaofeng and Pang [78] also show experimentally that, when using a linear shape function to measure a quadratic displacement field, as the subset size increases, \( \sigma_u \) initially decreases but then increases due to this mismatch. Using a quadratic shape
function eliminates this bias error since it can fit the quadratic displacement field. More generally, the shape function is a Taylor approximation to the displacement field, so a quadratic shape function fits a general displacement field more accurately than a linear shape function does. However, the standard deviation $\sigma_u$ is also larger when using quadratic shape functions [70], because the extra quadratic terms allow for more flexibility in matching the displacement. For instance, in a linear field, the quadratic terms should all be zero, but due to experimental noise will be some small but non-zero value, introducing more variability into the results. Computing the correlation function is about twice as expensive with a quadratic shape function compared to a linear shape function. Thus the choice of shape function is still a question of interest.

Regarding the choice of correlation grid spacing, a common practice is to overlap subsets by half the subset size, so a subset size of 41 voxels yields a grid spacing of 20 voxels between correlation points [20, 21, 22]. If this practice is followed, asymptotically the computational cost depends on the size of the entire region of interest being correlated, independent of the subset size and grid spacing. The computational complexity for 3D DVC with a subset size $s$, grid spacing $h$, and region size $R$ is

$$O\left(s^3 \left( \frac{R}{h} \right)^3 \right) = O\left(s^3 \left( \frac{R}{s/2} \right)^3 \right) = O(8R^3).$$

The factor of 8 occurs because every voxel in the region of interest is part of 8 overlapping subsets (except voxels near the boundary of the region, which are in fewer than 8 subsets). To resolve strain gradients that occur over a small region requires a sufficient number of samples within the region, implying a smaller correlation grid spacing, for example, with 5 voxel spacing. For a fixed subset size, the computational cost for DVC grows cubically with the inverse of the grid spacing. This was seen previously in Figure 5.6a, where the time increases by 6.7 times going from
17k points (20 voxel spacing) to 132k points (10 voxel spacing), and by another 6.7 times going to 1M points (5 voxel spacing).

Considering the standard deviation (6.1), bias error (6.2), and cost of grid refinement together suggests that, even for a single image, there is no optimal combination of subset size, grid spacing, and shape function. Rather, in regions of linear strain, a large subset size and correspondingly large grid spacing with linear shape functions is preferable to reduce the noise and cost, while in regions where the strain is changing (e.g., a quadratic displacement field), smaller grid spacing and smaller subsets or a quadratic shape function are required to measure the displacements accurately. Our goal in this chapter is to examine this issue in more detail and derive guidelines for how to refine a correlation.

6.2 2D quadratic displacement model problem

To test the effects of subset size on DIC accuracy, we will use a quadratic displacement model problem, defined in Section 2.2. This problem has a linear displac-

\[ u(x) = \begin{cases} 
  u_0 & \text{for } x < x_1 \\
  \frac{1}{2}a(x-x_1)^2 + u_0 & \text{for } x_1 < x < x_2 \\
  u_0 & \text{for } x_2 < x 
\end{cases} \]

\[ \varepsilon(x) = \begin{cases} 
  0 & \text{for } x < x_1 \\
  \frac{1}{2}a(x-x_1)^2 & \text{for } x_1 < x < x_2 \\
  0 & \text{for } x_2 < x 
\end{cases} \]

Figure 6.1: (a) Displacement field that is linear for \( x < x_1 \), quadratic for \( x_1 < x < x_2 \), and linear for \( x_2 < x \). (b) Corresponding strain field.
ment field with strain $\epsilon_1$ for $x < x_1$ (the “left” region), a quadratic displacement field with strain varying from $\epsilon_1$ to $\epsilon_2$ for $x_1 < x < x_2$, and a linear displacement field with strain $\epsilon_2$ for $x_2 < x$ (the “right” region), illustrated in Figure 6.1. By varying the width $x_2 - x_1$ of the quadratic region and the strains $\epsilon_1$ and $\epsilon_2$, we can vary the magnitude of the second derivative $\frac{\partial^2 u}{\partial x^2}$ in the quadratic region.

We start by examining tests for a 2D image using 2D DIC. For these tests we use the titanium sample shown previously in Figure 3.5a, and generate a second image by artificially deforming this image according to the displacement field in equation (2.1). We add random Gaussian noise with zero mean and standard deviation $\sigma = 3$ gray levels to both the reference and deformed image, to simulate noise from a camera. We compute DIC results on a correlation grid with 5 pixel spacing for various subset sizes.

Figure 6.2 shows results using $\epsilon_1 = 0.5\%$, $\epsilon_2 = 1\%$, and a 200 pixel wide quadratic region. The second derivative in the quadratic region is $2.5 \times 10^{-5}$ pixels$^{-1}$. Results exclude correlation points within 50 pixels of $x_1$ and $x_2$, so every subset is entirely in a linear region or entirely in the quadratic region. The blue lines with $\times$’s and green lines with $+$’s in Figure 6.2 are results for the linear regions, while the red lines with circles are for the quadratic region. The top row (a,b) uses linear shape functions and the bottom row (c,d) uses quadratic shape functions. The left column (a,c) shows the standard deviation of error for each region, as well as the theoretical value estimated by (6.1) as the dashed line. The standard deviations of experimental results decrease as the subset size increases and have the same general shape as the theoretical result, though they are somewhat larger here than expected. The standard deviations using the quadratic shape functions (c) are larger than those using linear shape functions (a), which is consistent with Schreier and Sutton [70].

The right column (b,d) shows the mean absolute value error for the three regions, i.e., the 1-norm of the error divided by the number of correlation points, as
Figure 6.2: Standard deviation and mean absolute error for model problem with 200 pixel wide quadratic region, going from 0.5% strain in left region to 1% strain in right region.
well as the theoretical error estimated by (6.2) for the linear shape function as the dotted line. The error is defined as $|u - u^*|$ at each correlation point, where $u$ is from DIC results and $u^*$ is the exact solution. Because the change in strain is small, the error is dominated by the standard deviation for both the linear and quadratic shape functions and decreases with increasing subset size. For the quadratic region, the error levels off and starts increasing slightly for large subset sizes.

With a larger change in strain, the dominant error is due to the mismatch between a linear shape function and the underlying quadratic displacement field. Figure 6.3 shows results for a test with $\varepsilon_1 = 2\%$ and $\varepsilon_2 = 4\%$. The second derivative is $10 \times 10^{-5}$ pixels$^{-1}$. Within the quadratic region (red line with circles), the error increases with subsets larger than 40 pixels (Figure 6.3b); this increase matches well with the theoretical results from (6.2) (dotted line), while for small subset sizes the standard deviation dominates the error. Minimizing the error yields an optimal subset size of 37 pixels, though the error is not sensitive to the exact value. A quadratic shape function (d) can accurately represent the underlying quadratic displacement field, so the error continues to decrease with increasing subset size. For the linear regions, since a linear shape function can represent the underlying displacement field and has a smaller standard deviation, its error is smaller than when using a quadratic shape function.

If the width of the quadratic region is made narrower, subsets will overlap both the linear and quadratic regions, so the displacement field in each subset will no longer be a single quadratic function. For subsequent tests, results for the linear regions still exclude points within 50 pixels of $x_1$ and $x_2$, but results for the quadratic region include all points in $[x_1, x_2]$.

We found that for narrower quadratic regions, equation (6.2) overestimates the error. Figure 6.4 shows results for a width of 10 pixels, with $\varepsilon_1 = 0$ and $\varepsilon_2 = 2\%$. The second derivative is $400 \times 10^{-5}$ pixels$^{-1}$. The theoretical error result in Figure
Figure 6.3: Standard deviation and mean absolute error for model problem with 200 pixel wide quadratic region, going from 2% strain in left region to 4% strain in right region.
Figure 6.4: Standard deviation and mean absolute error for model problem with 10 pixel wide quadratic region, going from zero strain in left region to 2% strain in right region.
6.4b increases much faster than the error for DIC results. The standard deviation in the quadratic region also does not continue to decrease but levels off for subset sizes larger than 30 pixels. Due to the large change in strain, the linear shape function requires small subset sizes and makes a larger 0.036 pixel error, compared to errors less than 0.01 pixels in previous tests. Using a quadratic shape function has improved results, with a minimum error of 0.021 pixels. Because the underlying deformation is no longer quadratic, the quadratic shape function cannot exactly represent it, causing the error to rise for subset sizes greater than 40 pixels.

