View-Dependent Representation of Shape and Appearance from Multiple View Video

Marco Volino

Faculty of Engineering and Physical Sciences
University of Surrey

This dissertation is submitted for the degree of

Doctor of Philosophy

January 2016

© Marco Volino 2016
Summary

Over the past decade, markerless performance capture, through multiple synchronised cameras, has emerged as an alternative to traditional motion capture techniques, allowing the simultaneous acquisition of shape, motion and appearance. This technology is capable of capturing the subtle details of human motion, e.g. clothing, skin and hair dynamics, which cannot be achieved through current marker based capture techniques. Markerless performance capture has the potential to revolutionise digital content creation in many creative industries, but must overcome several hurdles before it can be seen as a practical mainstream technology. One limitation of the technology is the enormous size of the generated data. This thesis addresses issues surrounding compact appearance representation of virtual characters generated through markerless performance capture, optimisation of the underlying 3D geometry and delivery of interactive content over the internet.

Current approaches to multiple camera texture representation effectively reduce the storage requirements by discarding huge amounts of view dependent and dynamic appearance information. This information is important for reproducing the realism of the captured multiple view video. The first contribution of this thesis introduces a novel multiple layer texture representation (MLTR) for multiple view video. The MLTR preserves dynamic, view dependent appearance information by resampling the captured frames into a hierarchical set of texture maps ordered by surface visibility. The MLTR also enables computationally efficient view-dependent rendering by pre-computing visibility testing and reduces projective texturing to a simple texture lookup. The representation is quantitatively evaluated and shown to reduce the storage cost by > 90% without a significant effect on visual quality.

The second contribution outlines the ideal properties for the optimal representation of 4D video and takes steps in achieving this goal. Using the MLTR, spatial and temporal consistency is enforced using a Markov random field framework, allowing video compression algorithms to make further storage reductions through increased spatial and temporal redundancies. An optical flow-based multiple camera alignment method is also introduced to reduce visual artefacts, such as blurring and ghosting, that are caused by approximate geometry and camera calibration errors. This results in clearer and sharper textures with a lower storage footprint.

In order to facilitate high quality free-viewpoint rendering, two shape optimisation methods are proposed. The first combines the strengths of the visual hull, multiple view stereo and temporally consistent geometry to match visually important features using a non-rigid
iterative closest point method. The second is based on a bundle adjustment formulation which jointly refines shape and calibration. While, these methods achieve the objective of enhancing the geometry and/or camera calibration parameters, further research is required to improve the resulting shape.

Finally, it is shown how the methods developed in this thesis could be used to deliver interactive 4D video to consumers via a WebGL enabled internet browser, e.g. Firefox or Chrome. Existing methods for parametric motion graphs are adapted and combined with an efficient WebGL renderer to allow interactive 4D character delivery over the Internet. This demonstrates for the first time that 4D video has the potential to provide interactive content via the internet which opens this technology up to the widest possible audience.
Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Adrian Hilton, for his advice, support and patience over the last four years. This work would not have been possible without his input. I would also like to thank the examiners, Prof. Peter Eisert and Dr. Jean-Yves Guillemaut, who made the viva process enjoyable and their suggestions lead to an improvement in the content of this document.

I also want to express gratitude to past and present members of CVSSP who have offered advice, encouragement and friendship over the years, including Dan Casas, John Collomosse, Peng Huang, Liz James, Martin Klaudiny, Charles Malleson, Armin Mustafa, Alexandros Neophytou and Margara Tejera. Finally, a special thanks goes to my family and partner Elaina for their continued love and support.

This research was supported by the Engineering and Physical Sciences Research Council (EPSRC) and British Broadcasting Corporation (BBC) through an iCASE studentship award 3D video editing.
Table of contents

List of figures xi

List of tables xv

Notation and Abbreviations xvii

1 Introduction 1
   1.1 Context and Motivation ........................................ 1
   1.2 Problem Statement and Objectives .............................. 2
   1.3 Methodology .................................................. 4
   1.4 RE@CT Project ............................................... 5
   1.5 Contributions ............................................... 6
   1.6 Thesis Outline ............................................. 7
   1.7 Publications ............................................... 8

2 Human Performance Capture 9
   2.1 Introduction ............................................... 9
   2.2 Data Acquisition ........................................... 11
      2.2.1 Cameras and Camera Calibration ......................... 11
      2.2.2 Scene Background .................................. 12
      2.2.3 Illumination ........................................ 14
   2.3 Shape Reconstruction and Representation .................... 14
      2.3.1 Shape Reconstruction Techniques ...................... 15
      2.3.2 Model based Approaches ............................... 17
      2.3.3 Model free Approaches ................................ 19
      2.3.4 Temporally Consistent Geometry ...................... 21
   2.4 Visualisation: Rendering from captured images ............. 22
# Table of Contents

2.4.1 The Free Viewpoint Video Renderer .................................................. 23
2.5 Multiple camera studio systems ............................................................... 25
2.6 4D Video Datasets .................................................................................... 27
  2.6.1 4D Video Textures Dataset ................................................................. 28
  2.6.2 The SurfCap Dataset ........................................................................... 29
  2.6.3 MIT T-Samba ....................................................................................... 30
  2.6.4 Cloth and Face Datasets ..................................................................... 31
2.7 Conclusion ................................................................................................. 32

3 Multiple Layer Texture Representation .......................................................... 35
  3.1 Introduction ............................................................................................. 35
  3.2 Background .............................................................................................. 36
    3.2.1 Texture Mapping ............................................................................... 36
    3.2.2 Surface Parametrisation .................................................................... 38
    3.2.3 Multiple Camera Texture Representation .......................................... 40
    3.2.4 Efficient View Dependent Rendering ................................................ 41
    3.2.5 Summary and Motivation ................................................................... 42
  3.3 Problem Formulation .................................................................................. 42
    3.3.1 Extraction .......................................................................................... 45
    3.3.2 Storage .............................................................................................. 47
    3.3.3 Rendering .......................................................................................... 48
  3.4 Results and Evaluation ............................................................................... 50
    3.4.1 Error Metrics .................................................................................... 50
    3.4.2 Rendering Quality .............................................................................. 52
    3.4.3 Storage ............................................................................................. 54
    3.4.4 Memory ............................................................................................. 55
    3.4.5 Dan Dataset ....................................................................................... 57
    3.4.6 Roxanne Dataset ............................................................................... 58
    3.4.7 Cloth Dataset .................................................................................... 59
    3.4.8 Face Dataset ..................................................................................... 60
  3.5 Conclusions .............................................................................................. 61

4 Optimal Representation of 4D Video ................................................................. 63
  4.1 Introduction ............................................................................................. 63
  4.2 Background .............................................................................................. 64
    4.2.1 Graph-Cuts in Computer Vision ........................................................ 64
Table of contents

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2 Markov Random Fields</td>
</tr>
<tr>
<td>4.2.3 Optical Flow</td>
</tr>
<tr>
<td>4.2.4 Multiple Camera Texture Alignment</td>
</tr>
<tr>
<td>4.3 Optimal Representation of 4D Video</td>
</tr>
<tr>
<td>4.4 Optimisation of Camera Assignment</td>
</tr>
<tr>
<td>4.4.1 Proof of Regularity</td>
</tr>
<tr>
<td>4.5 Multiple Camera Texture Alignment</td>
</tr>
<tr>
<td>4.6 Results and Evaluation</td>
</tr>
<tr>
<td>4.6.1 Error Metric</td>
</tr>
<tr>
<td>4.6.2 Optimisation Results</td>
</tr>
<tr>
<td>4.6.3 Rendering Quality versus Optimisation</td>
</tr>
<tr>
<td>4.6.4 Storage</td>
</tr>
<tr>
<td>4.6.5 Multiple Camera Optical Flow-based Texture Alignment</td>
</tr>
<tr>
<td>4.7 Conclusions</td>
</tr>
</tbody>
</table>

5 Spatio-Temporal Shape Optimisation

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
</tr>
<tr>
<td>5.2 Background</td>
</tr>
<tr>
<td>5.3 Approach 1: Non-Rigid ICP Shape Optimisation</td>
</tr>
<tr>
<td>5.3.1 Overview</td>
</tr>
<tr>
<td>5.3.2 Problem Formulation</td>
</tr>
<tr>
<td>5.3.3 Rim Computation</td>
</tr>
<tr>
<td>5.3.4 Visual Hull Confidence</td>
</tr>
<tr>
<td>5.3.5 Correspondence Search</td>
</tr>
<tr>
<td>5.3.6 Non-rigid ICP</td>
</tr>
<tr>
<td>5.3.7 Summary</td>
</tr>
<tr>
<td>5.4 Approach 2: Bundle Adjustment Shape Optimisation</td>
</tr>
<tr>
<td>5.4.1 Overview</td>
</tr>
<tr>
<td>5.4.2 Vertex Reprojection</td>
</tr>
<tr>
<td>5.4.3 Bundle Adjustment</td>
</tr>
<tr>
<td>5.5 Results and Evaluation</td>
</tr>
<tr>
<td>5.5.1 Error Metrics</td>
</tr>
<tr>
<td>5.5.2 Silhouette Overlap Evaluation</td>
</tr>
<tr>
<td>5.5.3 Reprojection Error Evaluation</td>
</tr>
<tr>
<td>5.6 Limitations</td>
</tr>
</tbody>
</table>
5.7 Conclusion ................................................................. 118

6 Online Interactive 4D Character Animation .......................... 121
   6.1 Introduction .......................................................... 121
   6.2 Background .......................................................... 122
      6.2.1 Character Animation Techniques .......................... 122
      6.2.2 WebGL .......................................................... 123
   6.3 System Overview .................................................... 124
      6.3.1 Offline Data Processing ...................................... 125
      6.3.2 Client Application .............................................. 127
   6.4 Results ............................................................... 130
   6.5 Conclusions ......................................................... 134

7 Conclusions and Future Work ........................................... 135
   7.1 Introduction .......................................................... 135
   7.2 Multiple Layer Texture Representation .......................... 135
   7.3 Spatio-Temporal Shape Optimisation ............................. 138
   7.4 WebGL-based Delivery of 4D Characters ......................... 139
   7.5 Conclusion .......................................................... 140

References ........................................................................ 141
## List of figures

1.1 Overview of thesis contributions ........................................ 4  
1.2 Organisation of RE@CT work packages ................................. 5  
2.1 History of motion capture ............................................... 10  
2.2 Tools for capture and camera calibration .............................. 13  
2.3 Shape reconstruction methods ....................................... 20  
2.4 Free viewpoint video rendering pipeline by Starck *et al.* [95]. 23  
2.5 View dependent rendering ........................................... 24  
2.6 Processing pipeline .................................................. 26  
2.7 Dan dataset overview ................................................ 28  
2.8 Roxanne dataset examples ........................................... 29  
2.9 MIT 4D capture dataset example ....................................... 30  
2.10 Face 4D dataset examples .......................................... 31  
2.11 Cloth 4D dataset examples ........................................ 32  
3.1 Cube texture mapping example ..................................... 36  
3.2 Photo-realistic render of Earth combing multiple appearance maps 37  
3.3 Examples of surface parametrisation methods ..................... 38  
3.4 UV parameterisation of a 3D surface ................................ 39  
3.5 Multiple layer texture representation: Approach overview .......... 43  
3.6 Multiple layer texture representation processing and rendering pipeline 44  
3.7 Multiple layer texture representation: Rendering pipeline .......... 49  
3.8 Multiple layer texture representation: Rendering quality comparison 52  
3.9 Rendering comparison of MLTR with FVVR .......................... 53  
3.10 Multiple layer texture representation: Storage reduction comparison 55  
3.11 MLTR GPU memory reduction v.s. FVVR, expressed as a percentage. 56  
3.12 Multiple layer texture representation: Dan dataset example .......... 57
3.13 Multiple layer texture representation: Rendering examples Dan dataset . . . 57
3.14 Multiple layer texture representation: Roxanne dataset example . . . . . . . 58
3.15 Rendering examples from Roxanne dataset walk sequence using the MLTR. 58
3.16 Multiple layer texture representation: Cloth dataset example . . . . . . . . 59
3.17 Rendering examples from Cloth dataset using the MLTR. . . . . . . . . . . 59
3.18 Multiple layer texture representation: Face dataset example . . . . . . . . . 60
3.19 Rendering examples from the Face dataset using the MLTR. . . . . . . . . . 60