We repeat this test for various changes in strain and quadratic region widths. Figure 6.5 summarizes the errors for the quadratic region using a linear shape function. The bottom graph is a close-up view of the top graph. The solid blue lines correspond to going from zero strain in the left region to 2% strain in the right region, the dashed green lines from zero to 0.5% strain, and the dotted red lines from zero to 0.1% strain. The widths of the quadratic region are indicated by line markers. For small strain, the linear shape function represents the displacement well, so the errors continue to decrease with increasing subset size. For larger strain changes, the error increases with subset size. Unlike the error estimate (6.2) for a purely quadratic region, which grows quadratically with the subset size, here the error asymptotically becomes linear in the subset size. A least-squares fit to the lines in Figure 6.5 for subset sizes greater than 50 voxels yields the estimate

\[ |u - u^*| = 0.12 \left( \frac{\partial u}{\partial x} \right) s + c, \]

where \( \frac{\partial u}{\partial x} \) is the strain in the right region, \( s \) is the subset size, and the constant \( c \) depends on the width of the region. The optimal subset size depends on the strain change and width of quadratic region. For the 0 to 2% strain change, the optimal subset sizes are between 20 and 25 pixels, while for the 0 to 0.5% strain change, the
Figure 6.5: Mean absolute error for model problems using linear shape function. Has zero strain in left region, strain varying from 0.1% to 2% in right region, and quadratic region varying from 10 to 50 pixels wide. Bottom graph is close-up view of bottom-right corner of top graph.
optimal subset sizes are around 30 pixels.

Figure 6.6 shows errors for the same problems using a quadratic shape function. The small (0 to 0.1%) and medium (0 to 0.5%) strain changes are represented well and the error continues to decrease with increasing subset size. For the larger 0 to 2% change, the error starts to increase with subset size, yielding an optimal subset size around 35 to 40 pixels. For small subset sizes, the error using quadratic shape functions (Figure 6.6) is higher than when using linear shape functions (Figure 6.5), but for larger subset sizes the quadratic shape functions have a smaller error.

### 6.3 3D quadratic displacement model problem

To extend these results to DVC, we perform a similar test for a 3D image, using a limited number of settings due to the computational expense. For 3D tests, we use two baseline images of the ceramic foam and apply the artificial deformation to one
of them, as described in Section 2.2. These images already include effects from CT scanner noise, so we do not add any additional noise.

The displacement $u$ has the same variation with $x$ along every line parallel to the $x$ axis. Therefore, to visualize both the accuracy and variability of the solution, we overlay multiple line scans for a single slice $z = 340$ as the light green lines in Figure 6.7, each line having a different fixed $y$ coordinate in the correlation grid. The thick dashed line shows the exact solution. The vertical dotted lines represent the boundaries of the quadratic region at $x_1 = 500$ and $x_2 = 520$. The strains are $\epsilon_1 = 0$ and $\epsilon_2 = 2\%$, with a quadratic region of width 20 voxels, yielding a second derivative of $10^{-3}$ pixels$^{-1}$. There is significant variability between line scans for the smallest subset size and the variability diminishes as the subset size increases. The DVC measurements with the largest subset size have a wider quadratic region and smaller curvature than the true solution, underestimating the magnitude of the strain gradient. Smaller subsets capture the solution more accurately but with higher noise. A slight sinusoidal signal is seen along the linear regions, most easily visible in Figure 6.7b for the subset size 31; this is caused by the bias from using cubic spline image interpolation in DVC. Using larger subsets helps to diminish this bias. Using higher order quintic splines would also help, at additional computational expense [71]. The corresponding standard deviations and error are shown in Figure 6.8, which show an optimal subset size of 31 voxels for the quadratic region.

Contours of the error for a single slice are plotted in Figure 6.9. Again, we see a large variability when using a small subset size (a) and small variability but a large error in the quadratic region when using a large subset size (c). We combine the solutions using 31 voxel subsets in the quadratic region and 51 voxel subsets in the linear regions to achieve a more accurate overall solution (d). The mean absolute error is reduced to 0.011 voxels for the combined solution, compared to 0.021 voxels for the 31 voxel subset and 0.013 voxels for the 51 voxel subset. The
Figure 6.7: Light green lines are multiple overlaid line scans of $u$ displacement for 3D model problem, each line for different $y$ in slice $z = 340$. Results for 3D model problem with 20 pixel wide quadratic region, going from zero strain in left region to 2% strain in right region.
standard deviation is reduced to 0.015 voxels for the combined solution, compared to 0.030 for the 31 voxel subset and 0.018 voxel for the 51 voxel subset.

### 6.4 Determining refinement region

We have demonstrated the need to refine the solution in areas of large strain gradients (e.g., regions with quadratic displacement fields) by using smaller subsets or quadratic shape functions. To determine these regions of large strain gradients, an appropriate refinement parameter should measure the local strain gradient in arbitrary directions. We propose using the norm of the second derivatives of the displacements, since it is zero in areas of constant strain and non-zero in areas with strain gradients. To discuss the tensor of second derivatives, we use index notation.
Figure 6.9: Error contours for one slice, $z = 340$, of 3D model problem defined in Figure 6.7. White areas have error less than 0.01 voxels. Dashed lines indicate boundaries of quadratic region. Combined graph (d) uses subset size 51 solution in linear regions and subset size 31 solution in quadratic region.
in this section, with

\[ u = [u_1, u_2, u_3]^T = [u, v, w]^T, \]
\[ x = [x_1, x_2, x_3]^T = [x, y, z]^T. \]

Let \( H \) be the tensor of second derivatives, with components

\[ H_{ijk} = \frac{\partial^2 u_i}{\partial x_j \partial x_k}. \]

We define the norm on \( H \) as

\[ \|H\| = \left( \sum_{i,j,k} H_{ijk}^2 \right)^{1/2}. \]

**Theorem.** The norm \( \|H\| \) is invariant with respect to coordinate frame.

**Proof.** Let \( R_{ip}, R_{jq}, \) and \( R_{kr} \) be rotation matrices defining a general 3D rotation as

\[ H_{ijk} = R_{ip} R_{jq} R_{kr} H_{pqr}. \]

Let \( H_{*,jk} \) be the \( j,k \) “column” of \( H \), with \( i = 1, \ldots, 3 \). The squared norm of \( H \) can be expressed as a summation of squared vector 2-norms,

\[ \|H\|^2 = \sum_{j,k} \|H_{*,jk}\|_2^2. \]

Since \( R_{ip} \) is orthogonal, when multiplied with a column of \( H \), the 2-norm of the column will not change,

\[ \|H_{*,jk}\|_2^2 = \|R_{ip} H_{*,jk}\|_2^2, \]

hence the matrix norm will be invariant under rotation by \( R_{ip} \),

\[ \|H_{iqr}\| = \|R_{ip} H_{pqr}\|. \]
A similar argument applies for $R_{jq}$ and $R_{kr}$.

Differentiating the displacement field measured experimentally using DVC yields a noisy second derivative field. We use smoothing splines again — now as a post-processing step to fit the displacement fields for $u$, $v$, and $w$. The smoothing splines are analytically differentiated to obtain the second derivatives. Figure 6.10 shows contours of the resulting $\|H\|$ for a single slice $z = 250$ of a 3D model problem. In this case, a quadratic field is applied in all three dimensions, so $u$, $v$, and $w$ all have quadratic regions that form a cross shape. The strains vary from $-2\%$ for $x, y, z < 500$ to $2\%$ for $x, y, z > 520$. With no smoothing (a), the cross shape is obscured by experimental noise. As the smoothing parameter increases to $\delta = 0.0005$ (b) and $\delta = 0.001$ (c), the cross shape becomes evident, indicating the region to refine. With a larger $\delta = 0.003$ (d), the second derivative loses information, showing an overly smooth solution. Currently we determine the amount of smoothing by manually adjusting the smoothing parameter to find a value that smooths out the noise in the image without destroying the salient features. To automate this process, methods such as generalized cross-validation [13, 32] or L-curve regularization [36] could be used. The exact values of the second derivatives compute by DVC will not be accurate, because the DVC method itself and the smoothing splines smooth the second derivatives out. This is especially true of large subset sizes, as was seen in Figure 6.7d where the $51^3$ subset cannot accurately capture the curvature of the exact displacement field. Therefore, the threshold to use for the $\|H\|$ to determine refinement depends on the particular experiment and DVC analysis, such as subset size and image noise, which influences the amount of smoothing required. Nonetheless, the norm of second derivatives indicates the regions containing strain gradients. Further experimentation may provide more definite guidelines for using this refinement parameter.
Figure 6.10: Contours of $\|H\|$ for one slice, $z = 250$, of problem with quadratic regions in $x$ and $y$ forming cross shape. Second derivatives computed using smoothing splines with varying amounts of smoothing. Dashed lines indicate boundaries of quadratic regions.
Chapter 7

Results and Discussion

We demonstrate the effectiveness of our enhanced, high-performance DVC code to detect a 3D deformation field based on a particular elastic solution. We first analyze baseline and rigid translation tests to measure the accuracy of DVC applied to our sample materials, then examine the deformation around a spherical inclusion in a material under uniaxial tension. As described in Section 2.1, we use a pair of baseline images and apply an artificial motion or deformation to one image using an analytical solution. We use two different sample materials, described more fully in Section 2.1. The first is PDMS with embedded silica particles scanned with the Skyscan MicroCT scanner at 10 \( \mu m \) resolution, resulting in a \( 1230 \times 1200 \times 900 \) voxel 3D image, pictured in Figure 7.1a. The second is ceramic foam scanned with the Xradia MicroCT scanner at 10 \( \mu m \) resolution, resulting in a \( 1013 \times 992 \times 994 \) voxel 3D image, pictured in Figure 7.1b. Results from Section 7.3.2 appeared in Gates et al. [25]; other results from this chapter are to appear in Gates et al. [27].
For both samples, we initially perform a baseline test between two scans with no applied motion or deformation between the scans, to determine the effects of noise introduced by the scanner and the accuracy of the DVC method. We compute an $11^3$ grid of 1331 correlation points, with 20 voxel spacing between correlation points in all dimensions, using a $31^3$ subset. For the PDMS/Skyscan sample, the average baseline measured displacements with one standard deviation error bars are $u = -0.13 \pm 0.15$, $v = -0.55 \pm 0.10$, and $w = -1.19 \pm 0.15$. The standard deviation for $v$ appears smaller than for $u$ and $w$; this discrepancy is accounted for in Section 7.2. As can be seen, the $v$ and $w$ displacements are clearly greater than the 0.15 voxel standard deviation. This means that the $v$ and $w$ values represent an actual half voxel and one voxel rigid motion, respectively, and are not really noise. This
motion, which DVC is able to capture, is likely because the tomograph rotation stage springs back during its 360 degree motion. As a more accurate alternative to fitting just rigid translations of the baseline DVC results, and as a comparison with the deformation results to follow, we also use a least squares fit to rigid translations and rotations, assuming small angles,

\[
\begin{align*}
    u_{\text{rigid}} + \theta_z Y - \theta_y Z & \equiv U, \\
v_{\text{rigid}} - \theta_z X + \theta_x Z & \equiv V, \\
w_{\text{rigid}} + \theta_y X - \theta_x Y & \equiv W,
\end{align*}
\]

where \( \theta_x \) is rotation about the \( x \) axis, \( \theta_z \) is about the \( z \) axis, \( \theta_y \) is about the \( y \) axis, and \( X, Y, Z, U, V, W \) are vectors of the \( x, y, z \) coordinates and \( u, v, w \) displacements, respectively, for all correlation points. Table 7.1a shows that we detected a rigid translation similar to the average results quoted above (repeated as the starred line in Table 7.1a). Upon closer inspection, the 1.2 voxel downward translation in the \( z \) direction was easily visible in the 3D images since it was clear that slice \( k \) in the first scan closely matched slice \( k - 1 \) in the second scan. The translations in \( x \) and \( y \) were smaller and rotations were small.

For the ceramic foam/Xradia sample, the average baseline measured displacements are \( u = 0.48 \pm 0.02, v = -0.09 \pm 0.07, w = 0.06 \pm 0.07 \). The smaller standard deviation for \( u \) is accounted for in Section 7.2. The standard deviation of 0.07 voxels for this sample is half that for the PDMS sample, which we attribute to both a better pattern in the sample and less image noise for the Xradia scanner than for the Skyscan scanner. The measured motion in \( u \) is greater than the 0.07 voxel standard deviation, indicating some motion occurred between scans due to physical perturbations in the CT scanner. A least squares fit to rigid translation and rotations is given in Table 7.1b; \( u \) and \( v \) are similar to the average values above (repeated as the
Table 7.1: Least squares fit of displacements to exact solution with rigid translation and small angle rotations. For spherical inclusion problem, exact applied tension $T$ is normalized to 1.
* starred baseline tests fit rigid translations only, not rotations.

(a) PDMS/Skyscan sample

<table>
<thead>
<tr>
<th>subset size</th>
<th>$u_{\text{rigid}}$</th>
<th>$v_{\text{rigid}}$</th>
<th>$w_{\text{rigid}}$</th>
<th>$\theta_x$</th>
<th>$\theta_y$</th>
<th>$\theta_z$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$31^3$</td>
<td>-0.13</td>
<td>-0.55</td>
<td>-1.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sphere</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$21^3$</td>
<td>-0.15</td>
<td>-0.46</td>
<td>-1.03</td>
<td>-3.0 x 10^{-4}</td>
<td>-2.4 x 10^{-4}</td>
<td>2.5 x 10^{-5}</td>
<td>1.00408</td>
</tr>
<tr>
<td>$31^3$</td>
<td>-0.09</td>
<td>-0.48</td>
<td>-1.24</td>
<td>-4.1 x 10^{-4}</td>
<td>-9.1 x 10^{-5}</td>
<td>-1.7 x 10^{-5}</td>
<td>1.00424</td>
</tr>
<tr>
<td>$41^3$</td>
<td>-0.08</td>
<td>-0.48</td>
<td>-1.26</td>
<td>-4.1 x 10^{-4}</td>
<td>-5.4 x 10^{-5}</td>
<td>-6.8 x 10^{-6}</td>
<td>1.00436</td>
</tr>
</tbody>
</table>

(b) Ceramic foam/Xradia sample

<table>
<thead>
<tr>
<th>subset size</th>
<th>$u_{\text{rigid}}$</th>
<th>$v_{\text{rigid}}$</th>
<th>$w_{\text{rigid}}$</th>
<th>$\theta_x$</th>
<th>$\theta_y$</th>
<th>$\theta_z$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$31^3$</td>
<td>0.48</td>
<td>-0.09</td>
<td>0.06</td>
<td>1.7 x 10^{-4}</td>
<td>-3.9 x 10^{-5}</td>
<td>1.3 x 10^{-6}</td>
<td></td>
</tr>
<tr>
<td>sphere</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$23^3$</td>
<td>0.47</td>
<td>-0.03</td>
<td>0.03</td>
<td>-9.6 x 10^{-6}</td>
<td>-8.8 x 10^{-6}</td>
<td>6.6 x 10^{-6}</td>
<td>1.00101</td>
</tr>
<tr>
<td>$31^3$</td>
<td>0.47</td>
<td>-0.03</td>
<td>0.03</td>
<td>-1.2 x 10^{-5}</td>
<td>-8.4 x 10^{-6}</td>
<td>1.1 x 10^{-5}</td>
<td>1.00084</td>
</tr>
<tr>
<td>$41^3$</td>
<td>0.47</td>
<td>-0.03</td>
<td>0.03</td>
<td>-1.4 x 10^{-5}</td>
<td>-1.1 x 10^{-5}</td>
<td>1.1 x 10^{-5}</td>
<td>1.00058</td>
</tr>
</tbody>
</table>

In addition to rigid body motion between successive scans, the image noise and contrast also affect the DVC results. The image noise we define as the standard deviation of the difference in gray levels between two baseline images. We first align the baseline images by subtracting the baseline rigid body motion determined
\begin{table}
\centering
\begin{tabular}{|l|c|c|}
\hline
Sample & Noise, gray levels (percent) & Contrast, gray levels \\
\hline
PDMS/Skyscan & 13.8 (5.4\%) & 15.7 \\
PDMS/Xradia & 20.2 (7.9\%) & 22.1 \\
Ceramic foam/Xradia & 30.3 (11.9\%) & 44.8 \\
\hline
\end{tabular}
\caption{Image noise and contrast for various samples.}
\end{table}

above, using cubic spline interpolation. For image contrast we use the root mean square (RMS) contrast,

\[
\text{contrast} = \left( \frac{1}{n} \sum_x (f(x) - \bar{f})^2 \right)^{1/2},
\]

for an image \( f \) of size \( n \) voxels with mean gray level \( \bar{f} \). As shown in Table 7.2, the PDMS/Skyscan sample had a noise of 13.8 gray levels, or 5.4\% of the 255 gray level range, and a contrast of 15.7 gray levels. A later scan of a PDMS sample with the Xradia scanner had a noise of 20.2 gray levels, or 7.9\%, and a contrast of 22.1 gray levels. The ceramic foam/Xradia sample had a noise of 30.3 gray levels, or 11.9\%, and a contrast of 44.8 gray levels. The contrast can be increased during the reconstruction, but this also increases the noise level. The two PDMS scans had noise levels that were around 90\% of the contrast, while the foam scan had a noise level that was 68\% of the contrast. The better contrast and better signal-to-noise ratio yield better DVC results for the ceramic foam sample compared to the PDMS sample, demonstrated by the smaller standard deviation in displacements observed earlier.