4.1 Visualisation of optical flow . . . . . . . . . . . . . . . . . . . . . . . . . . 67
4.2 Ideal texture representation properties . . . . . . . . . . . . . . . . . . . 69
4.3 Mesh structure represented as an undirected graph. . . . . . . . . . . . . . 71
4.4 Proof of regularity: Triangular inequality . . . . . . . . . . . . . . . . . . . 74
4.5 Multiple camera projective texture example . . . . . . . . . . . . . . . . . 77
4.6 Multiple camera optical flow . . . . . . . . . . . . . . . . . . . . . . . . . 78
4.7 Optimisation results for Dan dataset . . . . . . . . . . . . . . . . . . . . . 81
4.8 Optimisation results for Roxanne dataset . . . . . . . . . . . . . . . . . . . 82
4.9 Optimisation results Samba dataset . . . . . . . . . . . . . . . . . . . . . . 82
4.10 Optimisation results Cloth dataset . . . . . . . . . . . . . . . . . . . . . . 83
4.11 Optimisation results Face dataset . . . . . . . . . . . . . . . . . . . . . . . 83
4.12 The effect of optimisation on rendering quality . . . . . . . . . . . . . . . 84
4.13 The effect of optimisation on MLTR storage size . . . . . . . . . . . . . . 85
4.14 Multiple camera optical flow alignment result: Dan . . . . . . . . . . . . 87
4.15 Multiple camera optical flow alignment Face dataset . . . . . . . . . . . . 88
4.16 Multiple camera optical flow alignment Cloth dataset . . . . . . . . . . . . 88

5.1 Profile views of Dan character from several reconstruction techniques . . 94
5.2 Overview of non-rigid ICP shape refinement approach. . . . . . . . . . . . 95
5.3 Mid-point sub-division of temporally consistent geometry. . . . . . . . . . 95
5.4 Rim extraction using a min-cut/max-flow formulation . . . . . . . . . . . . 97
5.5 Visual hull rims example from frame in Dan dataset. . . . . . . . . . . . . 98
5.6 Hausdorff distance is used to measure the visual hull confidence. . . . . . . 99
5.7 Correspondence search constrained to displacement along vertex normal. 101
5.8 Overview of Shape refinement approach using bundle adjustment. . . . . 104
5.9 Vertex reprojection process. . . . . . . . . . . . . . . . . . . . . . . . . . . 105
5.10 Vertex reprojection error based on optical flow. . . . . . . . . . . . . . . 106
5.11 Profile view of Dan character after shape refinement . . . . . . . . . . . . 110
List of figures

5.12 Silhouette overlap evaluation of reconstruction methods for Dan dataset . . 112
5.13 Silhouette overlap evaluation of reconstruction methods for Dan dataset. . 113
5.14 Shape optimisation results for Dan dataset close up. . . . . . . . . . . . . 114
5.15 Shape optimisation results for Dan dataset. . . . . . . . . . . . . . . . . . . 115
5.16 Shape optimisation results for Roxanne dataset. . . . . . . . . . . . . . 116

6.1 WebGL character animation platform: system overview . . . . . . . . . . 124
6.2 Parametric motion graph example . . . . . . . . . . . . . . . . . . . . . . . . 126
6.3 Adapting the MLTR for WebGL based rendering . . . . . . . . . . . . . . 127
6.4 Flow chart of WebGL character animation engine . . . . . . . . . . . . . . 129
6.5 Memory arrangement of WebGL renderer. . . . . . . . . . . . . . . . . . . 129
6.6 WebGL character animation engine: Dan Example . . . . . . . . . . . . 130
6.7 WebGL character animation engine: Roxanne example . . . . . . . . . . 131
6.8 WebGL character animation engine: Ballet example . . . . . . . . . . . . 132
6.9 WebGL user interface examples . . . . . . . . . . . . . . . . . . . . . . . . 133

7.1 Texture map storage using principal component analysis . . . . . . . . . 137
List of tables

2.1 Comparison of reported human performance systems   27
2.2 Overview of datasets used for evaluation   27
3.1 Dataset summary   50
3.2 MLTR storage requirements v.s. captured video streams   54
4.1 Summary of processing times using MRF framework   80
4.2 Evaluation of temporal flicker.   81
4.3 Evaluation of optimisation on storage.   86
5.1 Silhouette overlap evaluation.   111
5.2 Reprojection error evaluation.   117
6.1 Overview of processed datasets.   132
Notation and Abbreviations

Typesetting

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar values</td>
<td>Lower-case letters in italic (e.g. (x, y))</td>
</tr>
<tr>
<td>Vectors</td>
<td>Lower-case letters with an arrow above (e.g. (\vec{x}, \vec{y})). elements of the vector are accessed using sub-script number or letter (e.g. (\vec{x}_1, \vec{p}_x))</td>
</tr>
<tr>
<td>Matrices</td>
<td>Upper-case letters in bold (e.g. (K, R))</td>
</tr>
<tr>
<td>Functions</td>
<td>Lower-case letter with brackets for variable inputs (e.g. (a(x), e_T(a, b)))</td>
</tr>
<tr>
<td>Sequences</td>
<td>Upper-case letter enclosed with curley brackets over a defined range (e.g. ({C(i)}_{i=1}^{N}))</td>
</tr>
<tr>
<td>Total number of elements</td>
<td>Upper-case (N) with subscript identifier (e.g. (N_T, N_C))</td>
</tr>
<tr>
<td>Elements from graph theory</td>
<td>Upper-case in calligraphic style (e.g. (\mathcal{G}, \mathcal{V}, \mathcal{E}))</td>
</tr>
</tbody>
</table>

Symbols

Scalars

\(i, j, k\)    Indexes used interchangeably. Each are defined in respective function.
\(t\)          Time instance or frame index
Vectors

⃗\(c^D_i\) Direction vector of \(i^{th}\) camera
⃗\(c^P_i\) Position vector of \(i^{th}\) camera
\(\vec{t}\) Camera translation
\(\vec{v}\) A vertex position
\(\vec{n}\) A vertex normal

Matrices

\(K(i)\) Intrinsic parameter matrix of the \(i^{th}\) camera
\(R(i, t)\) Rotation matrix of the \(i^{th}\) camera at frame \(t\)

Element Totals

\(N_T\) Number of frames
\(N_C\) Number of cameras
\(N_P\) Number of polygons
\(N_L\) Number of layers
\(N_L\) Number of labels
\(N_V\) Number of vertices
\(N_i\) Number of iterations

Sequences

\(\{\{C(i, t)\}\}_{i=1}^{N_C} \}_{t=1}^{N_T}\) Cameras
\(\{\{I(i, t)\}\}_{i=1}^{N_C} \}_{t=1}^{N_T}\) Camera Images
\(\{\{S(i, t)\}\}_{i=1}^{N_C} \}_{t=1}^{N_T}\) Camera Silhouette
\(\{\{D(i, t)\}\}_{i=1}^{N_C} \}_{t=1}^{N_T}\) Camera Depth map

Images

\(R(i, j, t)\) Rendered image from the view-point of the \(i^{th}\) camera using the texture of the \(j^{th}\) camera at time \(t\)
\(O(i, j, t)\) Optical flow between \(R(i, i)\) and \(R(i, j)\) at time \(t\)
Graph Cut Variables

\[ G \] Graph Structure \\
\[ V \] Graph Vertices \\
\[ E \] Graph Edges \\
\[ L \] Set of labels \\
\[ c_i \] Camera Labels \\
\[ p_i \] Polygon Nodes \\
\[ c_N \] Null Camera Label

Abbreviations

2D Two-Dimensional \\
3D Three-Dimensional \\
4D Four-Dimensional \\
BA Bundle Adjustment \\
CAE Character Animation Engine \\
FVVR Free Viewpoint Video Renderer \\
GB Gigabytes \\
MB Megabytes \\
MLTR Multi Layer Texture Representation \\
MRF Markov Random Fields \\
MVS Multiple View Stereo \\
NO No Optimisation \\
SO Spatial Optimisation \\
STO Spatial/Temporal Optimisation \\
TC Temporally Consistent \\
VH Visual Hull
Chapter 1

Introduction

1.1 Context and Motivation

Human motion capture is a fascinating field of study which has captivated scientists, artists and engineers for over a century [39]. Huge leaps in terms of accuracy, robustness and speed have been driven by technological advances in both hardware capabilities and algorithm design. Today, the holy grail of human motion capture is the creation of controllable digital doubles which look, move and behave in such a way that is indistinguishable from a real human. The ability to do so accurately, efficiently and cheaply would unify digital asset creation in the game, film and broadcast industries. It would also have far reaching applications outside of the entertainment industry, e.g. medical imaging, diagnosis and monitoring rehabilitation, as well as realistic training and simulation environments through virtual and augmented reality.

Traditional approaches to motion capture rely on optical markers or mechanical devices attached to the subject to detect joint positions or measure joint rotations. The pose of a subject is efficiently represented as a set of joint angles for an underlying skeletal structure. The captured joint angles are then retargeted to an artist-generated mesh which is deformed to match the skeletal pose using methods such as linear blend skinning (LBS). The advantage of this representation is that the pose of the human body can be captured, represented and intuitively controlled in a relatively low dimensional space, greatly reducing complexity. This is extremely useful for real-time applications. Over recent decades, production pipelines have been refined to handle skeleton based motion capture data and animation. However, there are several drawbacks to this approach. Firstly, it requires many hours for teams of skilled artists
Introduction

to create the base geometry, edit captured motions and design the appearance of the subject. Secondly, this approach only captures the changes in joint angles. Therefore, more subtle details required to convey realistic motion, such as non-rigid dynamics of clothing and facial expression, are lost.

To overcome these limitations, the computer vision and graphics research community moved from capturing a sparse set of joint positions to dense, dynamic, non-rigid surfaces using multiple synchronised cameras. Instead of markers, image based features are used to reconstruct the time varying geometry of the subject while simultaneously acquiring the appearance. This captures the subtle details of human motion which add an element of realism that cannot be achieved through traditional animation techniques. Advances in visual reconstruction of non-rigid shape from multiple view video enable video realistic rendering of dynamic scenes from novel views with the visual quality of the captured video. However, due to the independent reconstruction at each frame, resulting sequences are limited to replay of the captured performance. Animation based on concatenation of segments of the captured motion is limited to reordering and replay of the captured sequences [52].

Geometry processing algorithms allow non-rigid temporal alignment across multiple sequences and make it possible to create a consistent geometric representation across all captured frames. This is a compact and efficient representation of time varying geometry in terms of storage, transmission and utilisation. This is due to the topology of the mesh remaining fixed over all frames with only vertex positions changing over time. Recently, it has been demonstrated that four dimensional video (4D Video) can be used to create novel, real-time animations and is no longer limited to replay of the captured data [22]. 4D video is defined as a temporally consistent geometry representation plus the captured images. Combining 4D video data with view dependent rendering results in rendered models with a visual quality approaching that of the captured video. This is achieved by sampling the appearance directly from the captured frames. However, this visualisation process is inefficient as it requires access to the raw multiple view video at render time.

1.2 Problem Statement and Objectives

Multiple view capture inherently leads to large amounts of data, which is prohibitive to this technology being widely adopted. The capture process can result in hundreds of gigabytes (GB) per minute of capture. This large data size not only impacts storage, but requires a
huge amount of bandwidth to transfer and memory to use. Existing work has dealt with this problem by merging multiple images into a single texture map, either by selecting the best camera view or blending overlapping camera views together for each surface point. This motivates the starting point for the research presented in this thesis to address the open problem of how to efficiently represent the multiple view appearance. This should be done in such a way that preserves both view dependent and dynamic appearance of the captured video. These are important properties to reproduce the realism of the captured video as: (1) the capture video represents detailed dynamic appearance; (2) reconstructed shape is often not sufficiently accurate for sub-pixel accurate alignment of the surface appearance; (3) natural material such as skin and clothing change appearance with viewpoint. If the dynamic appearance observed in the multiple view video is not accurately represented the rendering of novel viewpoints will result in a loss of visual quality. Consequently, most approaches to date have directly represented the appearance from the captured multiple view video to preserve visual quality. However, this limits the utilisation of video based performance capture to local rendering on high performance platforms due to the associated data size.