\section{7.2 Translation test}

To examine further the range of errors for these samples, we perform rigid translation tests where we artificially translate one of the baseline images by a fraction of a voxel in the \( x \) direction. We perform this test with the translation \( \Delta u = 0.1 \) to
Figure 7.2: Mean error and standard deviation for translation tests.
0.9 voxels in 0.1 increments, and use the baseline test itself for $\Delta u = 0$. Since the image is composed of voxel data at integer intervals, translations by whole voxels yield the same error; for example, $\Delta u = 0.1$ and $\Delta u = 1.1$ produce identical errors. Therefore, studying sub-voxel shifts over a unit interval is sufficient to understand arbitrary translations. As mentioned in Section 6.1, the cubic spline interpolation used in DVC introduces a systematic bias error that appears as a sinusoidal signal in Figure 7.2. The $u$ in Figure 7.2 includes the rigid body motion previously determined by the baseline test, for example, the ceramic foam baseline test appears as $u = 0.48$ in Figure 7.2b. Because we do not know the exact rigid translation between the baseline images — that is, we know the rigid translation only as measured by DVC, which includes this bias — the vertical position of the error curve is arbitrary. Since it is known that $u = 0.5$ has no bias [71], we assign the $u$ closest to 0.5 to have an error of zero. While the general shape of the bias error is always sinusoidal, the exact magnitude depends on the frequencies of the function being interpolated, and thus it depends on the image’s speckle pattern. The bias error can be reduced by using quintic splines [71]. We also found that post-processing results with smoothing splines reduces the bias error effectively. The standard deviation curves in Figure 7.2 also shows a dependence on the translation, with a larger standard deviation near $u = 0$ and $u = 1$, and a smaller standard deviation near $u = 0.5$. This dependence, which we observed experimentally for several cases, was not predicted by the previous standard deviation estimates [71, 75] discussed in Section 6.1. However, it may be accounted for by a discontinuity in the bias error for $\Delta u$ near integer voxel locations, as observed in Wang et al. [75, Figures 1, 2]. It explains the differences in standard deviation for $u, v,$ and $w$ observed earlier in Section 7.1. For the ceramic foam sample, the measured rigid translation $u = 0.48$, being near $u = 0.5$, has a small standard deviation, while the measured translations $v = -0.09$ and $w = 0.06$, being near zero, have larger standard devia-
For the PDMS/Skyscan sample, the bias error has a maximum magnitude of 0.05 voxels, which is smaller than the standard deviation of 0.15 voxels, so its effect is difficult to see in our experiments. From these translation tests, we conclude that DVC with this combination of sample, scanner, and parameters is accurate to within 0.2 voxels, accounting for the maximum bias error plus the maximum standard deviation. The error can be reduced in a number of ways: using larger subset sizes, quintic splines, or reducing image noise. We also address the noise by smoothing the results during post-processing in Section 7.3.3.

For the ceramic foam/Xradia sample, the bias error has a maximum magnitude of 0.08 voxels, similar to the standard deviation of 0.07 voxels, yielding an accuracy of 0.15 voxels. The effects of the cyclic bias error become apparent in the strain fields measured by DVC, as shown in Section 7.3.3. Again, we compensate for this error through post-process smoothing. These translation tests give us a better understanding of the range of errors that affect DVC results, enabling us to interpret the results better and know what to change to improve results.

### 7.3 Spherical inclusion test

#### 7.3.1 Overview of solution

For strain-inducing deformations, we use the 3D deformation field around a rigid spherical inclusion of radius $R$ in a linear elastic material under uniaxial tension, illustrated in Figure 7.3. We apply an artificial deformation to the first baseline image using the analytical solution given in equation (2.3), derived by Goodier [33], then compute DVC between this artificially deformed image and the second baseline image. By using an artificial deformation, we have the exact solution (2.3)
to compare with and are better able to assess the accuracy of the DVC method, apart from additional uncertainties inherent in using a load frame to perform \textit{in situ} load tests. We use a least squares fit between the displacements measured with DVC and the exact solution to determine the rigid body motion and the applied load $T$,

\[
\begin{align*}
u_{\text{rigid}} + \theta_z \mathbf{Y} - \theta_y \mathbf{Z} + TU_{\text{exact}} & \approx \mathbf{U}, \\
v_{\text{rigid}} - \theta_x \mathbf{X} + \theta_z \mathbf{Z} + TV_{\text{exact}} & \approx \mathbf{V}, \\
v_{\text{rigid}} + \theta_y \mathbf{X} - \theta_x \mathbf{Y} + TW_{\text{exact}} & \approx \mathbf{W}.
\end{align*}
\]

The results shown in Table 7.1 for the sphere agree well with the rigid body motion detected for the baseline scan using the same original pair of images. We also detect the applied tension $T$ with 0.4\% relative error for the PDMS sample and 0.1\% relative error for the ceramic foam sample. For all error measurements and plots shown subsequently, we subtract this rigid body motion from the DVC solution.

Figure 7.4 shows vector plots of the 3D displacement field measured by DVC around the sphere, viewed from just off the $y$ axis so that effectively we see just the $u$ and $w$ displacements. Tension is applied in the $x$ direction, and the Poisson effect causes compression in the $y$ and $z$ directions. At the scale in Figure 7.4a, the

\begin{center}
\textbf{Figure 7.3:} Spherical inclusion in medium under tension.
\end{center}
Distortion caused by the sphere is not readily visible. Vectors in each row parallel to the \( y \) axis appear to have uniform length (i.e., same \( u \) and \( w \)). This is expected, since the sphere will have negligible influence on far-field displacements, so the solution will appear as that of a homogeneous solid under uniaxial tension. When the region near the sphere is viewed more closely in Figure 7.4b, we see how the displacement field is distorted close to the sphere. Within each row, displacements near the sphere are smaller than those farther from the sphere, approaching zero displacement at the surface of the sphere.

Figure 7.5 shows isosurfaces of DVC computed values for \( u \), \( v \), and \( w \), showing zero displacement along the midplanes of the sphere and a bulge in the displacement isosurface near the sphere, flattening to a linear displacement field far from the sphere. Note that since the signal-to-noise ratio is higher for the \( u \) values, they are captured better, but nonetheless the \( v \) and \( w \) values are also captured in these results.

### 7.3.2 Initial analysis

Far from the sphere, Saint-Venant’s principle implies that the effect of the inclusion will be negligible, so the strains should be constant. By computing correlation points along the \( x \), \( y \), and \( z \) axes, we can measure these far-field strains in all three directions accurately. Figure 7.6 shows the \( u \) displacement along the \( x \) axis, the \( v \) displacement along the \( y \) axis, and the \( w \) displacement along the \( z \) axis. The straight lines are a least squares fit for points farther than \( 4R \) from the center of the sphere of radius \( R \). The slopes of these lines give the far-field strains \( \frac{\partial u}{\partial x} \), \( \frac{\partial v}{\partial y} \), and \( \frac{\partial w}{\partial z} \). The ratio of \( \frac{\partial u}{\partial x} \) to \( \frac{\partial v}{\partial y} \) and \( \frac{\partial w}{\partial z} \) should give Poisson’s ratio, \( v \). The table in Figure 7.6 summarizes the slopes and the resulting Poisson ratio. We could determine Poisson’s ratio with less than 5% error for the PDMS sample. The exact Poisson’s ratio was
Figure 7.4: Arrow plot of displacements around spherical inclusion. Results using $41^3$ subset with ceramic foam sample. $R$ is radius of sphere.
Figure 7.5: Isosurfaces of $u$, $v$, and $w$ displacements, showing how they bulge outward near sphere and tend towards flat surfaces far away from sphere. Results using $41^3$ subset with ceramic foam. Applied loading is in $x$ direction. $R$ is radius of sphere.
Figure 7.6: Displacement along $x$, $y$, and $z$ axes for PDMS sample. Straight line fit to data farther than $4R$ from center of sphere (dashed line) yields good measurement of Poisson’s ratio, $v$. Exact $v = 0.49$.

<table>
<thead>
<tr>
<th></th>
<th>slope</th>
<th>$v$</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u$</td>
<td>0.0357</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$v$</td>
<td>-0.0167</td>
<td>0.467</td>
<td>4.6%</td>
</tr>
<tr>
<td>$w$</td>
<td>-0.0174</td>
<td>0.488</td>
<td>0.4%</td>
</tr>
</tbody>
</table>

input as 0.49 in equation (2.3) when applying the artificial deformation.

To see the effects of noise from the CT scanner, we compare tests using a single image and using a pair of images. In the single-image case, we take a single scan, apply an artificial deformation, and then perform DVC between the artificially deformed image and the original image. Because there is no noise difference between these images and no rigid motion, we obtain unrealistically good DVC results, which nonetheless give a lower bound on the achievable accuracy and insight into how subset size affects accuracy. Table 7.3a shows that the error for the PDMS sample without noise is less than 1% and increases slightly with larger sub-
### Table 7.3: Error in displacements, after subtracting rigid body motion, and error in least squares fit to applied tension $T$, for various subset sizes. Table (a) uses one image both for reference image and to generate deformed image, so has no noise. Tables (b) and (c) use two baseline images, one as reference and one to generate deformed image, so include CT scanner noise and rigid motion.

<table>
<thead>
<tr>
<th>Subset size</th>
<th>Mean abs. error in $u$</th>
<th>Relative $L_2$ error in $u$</th>
<th>Relative error in $T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) PDMS, without noise</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$21^3$ voxels</td>
<td>0.029 voxels</td>
<td>0.65%</td>
<td>0.021%</td>
</tr>
<tr>
<td>$31^3$ voxels</td>
<td>0.036 voxels</td>
<td>0.83%</td>
<td>0.027%</td>
</tr>
<tr>
<td>$41^3$ voxels</td>
<td>0.042 voxels</td>
<td>0.97%</td>
<td>0.015%</td>
</tr>
<tr>
<td>(b) PDMS, with noise</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$21^3$ voxels</td>
<td>0.54 voxels</td>
<td>12.26%</td>
<td>0.40%</td>
</tr>
<tr>
<td>$31^3$ voxels</td>
<td>0.21 voxels</td>
<td>4.67%</td>
<td>0.42%</td>
</tr>
<tr>
<td>$41^3$ voxels</td>
<td>0.16 voxels</td>
<td>3.77%</td>
<td>0.44%</td>
</tr>
<tr>
<td>(c) Ceramic foam, with noise</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$23^3$ voxels</td>
<td>0.12 voxels</td>
<td>2.35%</td>
<td>0.10%</td>
</tr>
<tr>
<td>$31^3$ voxels</td>
<td>0.08 voxels</td>
<td>1.41%</td>
<td>0.08%</td>
</tr>
<tr>
<td>$41^3$ voxels</td>
<td>0.06 voxels</td>
<td>1.11%</td>
<td>0.06%</td>
</tr>
</tbody>
</table>

This can be seen in the $u$ contour plots in Figures 7.7a and 7.7b, where contours for the smaller $31^3$ subset more accurately determine displacements near the sphere, while the $41^3$ subset averages strain over a larger window, so it cannot detect quickly changing strain as effectively. This is consistent with our observations in Chapter 6 that in a quadratic displacement field, the error using a linear shape function grows as the subset size becomes large. The solid contour lines are DVC results, while the dashed contour lines are the exact solution. The circle shows the position of the spherical inclusion.

In the case of a pair of images, we artificially deform the first baseline image, then compare it to the second baseline image. This includes the noise and any rigid motion inherent in using the CT scanner, and better reflects the accuracy obtainable for a mechanical load test. Table 7.3b shows the errors for PDMS are about
Figure 7.7: Contours of $u$ displacements at 0.5 voxel increments for plane $z = \frac{2}{5}R$, with PDMS sample. Dashed lines are exact solution. Circle is position of spherical inclusion. (a) and (b) use single image as reference image and to generate deformed image, so have no noise and produce unrealistically good results. (c) and (d) use one image as reference image and second image to generate deformed image, so include noise inherent in CT scanner and produce more realistic results.
Figure 7.8: Contours of $v$ and $w$ displacements at 0.25 voxel increments, corresponding to $u$ displacements in Figure 7.7d.

an order-of-magnitude larger than the PDMS test without noise, and decrease as the subset size increases, the opposite trend from the PDMS test without noise in Table 7.3a. The ceramic foam results in Table 7.3c exhibit significantly smaller errors than the PDMS results in Table 7.3b, but have the same trend that the error decreases with larger subsets and the error is larger than the PDMS test without noise. This demonstrates that, across the entire displacement field, the absolute error is dominated by the standard deviation $\sigma_{ur}$, described in Section 6.1. We will see in Section 7.3.3 that the error near the sphere is dominated by the mismatch between the linear shape function and the nonlinear displacement field. However, most of the error is attributable to noise in the images from the CT scanner, rather than the DVC method itself. With the Skyscan, we used image averaging during the CT scan to reduce noise, capturing 20 X-ray images per angle and averaging them. These averaged X-ray images are then used to reconstruct the 3D image. We expect averaging a larger number frames would reduce noise further, at the
cost of longer scan times. The Xradia scanner has a variable exposure time, which acts in a similar manner to reduce noise by averaging image intensity over time. We use an exposure of three seconds per X-ray image. Figures 7.7c and 7.7d compare the displacement contours for 31×3 and 41×3 subsets with PDMS, showing the smoother solution obtained with the larger subset. The v and w displacements for the 41×3 subset are shown in Figure 7.8, corresponding to the u displacements in Figure 7.7d. Again we see that the u values are captured better due to their higher signal-to-noise ratio, but we also capture v and w values.

We tested the effect of the image interpolation scheme by performing DVC with the PDMS sample, using C¹ tricubic interpolation, C² tricubic spline interpolation, and C² smoothing spline approximation. To make results comparable, we did not perform a coarse search but instead used the same initial guess for all three methods, which was on average 1.3 voxels from the computed minimum. Using C¹ tricubic interpolation required an average of 18.2 BFGS iterations per correlation point and had a relative error of 4.11%; tricubic spline interpolation required 15.3 iterations, a 16% improvement, and improved the error to 3.63%; smoothing spline approximation required 13.3 iterations, an additional 11% improvement, with an error of 3.57%.

7.3.3 Refinement and smoothing

To analyze how to refine our results, we compute solutions with a 23×3 subset and a 41×3 subset, both using our target 101×3 grid with over a million correlation points. The 23×3 solution was computed in 2 hours using 32 processors on Turing, for 64 CPU-hours total, while the 41×3 solution was computed in 4.7 hours using 64 processors on Turing, for 300 CPU-hours total. Both use the ceramic foam sample.

We post-process the DVC results by fitting tricubic smoothing splines to the
measured $u$, $v$, and $w$ displacement fields. Note this is different than our earlier use of smoothing splines for fitting image intensity during the DVC process; here we use smoothing splines for fitting displacements after the DVC process. The use of smoothing splines reduces the errors in measured displacements, revealing the underlying displacement field. This is particularly helpful in computing accurate derivatives for the strain fields, which are inherently more sensitive to noise than displacements. Splines offer a continuous representation of the displacement field that can be evaluated at any point. However, to give a fair comparison with raw DVC results and avoid a false sense of smoothness, we restrict ourselves to evaluating the splines at the correlation grid points. Derivatives are computed by analytically differentiating the B-spline basis functions and evaluating the spline at grid points.