The primary objective of the research presented in this thesis is to address the open problem of appearance representation from multiple camera capture. An emphasis is placed on developing a compact, efficient representation of the subject’s appearance. Any multiple camera texture representation should maintain the view dependent and dynamic appearance, e.g., creases in clothing, wrinkles and facial expression, without resulting in loss of visual quality.

A second objective is to improve shape reconstruction to facilitate high quality view dependent rendering. Existing model free reconstruction pipelines utilise several reconstruction techniques which allow errors to propagate through to the final rendered model as texture artefacts. Rather than detect and correct these texture artefacts at render time, as in other approaches [23, 35], the objective here is to correct the underlying calibration and geometry.

The final objective is to demonstrate that 4D video can be used to create interactive content for the widest possible audience. This is possible using WebGL, a cross platform graphics API based on OpenGL, allowing 3D graphics to be rendered natively in a web browser.
1.3 Methodology

The work presented in this thesis builds on the existing reconstruction and animation pipeline developed at the University of Surrey. An overview of the existing pipeline along with the contributions of this thesis are shown in Figure 1.1.

Fig. 1.1 Overview of thesis contributions and where they fit within a 4D video processing pipeline
1.4 RE@CT Project

The RE@CT project was a three year (November 2011-October 2014) seventh framework programme (FP7) project funded by the European Union. The project spanned the entire 4D performance capture pipeline from data acquisition, dynamic shape reconstruction, content analysis and editing, real-time interactive animation and content delivery. RE@CT aimed to revolutionise the production of realistic virtual characters for creating digital content for TV, the web and other interactive platforms. This was done with an emphasis on reducing the effort of skilled artists in the pipeline in order to reduce production time and cost compared to the conventional animation methods. The consortium included research institutions and SME’s from across Europe including: Artefacto, British Broadcast Corporation (BBC), Fraunhofer HHI, Inria, Oxford Metrics Group Vicon and the University of Surrey.

Within the scope of the project, the University of Surrey developed techniques for performing real-time character animation using parametric surface motion graphs, addressing multiple camera texture representation and online delivery of interactive 4D characters. The work presented in this thesis was conducted as part of the RE@CT project. An overview of the work packages of the RE@CT project are shown in Figure 1.2 along with the input/output data of each work package.

![Fig. 1.2 Organisation of RE@CT work packages](image-url)
1.5 Contributions

The contributions presented in this thesis are listed below. Figure 1.1 shows how these contributions fit within a 4D video processing pipeline.

• A novel multiple layer texture representation (MLTR) that significantly reduces the storage requirements of 4D video. This is achieved by resampling the multiple view video frames into a hierarchy of texture maps ordered by surface visibility based on the temporally consistent geometric representation. This removes redundant information from video frames and allows storage reductions through spatial and temporal redundancies.

• A method to optimise the MLTR assignment to further reduce the storage requirements. This is posed in a Markov random field framework that computes an optimal assignment that allows video compression algorithms to exploit increased spatial and temporal redundancies.

• An optical flow based multiple camera alignment method to reduce texture artefacts caused by approximate geometric models and erroneous camera calibration. Correspondence between wide baseline cameras is established by projectively texturing a model from the viewpoint of each camera using the texture of every other camera. Optical flow correspondence is computed between the projectively textured images which is used to correct errors when sampling into the UV domain.

• A novel rendering pipeline based on the MLTR that enables computationally efficient view dependent rendering to be performed. This is achieved given that the MLTR inherently pre-computes and stores results of computationally expensive operations directly into the representation. This includes depth testing and projective texturing which leaves only texture blending based on the virtual camera viewpoint.

• A shape refinement technique that frames the task as a registration problem and uses the strengths of two reconstruction techniques to create temporally consistent mesh sequences with increased geometric detail.

• A shape refinement technique that jointly refines both geometry and calibration to minimise the multiple camera reprojection error.

• A WebGL based character animation framework for delivering interactive characters via the internet. The client application and required data is hosted on a web server and is transferred via HTTP to a client device to be rendered locally.
1.6 Thesis Outline

Chapter 1: Introduction
This chapter presents the context of the research problem, motivation, objectives and contributions of the research. A list of publications resulting from this research is also given.

Chapter 2: Human Performance Capture
Chapter 2 presents a history of performance capture and an overview of state-of-the-art techniques in markerless 4D performance capture. The chapter is broken down into the different stages of the pipeline including capture, shape reconstruction and alignment, and free viewpoint rendering. A description of some example systems are given as well as a summary of datasets used in this thesis for evaluation purposes.

Chapter 3: Multiple Layer Texture Representation
This chapter introduces the novel multiple layer texture representation which was developed to reduce the storage requirements and enable efficient view dependent rendering. The method is quantitatively evaluated against an existing state-of-the-art free viewpoint video renderer.

Chapter 4: Optimal Representation of 4D Video
Chapter 4 builds on the multiple layer texture representation presented in the previous chapter by introducing methods to improve spatio-temporal alignment. This results in further storage reductions by exploiting increased spatial and temporal redundancies and reduced visual artefacts caused by inaccuracies in camera calibration and geometry.

Chapter 5: Spatio-Temporal Geometry Refinement
This chapter presents two methods for shape refinement. The first method approaches shape optimisation as a registration problem and uses a non-rigid Iterative closest point algorithm. Surface details are reintroduced by identifying visually important features from two other reconstruction methods. The second method approaches shape optimisation as a bundle adjustment problem and jointly refines geometry and calibration to minimise errors in reprojection error.

Chapter 6: Online Interactive Character Animation
Chapter 6 presents the first WebGL based character animation framework to use 4D video data. Geometry and texture data are processed into compact representations and sequences are analysed against a user defined motion graph. This data is hosted on a server, transferred to the client via HTTP, and then rendered locally.
Chapter 7: Conclusions and Future Work

This chapter draws conclusions from the work presented throughout this thesis and makes suggestions for future research avenues.

1.7 Publications

The following publications are a result of the work described in this thesis:

- **Layered View-Dependent Texture Maps**: M. Volino, J. Painsoit, O. Grau, and A. Hilton; European Conference on Visual Media Production (CVMP) 2012; Short Paper


- **Layered View-Dependent Texture Maps**: M. Volino and A. Hilton; 10th European Conference on Visual Media Production (CVMP) 2013; Poster Presentation.


- **Optimal Representation of Multiple View Video**: M. Volino, D. Casas, J. Collomosse and A. Hilton; British Machine Vision Conference (BMVC) 2014; Oral Presentation.

- **Online Interactive 4D Character Animation**: M. Volino, P. Huang and A. Hilton; International Conference on 3D Web Technology (Web3D) 2015; Oral Presentation.

- **RE@CT: A new production pipeline for interactive 3D content**: F. Schweiger, G. Thomas, W. Paier, M. Kettern, P. Eisert, J.S. Franco, M. Volino, P. Haung, J. Collomosse, A. Hilton, V. Jantet, P. Smyth; IEEE International Conference on Multimedia and Expo Workshops (ICMEW) 2015;
Chapter 2

Human Performance Capture

2.1 Introduction

In 1917 Max Fleischer patented the rotoscope [39], one of the first known devices used for motion capture. The rotoscope worked by projecting video onto a frosted glass plate allowing an artist to trace the desired subject frame-by-frame as shown in Figure 2.1a. This simple device allowed an artist to create two dimensional (2D) animated characters that exhibited natural, life-like motion. The process was extremely time consuming, but was used extensively during the first half of the 1900s in many animated productions such as Koko the Clown (1920s), Walt Disney’s Snow White and the Seven Dwarfs (1937) and Superman (1940s). Rotoscoping was later used for the scientific study of human and animal motion, e.g. human locomotion, the gallop of a horse and birds in flight.

In the 1970’s, the development of marker based systems allowed human performance capture to move from the 2D domain into the 3D domain. In these systems, a sparse set of optical markers were placed on the capture subject typically located at the joints, Figure 2.1b. These markers were detected by multiple sensors that could compute the 3D positions by triangulation. It was not until the late 1980’s that these systems could automatically and robustly detect markers in real-time. The position of the 3D points was used to estimate a set of joint angles of an underlying skeletal structure. This representation allowed high level and intuitive full body control of the virtual skeleton with relatively few degrees of freedom. In a process called rigging or skinning, the skeleton is embedded into an artist generated mesh calculating the influence each bone has on the surrounding mesh vertices that are described using a set of weights. A captured performance can then be replayed, edited
or novel animation created using the skeletal structure, artist mesh, skinning weights and time varying joint angles. Today, this pipeline is seen as the gold standard for performance capture and commonly used throughout the film, broadcast and gaming industries. However, many of the problems within this pipeline are solved through human intervention requiring many hours of manual work to create realistic animations.

Fig. 2.1 A history of motion capture - past, present and future. (a) Rotoscope device, from 1917 patent by Fleischer [39], allowed an artist to create 2D character with natural life-like motion. (b) Marker based capture uses optical markers attached to the actor (left) to track the position of joints that are mapped to an underlying skeleton structure (right) [60]. (c) Markerless capture requires the actor to be captured by multiple overlapping camera views. This acquires both motion and appearance simultaneous. Image taken as part of the RE@CT project.
Over the past decade, the computer vision and graphics community have developed markerless motion capture systems that reconstruct dense, dynamic surface geometry and appearance recorded by multiple cameras, Figure 2.1c. The major advantage of markerless motion capture systems is that they allow the simultaneous acquisition of both motion and appearance as well as the fine details of human motion, something that cannot be achieved through traditional animation techniques or marker based motion capture. Markerless motion capture, reconstruction and utilisation of the data can be broken down into five key areas: accurate data acquisition, robust reconstruction and compact representation of data, analysis of data, transfer of data, and efficient visualisation.

In this chapter, an overview of fundamental techniques and state-of-the-art approaches to markerless 4D performance capture is given. The main focus of this thesis is full body capture, however the same methods can also be applied to multiple camera capture of other subjects, e.g. face and cloth performance, which are used in this thesis for evaluation wherever possible.

## 2.2 Data Acquisition

In general, 4D performance capture takes place in the controlled environment of a multiple camera studio. This allows complete control over the capture environment including factors such as camera configuration, scene content and illumination. Starck et al. [96] presented a comprehensive report on the design, setup and operation of a multiple camera studio for human performance capture. The significance of cameras, scene background and lighting are now discussed.

### 2.2.1 Cameras and Camera Calibration

Almost every multiple camera system for 3D or 4D performance capture described in the literature use a set of static synchronised cameras featuring a global shutter [20, 26, 27, 91, 108]. Cameras are generally in a fixed position which simplifies the task of camera calibration but places restrictions on the size of the capture volume. Moving cameras have been used to improve resolution for larger capture environments, e.g. Guillemaut and Hilton [49] used moving cameras to enable coverage of an entire sports field. However, moving cameras
are not a common feature of multiple camera studios dedicated for human performance capture.

Cameras are generally synchronised meaning all images are captured at exactly the same time instance. This is achieved by triggering the cameras using an external signal. A global shutter ensures that all pixels in the camera sensor are exposed to light at and for the exact same time period. In contrast, a rolling shutter reads lines of pixels at different times which produces artefacts in the presence of fast motion.

The first step in any multiple camera capture session is to undertake camera calibration. This creates a mathematical model of the capture cameras in a world coordinate system [105]. A calibration object of known geometry, commonly a checker board pattern (Figure 2.2a) or wand (Figure 2.2b), is shown to all cameras in their fixed positions. Wands are commonly used in 360 degree capture environments as they allow full visibility from all directions whereas a checker board has limited visibility. 2D image features, e.g. corners of the checker board or points on the wand, are detected and matched across multiple camera views at each time instance. Bundle adjustment, e.g. Ceres-Solver [2], is then used to optimise the camera models by minimising the reprojection error between the predicted and observed 2D features across all cameras and all frames. This method has been successfully used to calibrate multiple camera systems ranging from $<10$ to $>100$ [26] with sub-pixel reprojection error.