As discussed in Section 4.3, the amount of smoothing can be controlled by the smoothing parameter $\delta$. Currently we adjust this parameter manually to find a fit that eliminates noise as much as possible while still yielding a faithful representation of the displacement field. Figure 7.9 shows the $v$ displacement field for a single slice $z = \frac{2}{3}R$ with different values of the smoothing parameter. The thin blue lines are contours of the original DVC results and the thick red lines are the smoothed results. A $\delta$ that is too small results in a solution that still contains significant error, shown in Figure 7.9b. A $\delta$ that is too large smooths the solution too much, eliminating important features of the displacement field near the sphere, as shown in Figure 7.9c. Choosing an optimal value for $\delta$ reduces the noise while preserving the shape of the displacement field, shown in Figure 7.9d. The corresponding $\frac{\partial v}{\partial y}$ strain fields are shown in Figure 7.10, which shows clearly how features are disguised by noise in (a) and (b), are smoothed too much in (c), and are best visible in (d). The optimal value for $\delta$ is different for every DVC analysis. Values for the ceramic foam sample differ by several orders of magnitude, depending on the subset.
Figure 7.9: Contours of displacement field $v$ with various levels of smoothing. Thin blue lines are raw DVC results. Thick red lines are smoothed results.
Figure 7.10: Contours of strain field $\frac{\partial v}{\partial y}$ corresponding to Figure 7.9.
size and grid spacing, from $\delta = 0.0003$ for $41^3$ subsets to $\delta = 0.015$ for $23^3$ subsets. The $u$, $v$, and $w$ fields for a single DVC analysis may also require different values, though in our experience are generally within a factor of four. We first determine $\delta$ to obtain a good fit for the $u$ displacement field, then try this same $\delta$ for $v$ and $w$, adjusting it as necessary to obtain good fits. For the spherical inclusion problem, we found $v$ and $w$ often needed slightly more smoothing than $u$, which is probably due to the higher signal-to-noise ratio for $u$, since tension is applied in the $x$ direction. This process of determining an optimal smoothing parameter can potentially be automated by using generalized cross-validation [13, 32] or L-curve regularization [36], which provide objective estimates of how well a smoothing spline fits the underlying function.

In Chapter 6 we concluded that large subsets capture the deformation field accurately in areas of constant strain, while smaller subsets or higher order shape functions are required to measure the deformation accurately in areas of large strain gradients. For the inclusion problem, this is demonstrated in Figure 7.11, which shows the error in voxels of the $u$ displacement field (top row) and $v$ displacement field (bottom row) for a single slice $z = \frac{2}{5}R$. White areas have an error less than 0.025 voxels. The left plots (a, c) use a $23^3$ subset; the right plots (b, d) use a $41^3$ subset. All four graphs use a 5-voxel grid spacing for easy comparison. Results are smoothed using smoothing splines with the $\delta$ parameter indicated in the figure. Near the inclusion, using a $23^3$ subset yields more accurate results, while far from the sphere, using a $41^3$ subset gives less noisy and more accurate results.

Looking at contour lines of the $u$ displacements in Figure 7.12, we see that the $23^3$ subset solution (a, c) accurately follows the exact solution, while the $41^3$ subset (b, d) smooths the solution too much, making the contour lines straighter. The far-field displacements in Figure 7.12a and b appear similar at this scale, though as previously shown in Figure 7.11, the $23^3$ solution has more noise.
Figure 7.11: Error in $u$ and $v$ displacement fields after smoothing with given $\delta$ parameter. White areas have less than 0.025 voxel error.
Figure 7.12: Displacement field $u$ after smoothing, showing both far-field and close-up views.
The noise difference between the $23^3$ and $41^3$ subsets is evident in the $v$ displacement field shown in Figure 7.13a and b. Since tension is applied in the $x$ direction, the $v$ field has a lower signal-to-noise ratio, hence more noise, than the $u$ field. For the $23^3$ subset, fitting smoothing splines significantly reduces this noise, as shown in Figure 7.13c. A close-up view of the $v$ displacement field in Figure 7.14d again shows that the $41^3$ subset smooths out the solution, reducing the curvature of the contour lines, though to a much less extent than with the $u$ displacement field in Figure 7.12d, since the $\frac{\partial v}{\partial y}$ strain is smaller than the $\frac{\partial u}{\partial x}$ strain. The $23^3$ subset (c) is able to capture the displacement field near the sphere more accurately.

The real advantage of fitting smoothing splines is shown in the strain fields. Figures 7.15 and 7.16 show far-field and close-up views of the $\frac{\partial u}{\partial x}$ strain. The top row (a, b) shows contours of the raw DVC results. The $23^3$ solution (a) has a significant amount of noise, which obscures the underlying features. The vertical banding in the raw results is due to the sinusoidal interpolation bias error discussed previously in Section 7.2. The cyclical error shown in Figure 7.2 is repeated once for every unit increment of $u$ in Figure 7.15a,b. Since far from the sphere the displacement $u$ varies linearly with $x$, this error causes the regular vertical banding. In the region shown, $u$ varies from $-7.5$ to $7.5$ (see Figure 7.12), so 15 cycles of vertical banding are visible in Figure 7.15a,b. Most of this banding is removed through the use of smoothing splines, shown in the middle row (c, d). After smoothing, the $23^3$ solution (c) still has some noise, but important features near the sphere are resolved. The $41^3$ solution (d) reveals the far-field strain behavior more accurately, but near the sphere it cannot resolve features well. Looking at the close-up view in Figure 7.16, the $23^3$ solution (c) reveals areas of high strain to the left and right of the sphere, and areas of low strain near $(\pm R, \pm R)$. The shape of the high strain regions is slightly concave towards the sphere. These features correspond well with the exact solution shown in (e). The $41^3$ subset (d) is unable to resolve the areas of
Figure 7.13: Displacement field $v$, before smoothing (a, b) and after smoothing (c, d).
Figure 7.14: Close-up view of displacement field $v$ from Figure 7.13.
Figure 7.15: Strain field $\frac{\partial u}{\partial x}$, computed as derivative of interpolating spline (a,b,e) or smoothing spline (c,d).
Figure 7.16: Close-up view of strain field $\frac{\partial u}{\partial x}$, as in Figure 7.15.
low strain. It does capture the two areas of high strain, but measures a maximum strain level less than the true value, seen here by one fewer contour line than in the $23^3$ and exact solutions, and the position of these strain concentrations is moved farther away from the sphere, though the smoothed version accentuates this error. The shape of the strain concentrations is rounded rather than being concave.

Similar features are seen when we examine the $\frac{\partial v}{\partial y}$ strain in Figure 7.17. The horizontal banding evident in raw DVC results (a,b) is again due to the interpolation bias error, and its effects are largely removed by the smoothing spline (c,d). The $41^3$ smoothed solution (d) has less noise in the far-field strain, but it doesn’t capture the magnitude of the strain concentrations near the sphere, while the $23^3$ smoothed solution (c) captures the magnitude of these concentrations better.