### 2.2.2 Scene Background

A typical feature of a multiple camera studio used for 4D capture is the controlled scene background, visible in Figure 2.1c and 2.2b. This is generally a uniformly coloured chroma cloth [94], matte painted walls and floor [108], or retro reflective cloth [48]. Retro reflective cloth reflects light back to the light source in a similar fashion to a road sign. In order to use the material, an LED ring is placed around the camera lens as shown in Figure 2.2c. This material provides a uniformly coloured background from the point of view of the camera and reduces the appearance of ground shadows. However, the material is expensive compared to chroma cloth and the LED lights cause colour spill and lens flares if visible from other cameras.

The purpose of controlling the background is to facilitate the segmentation of the subject, or foreground objects, from the background. This is known as segmentation, matting or
keying depending on the industry. This allows silhouettes of the subject to be extracted via a variety of methods, one of them being chroma keying. Chroma keying is a long established technique used throughout film and broadcast industries which assumes the background is a known colour, typically blue or green. A binary threshold operation is applied to the difference of each pixel against the known colour which produces the foreground matting. Difference keying is another segmentation method in which the difference between a camera image and a background plate from each camera, *i.e.* an empty studio, is compared and a binary threshold operation applied to the difference.

Both of these methods have been shown to be successful in producing clean silhouettes but both have drawbacks. Chroma keying systems do not allow the subject to contain any colours similar to the known background colour as this will be considered background while difference keying systems are sensitive to shadows and changes in illumination. Examples of typical silhouettes from a multiple camera studio using chroma cloth can be seen in the Section 2.6.

Fig. 2.2 Tools for camera calibration: (a) Calibration wand used to capture extrinsic parameters. (b) Checker board chart used for intrinsic parameters. (c) LED ring used to illuminate retro reflective cloth.
2.2.3 Illumination

Lighting conditions are an important consideration and dramatically affect many stages of the processing pipeline. Uniform ambient lighting conditions are desirable as this reduces shadows cast by the subject onto the background and also self shadowing. In contrast, directional lighting, such as spotlights, create strong shadows on the background as well as self shadowing on the subject. The amount of light also has a direct effect on the camera settings required to capture high quality, noise-free images. If the ambient light level is low, a wider camera aperture is required which in turn reduces the depth-of-field and therefore restricts the usable capture volume. Lighting also puts limitations on the camera shutter speed which needs to be fast enough to capture the desired motion without motion blur artefacts in the images, but slow enough to ensure sufficient light is captured by the camera sensor to avoid a low signal-to-noise level. To counter this, the gain could be increased but this may result in noisy images. The trade off of aperture, shutter speed and gain are all directly affected by the scene illumination.

Illumination can also be used to aid 3D reconstruction. This can either be in the form of visible light, as with light stage systems [29], or infra-red (IR) light [26, 80]. Light stage capture systems, described in Section 2.5, allow full control of lighting conditions, e.g. source position and colour. These have been shown to produce highly detailed estimates of geometry, surface normals and material properties [29]. Structured light sources are used in a variety of RGB+D sensors, such as Microsoft Kinect, to aid real-time depth map computation.

2.3 Shape Reconstruction and Representation

3D reconstruction algorithms used for human performance capture can generally be placed into two categories: model based and model free. In this section, both approaches are described with advantages and limitations of each highlighted. First, two fundamental techniques for shape reconstruction are presented: the visual hull and multiple view stereo. These 3D reconstruction methods are commonly used throughout the literature in both model based and model free pipelines, after which state-of-the-art model based and model free reconstruction techniques are reviewed.

There are several representations of 3D shape that can be used to store and manipulate 3D objects, e.g. point based, surface based or volume based. Point based representations require
dense point clouds to ensure sufficient detail is preserved and surfaces appear complete. Surface based representations, e.g. triangular mesh surfaces, enable a complete continuous surface to be created using relatively few points, compared to point based representations. This is achieved by forming a set of connected triangles from related points. Triangular mesh surfaces are efficiently handled by graphics hardware and are commonly used throughout the creative industries. There are many standard file formats used to store point and surface based shapes, e.g. OBJ, OFF and PLY formats, that are handled by 3D manipulation software packages, e.g. Maya, Blender, MeshLab.

Volume based representations separate a 3D space into discrete volume elements, or voxels, and can be efficiently stored in computer memory. However, the scale of detail is limited to the resolution of the voxels and volume based representations have a poor precision to complexity trade off. Volume based representations are not commonly used for character animation purposes in industry applications and few standard file formats exist. However, they are used in the visual effects industry for simulation of fluid, smoke and fire effects. The chosen output representation for the work presented in this thesis is triangular mesh surfaces, although volumetric methods are leveraged within the processing pipeline.

2.3.1 Shape Reconstruction Techniques

Visual Hull

Laurentini [65] introduced the concept of the visual hull which is one of the simplest methods for shape reconstruction. The visual hull is a shape-from-silhouette method that computes the maximum volume of an object consistent with a set of silhouettes. This is often used as an initial shape estimate as it is conceptually simple. Algorithms to compute the visual hull are fast, efficient and easy to parallelise, making them suitable for implementation on modern GPUs. A significant drawback of this method is the inability to reconstruct concave regions of an object as it is not possible to capture this through silhouette information. Visual hull reconstruction also suffers from phantom volume artefacts, volumes that have been falsely classified to lie within the object. However, these artefacts become less likely with more camera views.

Two commonly used representations of the visual hull are volumetric [65] and polyhedral [40, 74]. Volumetric methods use a 3D grid of voxels. A binary classification is applied to each voxel based on whether it is projected within the silhouette foreground or background.
This carves away the 3D voxel space leaving only the voxels that fall inside the foreground forming the visual hull volume. However, volumetric methods have a poor precision to complexity trade off. For instance, halving the voxel size of a regular voxel grid gives four times the number of voxels for the same volume. Franco and Boyer [40] proposed the exact polyhedral visual hull. The authors computed a polyhedral representation of the visual hull based on the exact intersection of silhouette contours followed by a Delaunay triangulation of the view edges. Volumetric shapes can be converted into a triangular mesh surface using methods such as Marching Cubes [72].

Visual hull reconstruction is reliant on the assumption that camera calibration and foreground/background segmentation is available and both are accurate. Probabilistic visual hull methods [100] have also been developed to handle errors in segmentation and calibration.

**Multiple View Stereo**

Two view stereo is the problem of extracting depth, or disparity, information for each pixel in a pair of calibrated images. Multiple view stereo, a natural extension to two view stereo, computes the 3D structure of a subject captured from multiple calibrated camera views. Both are fundamental problems in computer vision and have a long research history with a rich variety of algorithms and approaches. The introduction of benchmark datasets, ground truth data and a common evaluation framework [85, 86] have been of huge benefit to the advancement of both of these problems.

Seitz et al. [86] proposed six key properties that differentiate multiple view stereo algorithms: scene representation, photo consistency measure, visibility model, shape prior, reconstruction algorithm and initialisation requirements. As previously discussed, several scene representations exist, e.g. volumetric, polygon mesh or depth maps. Some multiple view stereo algorithms exclusively use a single representation [36] whereas others combine multiple representations in a reconstruction pipeline [97]. Photo consistency can be measured via several methods, e.g. sum of squared differences (SSD), sum of absolute differences (SAD) or normalised cross correlation (NCC). These measures are applied to local windows of pixels between image pairs. SSD and SAD give the absolute difference between the two sets of pixels. NCC gives a measure relative to the average pixel intensity of the windows which makes it invariant to changes in gain between images.
Some multiple view stereo methods also employ a shape prior that is refined based on photo consistency. A shape prior can also be used to assist visibility computation [109]. A common shape prior that can be used if accurate segmentation is available is a visual hull reconstruction [41, 88, 97, 109]. Volumetric approaches based on graph-cuts have been shown to be successful methods for extracting an optimal surface [88, 97, 109]. However, these methods tend to over smooth the final surface.

All multiple view stereo algorithms require calibrated images as input, however additional initialisation parameters are also required depending upon the technique. Volumetric methods require an approximate bounding box in which to initialise a voxel grid [88, 97, 109]. Methods that utilise a visual hull reconstruction require accurate silhouettes of the subject, whereas depth map based approaches constrain the range of depth values which in turn constrains the final scene geometry [62]. Most multiple view stereo algorithms work on the assumption that the observed object is Lambertian. This means that the surface appearance is view independent and pixel intensities do not change with viewpoint. In practice, most scenes do not adhere to this assumption due to lighting conditions and material properties causing specular highlights.

### 2.3.2 Model based Approaches

Model based approaches use assumptions about the capture subject in order to simplify the reconstruction process. Assumptions of the subject are made in the form of a template model. This can either be a generic human shape model with an underlying skeletal structure [20, 91], a subject specific model [28, 108] or a statistical body shape model [11, 80, 98]. This provides a starting point for shape reconstruction, but can also capture variations in body shape and/or pose, as well as provide limits to the subject pose based on human physiology.

Carazza et al. [20] developed a model based reconstruction pipeline that utilised a generic human shaped template model. The template comprised of a triangular surface mesh rigged with a skeletal structure that consisted of 17 joints. Each joint used a fixed number of degrees of freedom (DOF) to match its anatomical function, e.g. 1 DOF for hinge joints and 3 DOF for ball joints. In this framework, model pose parameters were optimised to minimise the overlap between silhouettes generated by the model and real captured silhouettes. In parallel to this, Starck and Hilton [91] also developed a model based reconstruction framework that
used a similar generic body shape model. Their formulation not only took into account shape-from-silhouette constraints, but also made use of multiple view stereo and manually defined feature cues to drive the template model towards the shape and pose of the captured subject. Both of these methods adapt the shape of the template model to match the observed shape of the performer. A problem faced with using a generic template model is that the methods have to estimate both shape and pose of the subject which can be ambiguous. This was addressed by using a subject specific template model that could be obtained in a number of ways, e.g. laser scanning [28, 98, 108], multiple view stereo [94, 108] or manually created by an artist [108].

Vlasic et al. [108] used template meshes obtained via all of the previously mentioned sources, e.g. laser scanning, multiple view stereo or artist generated. The template was manually rigged with a skeleton. A visual hull reconstruction was performed and a skeletal pose was estimated for all frames. Manual intervention was required to correct frames that failed to correctly identify the skeletal pose, e.g. fast motion or self occlusion. The template model was deformed to match the estimated pose using linear blend skinning (LBS). This resulted in a model which was approximately aligned, but did not respect silhouette constraints and suffered from typical artefacts introduced by LBS, e.g. mesh collapse. The final step in the approach was to correct these errors by ensuring that the deformed model adheres to silhouette contours. De Aguiar et al. [27] also acquired a high resolution subject specific template mesh using a laser scanner. The template mesh was deformed to match the pose of the subject in each frame using a Laplican surface deformation. Photo-consistency information was used to recover surface detail in areas of the deformed template that project outside the captured silhouette regions. A limitation of a single subject specific template model is that smaller deformations, such as the wrinkles that occur in clothing, are typically limited to those which exist in the template model and often results in a rigid surface appearance in non-rigid areas of the surface often referred to as baked-in details.

To overcome the limitation of baked-in details, Stoll et al. [98] initially fit the template model to a reference sequence which is decomposed into rigid body parts and non-rigid clothing, e.g. a dress. A physically based cloth model was used to simulate the motion of the non-rigid components taking into account potential collisions between the non-rigid dress and the arms and legs. This approach was able to reconstruct characters in highly non-rigid garments, e.g. loose fitting dress, as well as produce novel animations using skeletal motion capture data. However, the reconstructed non-rigid garment motion did not reflect the true motion of the garment in the captured video.
Artefacts that appear in these pipelines occur as they make use of a single template mesh which captures subject shape in a single pose. To overcome this, statistical body models have become increasingly popular [8, 11, 80]. These statistical body models have captured shape and pose variations of single or multiple subjects and recently have achieved impressive results. However, these methods have only been applied to subjects in skin tight clothing or underwear and would fail in the presence of loose fitting clothing. Moving outside of the statistical parameter space can also produce very unrealistic results.

Model based reconstruction techniques simplify the reconstruction process using prior knowledge about the nature of the subject. However, these assumptions come at the cost of flexibility. Model based reconstruction methods fail when applied to general scene reconstruction where the scene content is not known prior to reconstruction. In a human performance capture setting, these methods have been used to great effect but have only been reliable when the human subject is wearing tight fitting clothing. Non-rigid features, such as loose clothing or hair cannot be reliably reconstructed.

2.3.3 Model free Approaches

Model free reconstruction methods offer greater flexibility than model based reconstruction methods as they make no assumptions about the capture subject [26, 57, 94]. This is an advantage as it allows model free approaches to handle complex non-rigid surfaces and cope with general objects that may be unknown prior to reconstruction. However, model free approaches struggle to achieve the same level of perceived detail as model based approaches. Geometric details are limited to those captured in the template model, but these are often obtained using higher resolution methods, e.g. laser scanning. A challenge that arises from model free reconstruction methods is that each frame is reconstructed independently. This means the number of vertices and mesh topology change from frame-to-frame making it difficult to reuse the data for anything other than replay or concatenative animations [52]. These mesh sequences are known as temporally inconsistent. Methods to produce a temporally consist mesh representation given an unstructured collection of meshes are reviewed in Section 2.3.4.

Kanade et al. [57] developed one of the first model free systems for capturing human performance. The Virtualised Reality system used 51 cameras position around a geodesic dome to record an actor. Reconstruction was performed using multiple baseline stereo [78] which generated dense depth maps from the viewpoint of each capture camera. These
depth maps were then fused together and converted to a mesh based representation to allow rendering from novel viewpoints.

Starck and Hilton [94] proposed a model free reconstruction pipeline that consisted of an initial shape estimation given by a visual hull [65] that was iteratively refined using multiple view stereo in a volumetric graph cuts formulation [97]. This pipeline demonstrated that the fine details of the performance could be reconstructed with sub-pixel accuracy, e.g. wrinkles in clothing. Photo-realistic view dependent rendering was achieved using a free viewpoint video renderer [95].

More recently, Collet et al. [26] presented the first fully automatic pipeline to generate streamable free viewpoint video from multiple camera capture. Their system leveraged shape-from-silhouette and multiple view stereo from both RGB and IR sources to recover high quality shape. More details of this system are given in Section 2.5.

Fig. 2.3 Shape reconstruction. (a) High resolution laser scan taken prior to capture [108]. (b) Deformed template to match observations [108]. (c) Visual hull reconstruction representing the maximum volume of the shape [65]. (d) Multiple view stereo refinement of visual hull [97].
2.3 Shape Reconstruction and Representation

2.3.4 Temporally Consistent Geometry

Temporally consistent geometry is desirable as it is an efficient representation of dynamic geometry sequences as the mesh structure remains fixed over all frames and only the vertex positions change over time. This property enables the study of a wide range of problems including geometry compression [103, 107], editing of sequences [102], and efficient animation [22, 23]. While temporally consistent geometry is inherited in model-based reconstruction pipelines, it is a challenging problem that must be solved in model-free pipelines.

Cagniart et al. [18, 19] split a 3D surface into geodesic patches and employed an iterative closest point algorithm to match rigid patches in order to align one mesh with another. A Laplacian mesh deformation framework [89] is also used to enforce soft constraints between adjacent patches. This approach was applied sequentially to mesh sequences. Two issues that are encountered in sequential frameworks are drift, due to accumulation of errors, and gross errors in the presence of large differences between two meshes, e.g., caused by rapid motion.

To overcome these issues, Budd et al. [17] proposed a non-sequential, non-rigid global alignment framework based on Laplacian deformations. A collection of unstructured meshes were organised into a tree structure based on shape similarity. The shape similarity metric used a six-dimensional descriptor based on both geometry and appearance [53]. Arranging frames into a non-sequential tree structure ensures pairwise alignment is performed between meshes with the most similar shape. Accumulated error was reduced as the maximum number of sequential frame alignments is equal to the depth of the tree, as opposed to the total length of the sequence. The tree structure was computed using Prim’s minimum spanning tree [81]. This calculates the shortest path to visit every node for a given graph structure.

Allain et al. [5] also tackled the problem of temporal consistency of mesh sequences and proposed a volumetric framework that split the mesh up into small volumes based on a centroidal Voronoi tessellation. This gave a uniform and regular partition of the volume space compared to other methods, e.g., voxel grid or Voronoi tessellation. This work produced stable temporal alignment in subjects that are wearing tight clothing. It would likely struggle with features that change volume, e.g., loose hair or a dress, as it attempts to preserve the initial volume of the subject.
2.4 Visualisation: Rendering from captured images

The process of visualising data reconstructed from multiple camera images is commonly referred to as rendering. Methods for rendering from captured images broadly fall into two categories: image based [71] and model based [35, 95]. Image based methods synthesise novel views by interpolating between the captured 2D images. In contrast, model based methods require an underlying geometry proxy to synthesise novel views. Image based methods typically require camera views to be positioned in close proximity to allow interpolation for intermediate view synthesis, whereas model based methods can handle wide baselines between cameras. This thesis is only concerned with model based rendering techniques as currently an image based technique would require a dense array of cameras to capture the subject from all angles which would be impractical in this instance.

It is important to first make a distinction between two terms that are often used interchangeably in the literature. These are view dependent rendering and free viewpoint rendering. In this thesis, view dependent rendering refers to algorithms that account for changes in the surface appearance based on the viewpoint. These changes are typically caused by the lighting effects and material properties resulting in specular highlights. On the other hand, free viewpoint rendering techniques allow the user to interactively change the viewpoint from which the scene is rendered. This allows visualisation of the scene from a viewpoint that may not have been captured. A free viewpoint rendering method does not necessarily result in view dependent appearance.

The concept of view dependent rendering was first introduced by Debevec et al. [30]. It was used to create photo-realistic renderings of architectural scenes using approximate geometric models. A realistic appearance was achieved by sampling pixel intensities from multiple calibrated images onto the approximate geometry. The pixel intensities were weighted according to the view direction of the virtual camera used to navigate the scene, and the real cameras used to capture the scene. The closer the virtual camera gets to a real camera, the greater the contribution of its image.

Starck et al. [95] developed and released a view dependent free viewpoint video renderer (FVVR) and represents a typical state-of-the-art view dependent rendering pipeline. The FVVR is used throughout this thesis for evaluation purposes. For this reason, further details of the algorithm are described in Section 2.4.1.
2.4 Visualisation: Rendering from captured images

2.4.1 The Free Viewpoint Video Renderer

In this section, the FVVR [95] algorithm is presented as background work as this thesis builds upon this technique and the developed methods are compared directly with it. Key stages in the render cycle are illustrated in Figure 2.4. The algorithm starts by selecting a subset of cameras \( c_i \in C_{S_i} \) to be used for rendering, where \( c_i \) is the \( i^{th} \) camera in subset \( C_S \) consisting of \( N_S \) cameras. The selection is based on the view direction of the virtual viewpoint with respect to the capture cameras’ view direction. For each camera in the subset, a depth map \( D(i,t) \) of the geometry is rendered using the z-buffer of the graphics hardware. To minimise texture artefacts at the boundaries due to self occlusions, the depth map is rendered multiple times and overlaid with a small offset applied in the rendered screen domain. This has the effect of increasing the size of the mesh at its boundaries in the depth buffer. The visibility of a vertex is assessed by comparing the Euclidean distance between the vertex and camera centre with the depth of the vertex projected into the depth map of the respective camera, Equation 2.1.

\[
v(\bar{v}, c_i^D) = \begin{cases} 
1 & \text{if } d(\bar{v}, c_i^D) \leq p_d(\bar{v}, D(i,t)) \quad \text{(Visible)} \\
0 & \text{otherwise} \quad \text{(Not Visible)} 
\end{cases} \tag{2.1}
\]

where \( d(\bar{v}, c_i^D) \) returns the Euclidean distance between the vertex \( \bar{v} \) and the camera position in world coordinates \( c_i^P \) and \( p_d(\bar{v}, D(i,t)) \) returns the depth value obtained by projecting \( \bar{v} \) into the rendered depth map \( D(i,t) \) of the \( i^{th} \) camera of frame \( t \).

The vertex is assigned a weighting for each camera in the subset, Equation 2.2, which is then normalised with respect to the sum of all weights, Equation 2.3.

\[
w(\bar{v}, \bar{n}, c_i^D, c_v^D) = \max(c_i^D \cdot \bar{n}, 0) \cdot \max(c_v^D \cdot c_i^D, 0) \tag{2.2}
\]
where \( w(\vec{v}, \vec{n}, \vec{c}_i^D, \vec{c}_V^D) \) is a scalar weighting for a given vertex \( \vec{v} \) and associated normal \( \vec{n} \) in \( c_i \) the \( i \)-th camera and \( \vec{c}_V \) is the viewing direction of the virtual camera.

\[
\begin{align*}
w_n(\vec{v}, \vec{n}, \vec{c}_i^D, \vec{c}_V^D) &= \frac{v(\vec{v}, \vec{c}_i^D)w(\vec{v}, \vec{n}, \vec{c}_i^D, \vec{c}_V^D)}{\sum_{c_j \in \{c_S\}_{j=1}^{N_S} v(\vec{v}, \vec{c}_j^D)w(\vec{v}, \vec{n}, \vec{c}_j^D, \vec{c}_V^D)}} \tag{2.3}
\end{align*}
\]

where \( w_n(\vec{v}, \vec{n}, c_i, \vec{c}_V) \) is the normalised weighting.

The assigned weight is proportional to the angle between the virtual and capture cameras’ view directions. The final view dependent texture is given by multiplying the colour for each vertex in each visible camera by the normalised weight. The colour is sampled by projecting the vertex into the camera image domain using the camera projection matrix, Equation 2.4.

\[
\begin{align*}
f(\vec{v}, \vec{n}, \{C_S\}_{i=1}^{N_S}, \vec{c}_V) &= \sum_{c_j \in \{C_S\}_{j=1}^{N_S}} w_n(\vec{v}, \vec{n}, c_j, \vec{c}_V) p_c(\vec{v}, I(i,t)) \tag{2.4}
\end{align*}
\]

where \( f(\vec{v}, \vec{n}, \{C_S\}_{i=1}^{N_S}, \vec{c}_V) \) is the final view dependent colour for a vertex on the model surface, \( p_c(\vec{v}, I(i,t)) \) is the colour sampled by projecting \( v \) into the \( i \)-th cameras image domain.

For a novel viewpoint at time \( t \), the rendering process is performed for the multiple camera image set at that time point. This process is illustrated in Figure 2.5.

---

Fig. 2.5 View dependent rendering: the captured images are weighted proportionally to the view direction of the virtual camera with respect to the capture cameras.
In order to render at interactive frame rates, the mesh, images, silhouettes and camera calibration data is preloaded to the GPU and the programmable graphics pipeline must be utilised, e.g. OpenGL Shading Language. The algorithm is computationally intensive, requires multiple off screen renders, and demands significant amounts of GPU memory. For these reasons, the application is limited to desktop computers with a powerful dedicated graphics card.

### 2.5 Multiple camera studio systems

The University of Surrey’s capture studio consists of eight HD cameras positioned uniformly around an eight meter diameter circle giving full 360 degree coverage of the subject with 45 degree separation between camera views. The cameras are positioned approximately two and a half metres above the ground giving a usable capture volume of up to 5x5x2.5 metres. Capture volume and image resolution can be traded off depending on the type of performance being captured. This is a common studio configuration and several similar examples can be found in the literature [20, 28, 91, 104, 108]. The ceiling is lined with fluorescent strip lighting to create a uniform ambient illumination and the background is controlled using a blue chroma key cloth. The current 4D reconstruction pipeline uses the model free method of Starck and Hilton [94] and a global temporally consistent mesh representation over multiple sequences is achieved using the method of Budd et al. [17]. Data from this system is the primary input to the techniques developed in this thesis. An overview of the pipeline is shown in Figure 2.6.