Quadratic shape functions with large subsets offer another means to resolve accurately the regions with large strain gradients near the sphere, as an alternative to using small subsets with linear shape functions. Figure 7.18 shows the $u$ displacement field (a), $v$ displacement field (b), and two views of the $\frac{\partial u}{\partial x}$ strain field (c, d) using a $41^3$ quadratic subset. The close-up results in (a, b, c) are computed over the region $x, y, z \in [-\frac{5}{2}R, \frac{5}{2}R]$ with a 5-voxel grid spacing, and are smoothed with $\delta = 0.0015$ for $u$ and $\delta = 0.004$ for $v$ and $w$. The far-field results in (d) are computed over a larger region, $x, y, z \in [-\frac{9}{2}R, \frac{9}{2}R]$, with a coarser 10-voxel grid spacing, and are smoothed with $\delta = 0.003$ for $u$ and $\delta = 0.0045$ for $v$ and $w$. The results are qualitatively similar to the earlier results using a $23^3$ subset shown in Figures 7.12c, 7.13c, 7.15c, and 7.16c. Particularly, the quadratic solution reveals the regions of low strain and high strain near the sphere, and the shape of the high strain region is slightly concave towards the sphere. The far-field strain in 7.18d shows a similar amount of noise compared with the $23^3$ solution in 7.16c. Indeed, the striking similarity of the far-field strain suggests that the errors are systematic, caused by either interpolation bias error or properties of the ceramic foam image texture.
Figure 7.17: Strain field $\frac{\partial v}{\partial y}$, computed as derivative of interpolating spline (a,b,e) or smoothing spline (c,d).
Figure 7.18: Displacement and strain fields using $41^3$ quadratic subset, after smoothing.
To make a quantitative comparison between results with various subset sizes and shape functions, we report the mean absolute error of $u$ in Table 7.4. The error is computed for two separate regions: near the sphere, defined as correlation points such that $R \leq \|x\| \leq 2R$, and far from the sphere, defined as correlation points such that $2R < \|x\|$. We show errors for the raw DVC results and after smoothing the results. Smoothing reduces the error in nearly all cases, except for the $41^3$ linear subset near the sphere. In this case, the spline may be smoothing too much near the sphere, where the $41^3$ solution is already the worst approximation, so we will prefer another solution there. Near the sphere, the error increases with subset size, except for the quadratic shape function, which is very close to the $23^3$ solution. Far from the sphere, the $41^3$ linear subset has the smallest error. With the same $41^3$ subset size, the quadratic shape function has a higher error than the linear shape function, consistent with our earlier results in Section 6.2. These observations suggest that an optimal solution combines the use of large subsets and linear shape functions in the regions of constant strain far from the sphere, with small subsets or quadratic shape functions in the regions of large strain gradients near the sphere, supporting our conclusions in Chapter 6.

Thus far we have examined the effects of subset size and shape function on the accuracy of the solution. Often in simulation methods such as the finite element method (FEM), adaptive refinement is done to achieve accurate results in a
reasonable computation time; uniform refinement everywhere would produce an accurate solution but is too expensive. For DVC, in contrast, a uniform choice of subset size and shape function will not produce the most accurate solution. Therefore, it is important to understand the accuracy issues to know how to refine the solution. The computational cost is also an important consideration. The average time for DVC to compute a single correlation point for the ceramic foam sample, with 5-voxel grid spacing, is given in Table 7.5. As previously discussed in Section 6.1, using the half-subset correlation grid spacing of half the subset size results in computation time that is roughly constant, independent of the subset size, here 5–6 CPU-hours. The time per point grows cubically with the subset size, and hence, so does the time for a uniform correlation grid with 5-voxel spacing for all subset sizes. The quadratic shape function is 1.75 times as expensive as the linear shape function for the same subset size. Utilizing parallel computing, the large number of CPU-hours can be performed in a reasonable wall clock time. For instance, in Section 5.3, we computed the $101^3$ uniform grid using $31^3$ subsets in 1.15 hours using 128 processors on Turing. However, a faster and more accurate solution can be obtained by refining the solution adaptively instead of using uniform refinement.

<table>
<thead>
<tr>
<th>subset size and shape function</th>
<th>seconds per point</th>
<th>half-subset grid dimensions</th>
<th>estimated CPU-hours</th>
<th>5 voxel grid dimensions</th>
<th>estimated CPU-hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>$23^3$ linear</td>
<td>0.219</td>
<td>$46^3$</td>
<td>5.9</td>
<td>$101^3$</td>
<td>63</td>
</tr>
<tr>
<td>$31^3$ linear</td>
<td>0.493</td>
<td>$34^3$</td>
<td>5.4</td>
<td>$101^3$</td>
<td>141</td>
</tr>
<tr>
<td>$41^3$ linear</td>
<td>1.043</td>
<td>$26^3$</td>
<td>5.1</td>
<td>$101^3$</td>
<td>299</td>
</tr>
<tr>
<td>$41^3$ quadratic</td>
<td>1.823</td>
<td>$26^3$</td>
<td>8.9</td>
<td>$101^3$</td>
<td>521</td>
</tr>
</tbody>
</table>

Table 7.5: Average time per correlation point to compute DVC results on Turing and estimated time to compute DVC on $500^3$ region using given grid dimensions.
7.3.4 Final analysis

For an optimal solution, we compute the solution with a $41^3$ linear subset at 20 voxel grid spacing, yielding a $26^3$ grid with 17,576 correlation points. With 16 processors on Turing this took 29 minutes, for a total 7.7 CPU-hours. We fit smoothing splines to this solution and examine the norm of the second derivative, shown in Figure 7.19, to determine where to refine the solution. This suggests computing a refined solution in the region $x, y, z \in [-2R, 2R]$ around the sphere. Using a 5 voxel spacing within this region yields a $41^3$ refined grid with 68,921 correlation points.

To compute DVC on this refined grid with a $23^3$ linear subset, the estimated time is 4.2 CPU-hours, while for a $41^3$ quadratic subset the estimated time is 34.9 CPU-hours. Using 4 processors, the $23^3$ solution can be done in about one hour, while the $41^3$ quadratic solution requires around 36 processors to reach our one-hour goal. For this sample, a $23^3$ subset is clearly preferred because of the significant computational savings and similar error compared to a $41^3$ quadratic subset. To combine solutions, we up-sample the $41^3$ linear solution by evaluating the splines fitting $u, v,$ and $w$ at a 5 voxel grid spacing, then copy the $23^3$ solution onto that grid. There are small discrepancies between the two solutions at their interface, which we smooth by fitting a smoothing spline with $\delta = 1 \times 10^{-6}$. Smoothly interpolating between the two solutions at their interface is another potential means to deal with this discrepancy. The $\frac{\partial u}{\partial x}$ and $\frac{\partial v}{\partial y}$ strains for the combined solution are shown in Figure 7.20 (compare with Figures 7.15 and 7.17, respectively). This solution combines the benefits that we have observed for the two subset sizes: near the sphere, the strain gradients are captured well, while far from the sphere, the solution has little noise. While this combination works well for the ceramic foam sample, the choice of subset size and shape function depends on the sample being analyzed. For instance, with the PDMS sample, $21^3$ subsets incurred substantial error and were unusable.
Figure 7.19: Norm of second derivatives, $\|H\|$, for $41^3$ subset, after smoothing.

due to the sparser speckle pattern. In that case, using a large subset with quadratic basis functions may be preferable despite the computational expense.

Note that the far-field strains obtained here in Figure 7.20 using $41^3$ subsets with a 20 voxel grid spacing looks similar to the strain previously obtained using $41^3$ subsets with a 5 voxel grid spacing in Figure 7.15d and Figure 7.17d. Only in the region near the sphere is the fine, 5 voxel grid spacing necessary to resolve the strain gradients that occur there. The $41^3$ subset coarse grid solution cost 7.7 CPU-hours, compared to 300 CPU-hours for the $41^3$ subset fine grid solution. Therefore, using a coarse grid for the initial $41^3$ solution, we save computational time compared to using a fine grid everywhere, without sacrificing accuracy.
Figure 7.20: Strain fields for refined solution with $23^3$ used in $x, y, z \in [-2R, 2R]$ and $41^3$ subsets used elsewhere.
Chapter 8

Conclusions

We developed a digital volume correlation method that uses linear or quadratic shape functions, interpolating or smoothing tricubic splines, and is scalable to large problem sizes. To make large-scale 3D DVC problems tractable we developed and implemented numerous improvements to the DVC algorithm. Using this method, we performed DVC with over an order-of-magnitude more correlation points than previously published studies, shown in Table 1.1.

We developed improvements to the coarse search algorithm to make it faster, by extrapolating the displacement from previously computed solutions, and more reliable, by searching a large area when necessary using either the expanding search or the hybrid expanding and FFT-based search. The large search region allows each parallel processor to determine its initial guess independently and reduces or eliminates the need for the user to provide an initial starting guess.

We determined scaling factors that improve the convergence rate of BFGS and steepest descent, and were necessary for convergence with quadratic shape functions. We compared the performance of various optimization algorithms and found BFGS with appropriate scaling to be efficient and reliable. Levenberg-Marquardt was also reliable, though in our tests took about twice as long as BFGS.
An analysis of different spline basis functions leads us to recommend the B-spline basis due to its smaller memory requirements, which enables an efficient implementation. The B-spline basis is also amenable to computing smoothing splines. We have shown how to optimize the spline implementation for DVC, for instance by taking advantage of the fixed-width spacing of image data. The recursive definition of the B-spline basis also makes it extendable to higher orders, such as quintic splines. Since the interpolation bias error became evident in later testing, we feel that it would be worthwhile to investigate implementing quintic splines, to assess the performance cost and the increase in accuracy. Before implementing quintic splines, a more detailed profiling of the DVC code could be used to determine the percentage of time spent in the spline evaluation and hence to estimate the overall cost of changing from tricubic to triquintic splines.