Recently, Collet et al. [26] presented the first fully automatic end-to-end pipeline for free viewpoint video. The system makes use of 106 RGB and IR cameras to reconstruct highly detailed geometry together with an active IR speckle pattern projector to assist stereo matching in the IR spectrum. Visual hull models, created from extracted silhouettes, are combined with stereo RGB and IR reconstructions. Meshes are adaptively sampled to place more vertices in perceptually important areas, e.g. face and hands. Local temporal consistency is enforced by identifying key frames throughout the sequence and propagating consistent geometry to neighbouring frames using a non-rigid iterative closest point algorithm [70]. Texture maps were extracted on a per frame basis by blending multiple camera images together to make a seamless texture map. Texture map sequences and geometry were encoded into the MPEG H.264 AVC [118] format which can be streamed at equivalent bit rates as video streaming.
services. This system was able to successfully reconstruct a variety of subjects, e.g. single and multiple people interacting with objects and animals.

One of the state-of-the-art systems for multiple camera capture uses active illumination and is known as a light stage [29]. Light stage systems perform time multiplexed capture of the subject under different known lighting conditions. This allows the extraction of detailed surface normals and material properties [3, 47, 76], as well as creating fully relightable subjects [47].

Light stage systems have also made use of polarised light to separate diffuse and specular appearance surface components to assist appearance and geometry reconstruction [3]. Results from these systems have demonstrated the ability to reconstruct skin pore level detail in facial capture [3]. These systems produce impressive results and have been used in film productions including Spiderman (2004), Superman Returns (2006), The Curious Case of Benjamin Button (2008), Avatar (2009), and many others. However, light stage systems are expensive, require precise control of illumination conditions synchronised to the camera shutters and captured at high frame rates. This restricts the length of captured data, requires
2.6 4D Video Datasets

A huge amount of storage and multiple lighting conditions make background subtraction challenging.

Table 2.1 compares multiple camera systems reported in the literature as well as information collected from the websites of research institutions. Academic research has resulted in several spin out companies specialising in accurate body measurement for health fitness and fashion using statistical body models and markerless performance capture e.g. Body Labs [114], 4D Views [112] and The Captury [115].

Table 2.1 Comparison of systems reported in literature. Terms used are Model based (MB), model free (MF), active (ACT), passive (PAS), polarised (POL)

<table>
<thead>
<tr>
<th>System</th>
<th>Cameras</th>
<th>Modalities</th>
<th>Illumination</th>
<th>Reconstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collet [26]</td>
<td>106</td>
<td>RGB+IR</td>
<td>PAS-RGB + ACT-IR</td>
<td>MF</td>
</tr>
<tr>
<td>Inria [116]</td>
<td>68</td>
<td>RGB</td>
<td>PAS-RGB</td>
<td>VH</td>
</tr>
<tr>
<td>Kanade [57]</td>
<td>51</td>
<td>RGB</td>
<td>PAS-RGB</td>
<td>VH+MVS</td>
</tr>
<tr>
<td>Light Stage X [29]</td>
<td>7</td>
<td>RGB+POL</td>
<td>ACT-RGB</td>
<td>MB</td>
</tr>
<tr>
<td>Starck [94]</td>
<td>8</td>
<td>RGB</td>
<td>PAS-RGB</td>
<td>VH+MVS</td>
</tr>
<tr>
<td>Vlasic [108]</td>
<td>8</td>
<td>RGB</td>
<td>PAS-RGB</td>
<td>MB</td>
</tr>
</tbody>
</table>

2.6 4D Video Datasets

This section presents details of 4D video datasets used for evaluation throughout this thesis. These datasets were selected as they cover model based and model free performance capture pipeline and used a variety of reconstruction methods. These dataset also include a range of subjects including full body, cloth and face. All datasets used for evaluation are publicly available for research purposes [23, 61, 94, 108].

Table 2.2 Overview of datasets used for evaluation. Terms used are Model based (MB), model free (MF), visual hull (VH), multiple view stereo (MVS), temporal alignment (TA), linear blend skinning (LBS), non-rigid shape matching (NRSM)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cameras</th>
<th>Mesh</th>
<th>Frames (Sequences)</th>
<th>Reconstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4DVT: Dan [23]</td>
<td>8</td>
<td>5330</td>
<td>254 (8)</td>
<td>MF:VH+MVS+TA</td>
</tr>
<tr>
<td>SurfCap: Roxanne [94]</td>
<td>8</td>
<td>4950</td>
<td>432 (10)</td>
<td>MF:VH+MVS+TA</td>
</tr>
<tr>
<td>MIT: Samba [108]</td>
<td>8</td>
<td>20,000</td>
<td>175 (1)</td>
<td>MB:LBS+NRSNM</td>
</tr>
<tr>
<td>Face [61]</td>
<td>5</td>
<td>5248</td>
<td>354 (1)</td>
<td>MF:MVS+TA</td>
</tr>
<tr>
<td>Cloth [61]</td>
<td>5</td>
<td>768</td>
<td>322 (1)</td>
<td>MF:MVS+TA</td>
</tr>
</tbody>
</table>
2.6.1 4D Video Textures Dataset

The 4D video textures dataset [23] was recorded in the University of Surrey’s multiple camera studio against a blue chroma cloth background. The subject is a male in a red sweater with black horizontal stripes, dark blue jeans and brown shoes. The subject is captured from eight Thompson viper cameras at full HD resolution (1920x1080). Cameras are positioned uniformly around a circle at approximately 45 degree separation, giving full 360 degree coverage of the subject. The subject performs typical game character motions such as walk, jog, vertical jumps and horizontal jumps. Sequences typically range from 20 to 50 frames in length. Reconstruction was performed using the method of Starck et al. [94] and a temporally consistent geometry representation was created using the method of Budd et al. [17]. The mesh consists of 5330 polygons. Examples of the dataset are shown in Figure 2.7. This is one of the simplest datasets used in the evaluation, as the character is wearing fairly tight fitted clothing and the hair of the character is short. The challenges that arise from this dataset are that the colour calibration was not performed prior to capture meaning that the colour response between cameras varies. Also, there is an error in the calibration of camera 5 leading to quite large reprojection errors.

![Figure 2.7](image)

(a) Captured camera images. (b) Extracted silhouettes. (c) Examples of temporally consistent geometry. (d) Projectively textured geometry.
2.6 4D Video Datasets

2.6.2 The SurfCap Dataset

The SurfCap dataset was recorded in the University of Surrey’s multiple camera studio using eight Thompson viper full HD resolution (1920x1080) cameras with a blue chroma cloth background. The released dataset contains two characters, JP and Roxanne. The Roxanne dataset consists of a female character with long brown hair in a ponytail, light green commando shorts and a dark green vest. The subject, Roxanne, performs a set of game character style motions including walking, running, jumping and punching. Sequences range from 30 to 60 frames in length. Reconstruction was performed using the method of Starck et al. [94] and a temporally consistent mesh representation was created using the method of Budd et al. [17]. The mesh sequences consist of 4950 polygons. The Roxanne dataset is slightly more challenging than the 4D video textures character, as the hair and head scarf exhibit large deformations that change size and shape rapidly during fast motions. This is particularly challenging for surface tracking algorithms. Example images, silhouettes, geometry and projectively textured geometry are shown in Figure 2.8.

![Example images, silhouettes, geometry, and projectively textured geometry for Roxanne dataset.](image)

Fig. 2.8 Roxanne dataset walk sequence frame 28 cameras 0, 3, 4 and 5. (a) Captured camera images. (b) Extracted silhouettes. (c) Examples of temporally consistent geometry. (d) Projectively textured geometry.
2.6.3 MIT T-Samba

The MIT T-Samba sequence was released by Vlasic et al. [108]. The dataset features a female subject in a red t-shirt and a black skirt performing a samba style dance routine. The sequence consists of 175 frames recorded from eight (1600x1200 resolution) cameras at 25 FPS. Prior to the multiple camera capture session, a high resolution laser scan of the subject was undertaken, shown in Figure 2.3a. This scan serves as the template that all reconstructed frames are derived from. The template mesh is made up of approximately 20k polygons. This sequence is challenging due to the non-rigid dynamics of the skirt which can appear rigid due to the reconstruction method. Example images, silhouettes, geometry and projectively textured geometry are shown in Figure 2.9.

![Example data from MIT Samba dataset. (a) Captured images from MIT dataset T-Samba sequence. (b) Extracted silhouettes. (c) Examples of temporally consistent geometry. (d) Projectively textured geometry.](image-url)
2.6 4D Video Datasets

2.6.4 Cloth and Face Datasets

Although the primary focus of this work is targeted towards full body performance capture, wherever possible the methods are also evaluated against 4D video of general objects. In this thesis, evaluation is also performed against 4D video of face and cloth performance. Both face and cloth datasets were recorded using the same setup of five full HD resolution (1920x1080) cameras giving a 180 degree view of the subject. Initial geometry reconstruction was given by performing stereo matching in a graph cut formulation [83] for each pair of stereo cameras. Depth maps were filtered and converted to clouds of orientated points which are fused into a single triangular mesh using Poisson reconstruction [58]. The final temporally consistent mesh sequence was performed using the method of Klaudiny et al. [61].

The face dataset features a male subject performing a series of facial expressions. The sequence consists of 355 frames and the mesh has 5248 polygons. Figure 2.10 show examples of the image data, temporally consistent geometry and projectively textured mesh from the face dataset.

![Example data from face dataset](image)

Fig. 2.10  Example data from face dataset. (a) Capture images from Face dataset. (b) Temporally consistent geometry. (c) Projectively textured geometry rendered from a novel viewpoint.
The cloth dataset consists of 322 frames of a blue dress with a green floral pattern. In the sequences, the cloth is deformed by the motion of the body and folds appear and disappear at various speeds. The mesh consists of 768 polygons. Figure 2.11 show examples from the cloth dataset of the image data, temporally consistent geometry and projectively textured.

![Example data from cloth dataset](image)

(a) Capture images from Cloth dataset. (b) Temporally consistent geometry. (c) Projectively textured geometry rendered from a novel viewpoint.

2.7 Conclusion

Human performance capture has fascinated artists, scientists and engineers for well over a century. The range of motion exhibited by the human body make accurate and robust capture of performance an extremely challenging problem. This chapter has given a history of human performance capture from early 2D hand drawn characters, and the marker based systems that are still widely used in production pipelines to the current state-of-the-art in 4D markerless performance capture.
Markerless 4D performance capture is an active area of research which has led to several research groups forming companies around the technology, *e.g.* 4D View solutions, The Captury and Body Labs. This demonstrates confidence that this technology has the potential to revolutionise content creation for television, film and game production. It also has applications in many other areas such as medical imaging and diagnosis, sport broadcast and athlete performance analysis as well as creating realistic training and simulation environments in AR and VR applications.

It is now possible to define *4D Video* as the calibrated camera images from a multiple camera capture with the temporally consistent geometry that can be produced by model based or model free approaches. 4D video is the input to all work presented in this thesis.
Chapter 3

Multiple Layer Texture Representation

3.1 Introduction

Four dimensional performance capture combined with view dependent rendering produces highly realistic virtual characters with a visual quality approaching that of the captured video. These technologies have applications in the creation and display of digital assets for game, broadcast and film production industries. Photo-realism of the final rendered model is achieved by sampling and blending the appearance directly from the captured video frames. Existing free viewpoint video rendering techniques demand high computational resources, which restricts rendering to be performed locally on relatively high performance desktop computers. These requirements make current view dependent rendering techniques unsuitable for platforms with limited resources, such as mobile devices and web browsers.

The major bottleneck preventing view dependent rendering moving to alternative, lightweight platforms comes from the high storage and bandwidth overheads of the captured 4D video data, e.g. a ten second clip from eight HD resolution cameras requires approximately eight gigabytes (GB) of storage. Existing work has focused on either merging all camera views into a single seamless texture map to represent the appearance of the virtual character on a per frame basis [6, 43, 50, 56, 68, 110] or finding a suitable texture map over multiple frames [56, 106]. These texture representations are effective in addressing the storage requirements and are easily integrated into existing graphics pipelines. However, they result in significant
losses of both view dependent and dynamic surface appearance, both of which are important in preserving the realism of the captured video.

In this chapter, a novel multiple layer texture representation (MLTR) is introduced to address storage requirements of 4D video while maintaining the dynamic, view dependent surface appearance. Rendering is made computationally efficient by pre-computing occlusions and projective texturing, leaving only a minimal amount of texture blending to be performed at runtime.

### 3.2 Background

First, a review of related theory and work is presented including: texture mapping, UV parametrisation, multiple camera texture representation and efficient view dependent rendering. This is then summarised in order to motivate the contributions of the work presented in this chapter.

#### 3.2.1 Texture Mapping

The concept of texture mapping was first introduced by Edwin Catmull in 1974 [24]. The purpose was to display highly detailed surfaces while using relatively simple geometry. The core idea was to map an image to the surface of a model as shown in Figure 3.1. This was

Fig. 3.1 Texture mapping example using a cube consisting of eight vertices. (a) Example of a 2D texture map for a 3D cube geometry. (b) Texture map in the process of being applied to a cube model. (c) Final render of cube with texture map applied. The final surface of the cube is much more detailed than modelled by the geometry.
achieved by creating a mapping from the 3D model vertices to the 2D image domain, known as UV or texture coordinates. Often, it is required that the model be split into pieces to create 2D patches, also known as islands. This is a simple task in the case of a cube but becomes a non-trivial problem when handling complex shapes. At render time, the colour of each rasterised fragment generated by the 3D model is sampled from the image, known as a texture map, UV map or texture atlas.

The simplicity and efficiency of texture mapping made the method a core component of the graphics pipeline. Forty years since its introduction the idea remains the same, yet the advent of the programmable graphics pipeline has allowed the combination of multiple appearance maps to create complex renders in real-time. An example shown in Figure 3.2 demonstrates the power of texture mapping with a photo-realistic rendering of the earth [113]. This was achieved using a day texture, night texture, bump map, specular map and cloud map rendered on top of a simple sphere geometry.

Fig. 3.2 Photo-realistic rendering of the Earth (g) composed of day texture, night texture, bump map, specular map and cloud map, (a)-(e) respectively, with the use of a simple sphere geometry (f). Textures available from NASA Visible Earth Catalogue [117]. The rendered result was obtained by following a tutorial [113].
3.2.2 Surface Parametrisation

The generation of texture coordinates for simple shapes is trivial, e.g. a cube. Automatic generation of texture coordinates for generic shapes in such a way that makes them useful for artists is still an open problem in computer graphics. There are three main properties that the ideal surface parametrisation should have in order to make it useful for editing [46, 90]:

- Sampling from the model space to the texture space should be consistent, i.e. one pixel in UV space should represent one pixel in the camera domain from where it was sampled.
- Continuous mapping in the texture domain.
- Optimal use of available space in the UV domain.

![Fig. 3.3 Examples of surface parametrisation methods. (a) Spherical parametrisation [46]. (b) Model pelting [79, 92]. (c) Least-squares conformal maps [69].](image)

One parametrisation technique developed by Gotsman et al. [46] noted that a closed genus-0 manifold, a watertight mesh containing no self loops, is topologically equivalent to a sphere. Therefore, spherical parametrisation is a natural domain for the problem. However, spherical parametrisation introduces large distortions between the mesh and texture domain resulting in non-uniform stretching of mesh polygons. Piponi et al. [79] introduced the concept of model pelting which required the user to define a cut in the mesh. In a similar fashion to the pelting of an animal hide, the mesh is stretched and flattened. This gives a single large area in the UV domain, but also results in distortion between the mesh and texture domain. Levy
et al. [69] introduced least-squares conformal maps which take a different approach to those previously presented. Instead of creating a single seamless mapping, they introduce seams in areas of high curvature to create small patches. Each patch is then unfolded into a flat 2D island and packed into the UV domain.

Each of these methods have advantages and disadvantages. Spherical parametrisation and model pelting both produce a single seamless mapping that maximises the available space but results in distortion between the model and UV domain caused by non-uniform stretching of mesh polygons. On the other hand, methods that divide the mesh either through projections or based on surface curvature result in many small patches. Although these methods are able to effectively minimise distortion, they do not make optimal use of the space in the UV domain.

Currently, many production pipelines require skilled artists to create UV coordinates and texture maps. In this thesis, UV coordinates are generated manually by unwrapping the mesh based on user defined seams placed on the mesh surface as shown in Figure 3.4. This can be performed in most 3D manipulation applications, e.g. Blender, MeshLab and Maya. The mesh surface is split into semantically meaningful parts, i.e. head, arms, legs, torso. This will allow an artist to make edits in a more intuitive way than arbitrary placed seams. As the work in this thesis is based on temporally consistent mesh sequences, this task is performed once per character database as mesh topology remains constant.

![Fig. 3.4 Example of a UV coordinates for a human shape. (a) UV map coloured using the UV coordinates, human model in a T-pose rendered from the front (b) and back (c). This shows typically where the seams are placed.](image)
3.2.3 Multiple Camera Texture Representation

There have been many publications that tackle the problem of producing a texture map for reconstructed objects from multiple camera capture. This is desirable, as discussed in section 3.2.1, as it is an efficient way to give a complex appearance to a relatively simple 3D model. This is important as it may not be necessary, or sometimes even be possible, to model fine surface detail geometrically. The challenges that arise in this area originate from using inaccurate geometric models, erroneous camera calibration, and specular surfaces with view dependent appearance. This ultimately results in texture artefacts such as ghosting and blurring.

One method that has been used to great effect in the computer vision community is Markov random fields (MRF)[6, 43, 50, 56, 68, 110] to represent the spatial dependency of pixels in an image or vertices and polygons in a mesh surface. An overview of MRF is given in section 4.2.1. Typically in these approaches, an undirected graph is generated based on the mesh topology. The nodes of the graph represent mesh polygons and edges represent the polygon connectivity. A labelling of camera images to mesh polygons is sought to minimise an energy function consisting of a unary cost and a pairwise cost.

An early example of MRF applied to texture map generation was developed by Lempitsky and Ivanov [68]. In this work, the energy function maximised the quality of each polygon (unary cost) while minimising the colour difference between adjacent polygons (pairwise cost). Polygon quality was measured by the directness to the capture cameras using the camera direction vector and polygon surface normal. A seam levelling process was added to reduce visible seams between polygons caused by lighting effects and the colour response difference between cameras. Later, Gal et al. [43] extended this approach by allowing small translations of the polygons in the camera domain to account for geometric and calibration inaccuracies. This greatly reduced the visibility of seams.

Janko and Pons [56] also adopted an MRF based approach specifically for multiple camera capture of human performance. Their work presented two different methods to represent the multiple camera appearance, single-frame optimised and multi-frame optimised. The single-frame optimised approach used the appearance information from a single frame. This approach resulted in holes, due to camera coverage and occlusions, and slight misalignments between polygons caused by geometric and calibration errors. The resulting texture maps, one for each frame in the sequence, preserved the dynamic changes in the surface appearance (e.g. wrinkles in cloths and face expressions), but the view dependent surface appearance was
3.2 Background

lost. The \textit{multi-frame optimised} method found a single optimised texture map using multiple frames sub-sampled from a sequence. This approach produced a more complete texture map with noticeably fewer seams as each polygon had many more options from the use of multiple frames. However, the \textit{multi-frame optimised} approach lost both view dependent and dynamic surface appearance. Janko and Pons’ approach measured polygon quality by the area a polygon occupied in the camera domain, maximising sampling resolution. MRF-based texture map generation methods have successfully been used with tens of cameras \cite{6, 43, 56, 68} up to hundreds of cameras \cite{50, 110}.

Another general method that has produced impressive results is based on a super-resolution framework. Goldluecke \cite{45} used a variational formulation of the image formation model to super-resolve the appearance from multiple camera capture. This method was extended by Tsiminaki \textit{et al.} \cite{106} to cover a small temporal window of dynamic scenes, allowing areas of self occlusion to be recovered. These methods use computationally intensive and complex optimisation algorithms that require hours of computation to process a single frame.

Ziegler \textit{et al.} \cite{122} not only tackled the problem of multiple camera texture representation but also of compression. In this work, a texture map for each frame is extracted by blending together camera views. An average texture map was computed by averaging together all extracted texture maps on a per-pixel basis. A wavelet based compression scheme was proposed using the mean texture map as a basis for compression. This achieved a 50:1 compression ratio. A limitation of this approach is that by blending camera views, the view dependent appearance information is lost meaning that the captured video cannot be recovered.

3.2.4 Efficient View Dependent Rendering

Debevec \textit{et al.} \cite{31} addressed the efficiency of view dependent rendering with the introduction of the view-map data structure. A view-map precomputed visibility and selected the $N$-best views of each mesh polygon to be used for rendering (typically $N = 3$). This eliminated the need for off-screen rendering and the additional texture buffers required for depth maps. At render time, the appearance of the polygons were sampled from the original captured images. The work presented in this chapter also pre-computes and selects the best views of each polygon for efficient view dependent rendering, but goes a step further by resampling the appearance into a hierarchical set of texture maps eliminating redundant information in the captured camera frames.
3.2.5 Summary and Motivation

This section has presented an overview of texture mapping, surface parameterisation, several state-of-the-art approaches to multiple camera texture representation and efficient view dependent rendering. A compact and efficient representation of 4D video is an important problem to solve in order to overcome its prohibitive storage and transfer overheads. Current approaches achieve large storage reductions simply by discarding huge amounts of visually important information. The major disadvantage of previous approaches is that none preserve both the view dependent and dynamic changes in surface appearance, which is essential to reproduce the realism of the captured video. This motivates the need for a new compact representation of 4D video that maintains the view dependent and dynamic changes in the surface appearance.

3.3 Problem Formulation

Given a 4D video sequence captured from $N_C$ cameras consisting of an image and a binary silhouette mask per camera per frame and a temporally consistent mesh sequence, a set of $N_L$ texture map layers can be generated for each time frame $t$ where $N_L \leq N_C$. A straightforward approach would be to map each camera to a separate texture map $N_L = N_C$. However, this would not take advantage of the spatial or temporal redundancy in the input 4D video. Instead, a hierarchy of $N_L$ texture maps ordered according to the visibility of each mesh polygon such that the first texture layer resamples from the camera view from which each polygon is most visible. Polygon visibility is evaluated based on the angle between the camera view direction and surface normal [31, 68] taking into account occlusions. Other criteria such as camera sampling resolution [56, 91] could also be incorporated but are not required for uniform camera spacing as used throughout this thesis.

This results in a layered hierarchy of texture maps $\{L(j,t)\}_{j=1}^{N_L}$ with the $N_C$ input camera views resampled to a set of $N_L$ texture maps. Due to the ordering of the layers based on surface visibility, the view dependent appearance information required to maintain view dependent rendering quality can be represented with $N_L \ll N_C$. As the MLTR stores the appearance in the UV domain which is mapped to the 3D model surface using texture coordinates, the visibility of each polygon in the capture cameras is built into the representation. The camera identity from which each polygon is sampled is also encoded into the representation and
used for efficient view dependent rendering. An overview of the approach is shown in Figure 3.5 and key stages in the processing pipeline shown in Figure 3.6.

The calibrated cameras used to capture the scene have several associated attributes and can formally be defined as follows:

$$\{\{C(i,t)\}_{i=1}^{N_C}\}_{t=1}^{N_T} = \{\{K(i), R(i,t), \vec{t}(i,t), I(i,t), S(i,t)\}_{i=1}^{N_C}\}_{t=1}^{N_T}$$  \hspace{1cm} (3.1)$$

where $K(i)$ is a 3x3 matrix of intrinsic parameters, $R(i,t)$ is a 3x3 orthonormal rotation matrix, $\vec{t}(i,t)$ is the translation vector, $I(i,t)$ represents the captured image and $S(i,t)$ is the extracted silhouette. All attributes are defined with respect to the $i$th camera of $N_C$ cameras and at frame $t$ of $N_T$ frames. Camera images are assumed to have been undistorted allowing a pinhole camera model to be used.