Our 3D image data structure loads just the image slices currently in use, making our DVC implementation scalable to large problem sizes using a limited amount of memory. By computing splines over each subset independently, we decouple the computation of correlation points, allowing us to implement DVC in parallel. Our MPI-based code scales nearly linearly, with the dynamic load balancing scheme achieving greater than 90% parallel efficiency on 128 processors. Using this parallel code, we solved our target resolution of over a million correlation points in 69 minutes on 128 processors, achieving our goal of computing DVC in time commensurate with image acquisition time.

We also implemented a hybrid GPU and MPI parallel code that simultaneously does both coarse-grained parallelism to compute multiple correlation points across multiple CPUs and, for each CPU, fine-grained parallelism using a GPU to parallelize the computation of a single correlation function. This code achieved a 5.9 times speedup for a single CPU and GPU compared to a CPU-only code, and 20.3 times speedup using four CPU cores and a single GPU. We estimate it can solve
our target million-point problem in 3.1 hours on four CPU cores with one GPU. Other portions of the DVC code are also amenable to parallelization using GPUs, for instance, computing the spline coefficients, which should further increase the speedup using a GPU. We also want to test our GPU code on large problems using a cluster with several GPUs.

An examination of the error due to image noise and due to the linear or quadratic shape function approximating the nonlinear deformation field leads to the conclusion that a combination of subsets are required for best accuracy: large linear subsets in areas of constant strain, and either small linear subsets or large quadratic subsets in areas with large strain gradients. To our knowledge, this is the first DVC code to implement quadratic shape functions, bringing 3D DVC to the same level as 2D DIC. Given a particular image, tests using a quadratic model problem as in Chapter 6 can be used to determine optimal subset sizes based on the expected strain rates. We showed that the norm of the second derivatives was an effective measure to determine where to refine the solution. Further experimentation with different samples and deformation fields will hopefully improve the use of this parameter, towards the eventual goal of automating the refinement process.

To deal with a noisy displacement field, we use smoothing splines to fit the displacements, and demonstrated the benefits of computing the strain field from the smoothing splines, including reducing noise and reducing effects of the interpolation bias. For fitting displacement fields with smoothing splines, thus far we have manually determined the amount of smoothing. Methods exist to determine the smoothing parameter automatically, so research into their application to DVC results would be valuable. This will make the smoothing process more objective and help to automate the refinement process, which uses the smoothing splines to compute the second derivatives.

Our final, refined solution initially uses a large subset on a coarse grid to provide
a smooth solution across the whole image, solved in half-an-hour with 16 processors. Smoothing splines are fit to the displacements to reduce noise. Regions with large strain gradients that require refinement are identified by examining the norm of the second derivatives, again computed from the smoothing splines. A refined solution is then computed in those areas, in our case using smaller linear subsets, solved in an hour with four processors. This adaptively refined solution is accurate both in regions of large strain gradients, near the sphere, and also in the far-field regions with constant strain. By comparison, a solution with the same subset everywhere will not be as accurate in both regions simultaneously. An adaptively refined solution is also less computationally expensive than a uniformly refined solution.

For this work, we used CT scans of several materials, but applied artificial deformations so that we could test the accuracy of the DVC method. A compression load frame for use in the Xradia MicroCT scanner was recently designed and built by Prof. Lambros and Rebecca Mudrock. This provides us with the capability to perform mechanical load tests of materials. With a scalable, robust, efficient DVC implementation, we can now explore applications of DVC to investigate actual load tests, with resolution not previously achievable.
Appendix A

PDMS sample preparation

1. Sieve silica particles to obtain desired particle size range.

2. Prepare mold.

3. Using electronic scale, measure PDMS base, linker, and silica particles specified in Table A.1 into mixing container (e.g., paper cup). Dispense linker with bulb pipette for better control. Proportions we used are listed in Table A.2.

4. Mix with stir rod (e.g., tongue depressor).

5. Place container in vacuum chamber to degas for approximately 5 minutes, until bubbles stop forming.

6. Remove from vacuum chamber.

7. Using electronic scale, measure catalyst into mixing container with bulb pipette.

8. Stir for a few seconds, taking care to avoid introducing air bubbles.

9. Pour into mold. The PDMS we used solidifies in less than a minute.

10. Let cure fully overnight.

11. Remove from mold.
<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silica particles</td>
<td>Glass beads</td>
<td>CAS 65997-17-3</td>
</tr>
<tr>
<td>Base</td>
<td>Silanol terminated polydimethylsiloxane, viscosity 700–800 cSt.</td>
<td>DMS-527-100GM</td>
</tr>
<tr>
<td>Linker</td>
<td>Poly(diethoxysiloxane)</td>
<td>PSI-021-100GM</td>
</tr>
<tr>
<td>Catalyst</td>
<td>Tin II Oleate 85%</td>
<td>SNT7955-100GM</td>
</tr>
</tbody>
</table>

**Table A.1:** Chemicals used for PDMS samples. Glass beads were obtained from McMaster-Carr Supply Co., Elmhurst, IL. PDMS base, linker, and catalyst were obtained from Gelest, Inc., Morrisville, PA.

<table>
<thead>
<tr>
<th>Particle size</th>
<th>Base</th>
<th>Linker</th>
<th>Silica particles</th>
<th>Catalyst</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 53 $\mu$m</td>
<td>9.96g</td>
<td>2.07g</td>
<td>1.96g</td>
<td>0.5g</td>
</tr>
<tr>
<td>53–75 $\mu$m</td>
<td>10.30g</td>
<td>2.06g</td>
<td>2.03g</td>
<td>0.5g</td>
</tr>
<tr>
<td>75–125 $\mu$m</td>
<td>10.17g</td>
<td>2.02g</td>
<td>2.09g</td>
<td>0.5g</td>
</tr>
<tr>
<td>125–180 $\mu$m</td>
<td>9.96g</td>
<td>2.08g</td>
<td>2.16g</td>
<td>0.5g</td>
</tr>
<tr>
<td>180–250 $\mu$m</td>
<td>10.09g</td>
<td>2.00g</td>
<td>1.96g</td>
<td>0.5g</td>
</tr>
</tbody>
</table>

**Table A.2:** Quantities used for PDMS samples.
Appendix B

Skyscan MicroCT procedure

B.1 Scanning

1. Turn on Skyscan, login to computer, and launch Skyscan software.

2. Mount sample on a stage using putty.

3. Use software to open scanner door, then tighten stage into spindle and close scanner door.

4. Turn on X-ray source in software. It takes approximately 15 minutes to warm up.

5. Raise sample into view in software.

6. Right-click image to get X-ray intensity histogram. The desired background level is about 90%.

7. Set acquisition pixel size. We used medium pixels and 10 μm zoom.

8. Adjust camera gain using supervisor mode (control-alt-shift S, H). We used 83% gain.

9. Adjust voltage in Options menu, X-ray source. We used 100 kV.

10. Lower sample out of view and perform bright-only flat field calibration using supervisor mode.

11. Raise sample again.
12. Set scan parameters and initiate scan. We averaged 20 frames and used 360° rotation with 0.4° increments. Do not turn X-ray off between scans of the same material, to avoid resetting scanner settings. Turn X-ray off after last scan.

13. Save X-ray images to network drive for reconstruction.

14. Open scanner door, remove sample, close door, quit software, and logout.

B.2 Reconstruction

1. Login to reconstruction computer and launch NRecon software.

2. Open an X-ray image out of set to be reconstructed.

3. Perform preview.

4. Adjust post-alignment as required to avoid double images. This varies on a case-by-case basis. We often used the automatically determined value.

5. Adjust beam hardening compensation. We used 100%.

6. Adjust ring artifact reduction. We used 20.

7. Adjust histogram to get good contrast. This varies on a case-by-case basis. Use the same values for all images in series of scans.

8. Set format to TIFF. (Our DVC code does not support BMP.)

9. Run the reconstruction.

10. Convert TIFF images from 16-bit to 8-bit grayscale, e.g., using Photoshop or ImageMagick. Save as PNG or TIFF images, which are both lossless formats.
References