A temporally consistent mesh sequence can be defined as:

$$\{M(t)\}_{t=1}^{N_T} = \{V(t), U, W\}_{t=1}^{N_T}$$  \hspace{1cm} (3.2)$$

where $\{M(t)\}_{t=1}^{N_T}$ is a temporally consistent mesh sequence of length $N_T$ frames composed of vertex positions $V(t)$ for frame $t$, a set of texture coordinates $U$ and mesh connectivity $W$. $U$ and $W$ remain constant over all $N_T$ frames.
The output layer sequence consists of two components: the colour texture map layers and camera assignment layers. This can be formally defined as:

\[
\{\{L(j,t)\}_{j=1}^{NL}\}_{t=1}^{NT} = \{\{L_a(j,t),L_t(j,t)\}_{j=1}^{NL}\}_{t=1}^{NT} \quad (3.3)
\]

where \(L(j,t)\) is the \(j^{th}\) layer for frame \(t\), consisting of a three channel colour image \(L_t(j,t)\) of the appearance information and a single channel image \(L_a(j,t)\) which stores the camera assignment of each pixel of \(L_t(j,t)\).

Fig. 3.6 Multiple layer texture representation processing and rendering pipeline: offline operations (gold boxes) produce the MLTR which is stored to use in the rendering stage. The simplified rendering pipeline (blue boxes).
3.3 Problem Formulation

3.3.1 Extraction

The algorithm for extracting the MLTR from a 4D video frame is presented below.

---

**INPUT**
Temporally consistent mesh $M(t)$ with texture coordinates $U$, calibrated cameras $\{C(i,t)\}_{i=1}^{NC}$, captured images $I(i,t)$, extracted silhouettes $S(i,t)$ and rendered depth maps $D(i,t)$.

**PROCESS**
For each polygon in the $j^{th}$ layer of frame $t$:

1. **Assess Visibility**: The visibility of a polygon is assessed in every camera by projecting each vertex, Equation 3.5, into $D(i,t)$, the rendered depth map for each camera. A polygon is considered visible if all vertices pass a depth test shown in Equation 3.7, the angle between the vertex normal and camera view direction is $<90$ degrees, and all vertices project within the extracted silhouette, $S(i,t)$.

2. **Rank Cameras**: Visible cameras are assigned a score, Equation 3.8, equal to the cosine of the angle between the vertex normal and the camera view direction. Cameras that are not considered to be visible are assigned a score of 1000 as this is much greater than the maximum score of a visible camera.

3. **Sort Cameras**: The list of cameras is sorted based upon the assigned score from lowest to highest using a bubble sort algorithm. This orders the cameras in terms of directness to the vertex normal.

4. **Sample Colour**: The colour is then sampled by projecting the polygon into the camera image $I(i,t)$ from the $j^{th}$ camera in the sorted list and stored in the RGB colour channels. In the alpha channel, a grey-scale value is stored which identifies the selected camera. If the polygon is not visible in the $j^{th}$ camera, the polygon is discarded.

**OUTPUT**
$L(j,t)$, the $j^{th}$ layer of frame $t$. 

---
A 3D point is projected into the camera domain using the intrinsic and extrinsic matrix as shown in Equation 3.4. This is shown for any camera at any frame. For convenience variables \( i \) and \( t \) have been dropped.

\[
\mathbf{p}' = \mathbf{K} \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}
\]  

(3.4)

where \( \mathbf{p} \) is the original position in 3D space, \( \mathbf{K} \) is the matrix of intrinsic camera parameters, \( \mathbf{R} \) is the rotation matrix, \( \mathbf{t} \) is the translation vector, \( \mathbf{0}^T \) is a transposed 3D vector of zeros, \( \mathbf{p}' \) is the 3D position mapped to the camera image domain. This can be expanded as shown in Equation 3.5.

\[
\begin{bmatrix} \bar{p}'_x \\ \bar{p}'_y \\ \bar{p}'_z \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & \bar{t}_x \\ R_{2,1} & R_{2,2} & R_{2,3} & \bar{t}_y \\ R_{3,1} & R_{3,2} & R_{3,3} & \bar{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{p}_x \\ \bar{p}_y \\ \bar{p}_z \\ 1 \end{bmatrix}
\]  

(3.5)

where \( \mathbf{p} \) is the 3D position expressed using homogeneous coordinates. \( \mathbf{R} \) is a 3x3 orthonormal rotation matrix and \( \mathbf{t} \) is the translation vector that together describe the orientation and pose of the camera in the world coordinate system. \( f_x \) and \( f_y \) are vertical and horizontal focal lengths expressed in pixel units, \( c_x \) and \( c_y \) give the principal point on the image also in pixel units, and \( s \) is the skew factor. These five parameters describe how a 3D point in the camera coordinate system is projected into the image domain. \( \mathbf{p}' \) is the 3D point transformed into the camera coordinate system which can be converted to pixel locations using Equation 3.6.

\[
\begin{bmatrix} \bar{u}_x \\ \bar{u}_y \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{p}'_x/\bar{p}'_z \\ \bar{p}'_y/\bar{p}'_z \\ \bar{p}'_z/\bar{p}'_z \end{bmatrix}
\]  

(3.6)

where \( \bar{u} \) is the projection of \( \mathbf{p}' \) into the image plane in pixel units.

Depth testing is performed by comparing the Euclidean distance between the camera position and vertex position against the distance in a depth map rendered from the viewpoint of a camera. Visibility testing is shown in Equation 3.7.

\[
v(\bar{v}, e^p_i, p_d(\bar{v}, D(i, t))) = \begin{cases} 1 & \text{if } d(\bar{v}, e^p_i) \leq p_d(\bar{v}, D(i, t)) \\ 0 & \text{otherwise} \end{cases}
\]  

(3.7)
where $\vec{v}$ is a 3D vertex, $\vec{c}_i^P$ is the position of the $i^{th}$ camera in the world coordinate system, and $p_d(\vec{v}, i, t)$ is the depth value acquired by projecting the vertex position into the camera depth map $D(i, t)$ for the $i^{th}$ camera in frame $t$.

The score assigned to a visible camera is given by the angle between the polygon normal and camera viewing direction as shown in equation 3.8.

$$w(\vec{p}_n, \vec{c}^D_i) = 1 - \vec{c}^D_i \cdot \vec{p}_n$$

where $\vec{c}^D_i$ is the normalised direction vector of the $i^{th}$ camera and $\vec{p}_n$ is the polygon normal.

The size of the extracted texture map can be arbitrarily chosen. However, it is desirable that the texture information is not down sampled, resulting in data loss, and not up-sampled as this would result in an unnecessary increase in data size. In this thesis, three sizes of texture map layers are considered: 512x512, 1024x1024 and 2048x2048 pixels. These power of two texture sizes were chosen as they can be efficiently handled by all versions of OpenGL, whereas non-power of two sizes are not supported in some versions of OpenGL.

### 3.3.2 Storage

The output from the MLTR extraction process is $N_T \times N_L$ RGBA images containing texture colour and camera assignment. In this thesis, two options are considered for storage of the MLTR. These are:

**Image based storage**

Each layer of every frame is stored as a single RGBA image encoding colour and camera assignment. The PNG image format is used as it is a lossless compression format. Lossless compression is necessary to fully recover the camera assignment. A lossy format, such as JPEG, introduces compression artefacts and is unable to handle four channels.

**Video based storage**

Layer sequences $\{L_t(i, t)\}_{t=1}^{N_T}$ are encoded into a MPEG video stream and camera assignment $L_{ai}(i, t)$ is stored in a single channel greyscale PNG image. When loaded, each video frame
is decoded and combined with the corresponding camera assignment into a four channel image. This allows the video codec to take advantage of the temporal redundancy in the representation increasing the storage reduction.

In this thesis, all video streams are encoded with H.264 AVC codec [118] using a variable bit rate and constant quality at the highest setting, resulting in near lossless compression.

3.3.3 Rendering

The online rendering stage computes the view dependent texture for the model based upon a user selected virtual camera position. When rendering, \( n_L \) layers are made available based upon the desired quality and available system resources where \( n_L \leq N_L \). An overview of the rendering process shown in Figure 3.6 and 3.7 with details given below:

**INPUT:**
Temporaly consistent mesh \( M(t) \) with texture coordinates \( U \), cameras \( \{ C(i,t) \}_{i=1}^{N_C} \), layers \( \{ L(j,t) \}_{j=1}^{n_L} \) and user-selected virtual camera position \( C_v \).

**PROCESS:**
For each output fragment when rendering \( M(t) \):

1. **Assign layer pixels to cameras:**
   Assign each pixel from every \( n_L \) layers to one of the \( N_C \) cameras. This is done by comparing the camera assignment stored in the alpha channel to the ID of each camera. Pixels which cannot be assigned to a camera are discarded.

2. **Calculate contribution of each layer:**
   For each pixel/camera tuple of each layer, calculate weighting, Equation 2.3.

3. **Calculate view-dependent colour:**
   The weighting for all \( n_L \) matched cameras is normalised and the final view-dependent colour of the fragment is calculated, Equation 3.9.

**OUTPUT:**
View-dependent textured mesh
For each output fragment of the mesh, the contribution each layer makes to the final colour is calculated. This is based upon the direction of the user selected virtual camera position. In order to do this, the identity of each camera must first be defined. To identify the cameras, the grey scale camera ID stored in the alpha channel is used. Once a camera has been identified, the weighting is calculated as shown in Equation 2.3. If a camera is not able to be identified, because the mesh element was not visible in any capture camera, the pixel is discarded. This method is independent of the weighting scheme, however it is required that the sum of the weights is one. The weighting scheme used throughout this thesis is that proposed by Starck et al. [95] which ensures smooth transitions between camera views. This is done for evaluation purposes. The weighting function is shown in Equation 2.2 and Equation 2.3. Once weightings have been calculated for all $n_L$ layers, the weights are normalised. The final view dependent colour for the output fragment is given by summing together the RGB colour for each layer scaled by the associated weighting, shown in Equation 3.9.

$$f(\{L_i\}_{i=1}^{n_L}, \{c_j\}_{j=1}^{N_C}, \vec{c}_D, \vec{v}, \vec{n}) = \sum_{i=1}^{n_L} w_i(\vec{v}, \vec{n}, \vec{c}_D) \text{RGB}_i$$  \hspace{1cm} (3.9)$$

where $f$ is the fragment colour given a set of layers $\{L_i\}_{i=1}^{n_L}$, camera parameters $\{c_j\}_{j=1}^{N_C}$ and virtual camera direction $\vec{c}_D$, $n_L$ is the number of available layers, $w_i()$ is the normalised weighting for a fragment in the $i^{th}$ layer with a camera assignment of the $j^{th}$ camera, $\text{RGB}_i$ is the colour for a fragment of the $i^{th}$ layer.
3.4 Results and Evaluation

In this section, the MLTR is quantitatively evaluated against the free viewpoint video renderer (FVVR) by Starck et al. [96], which represents a typical state-of-the-art approach. Both approaches are compared in terms of rendering quality, required storage, and GPU memory. Table 3.1 gives a summary of the evaluated datasets including cloth, face and full body capture subjects. All tests were performed on a Dell Optiplex 9010 desktop computer running Ubuntu 12.04 with an Intel i7 processor, 8GB of RAM and a Nvidia GeForce FX 640 graphics card. The extraction algorithm was implemented using C++11, OpenGL 4.3 and OpenGL Shading Language 1.5. The rendering algorithm was implemented using C++11, OpenGL Shading Language 1.5 and OpenSceneGraph 3.2.1 graphics library.

Table 3.1 Summary of datasets used for evaluation showing number of cameras, number of frames and data size for raw storage of captured data, lossless image based storage and video based storage in megabytes (MB).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cameras</th>
<th>Frames</th>
<th>Captured Data (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloth</td>
<td>5</td>
<td>320</td>
<td>Raw Data PNG Images Compressed Video</td>
</tr>
<tr>
<td>Dan (Walk)</td>
<td>8</td>
<td>28</td>
<td>19000 17200 906</td>
</tr>
<tr>
<td>Face</td>
<td>5</td>
<td>355</td>
<td>1600 716 57</td>
</tr>
<tr>
<td>Roxanne (Walk)</td>
<td>8</td>
<td>31</td>
<td>21000 12900 386</td>
</tr>
</tbody>
</table>

3.4.1 Error Metrics

Image Quality: Structural Similarity Index Measure

The structural similarity index measure (SSIM) is a full reference image metric used to assess image quality. It has been shown to be more correlated to the human perceptual system than other methods such as PSNR [111].

\[
SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}
\]

where \(\mu_x\) is the mean of \(x\), \(\mu_y\) is the mean of \(y\), \(\sigma_x\) is the variance of \(x\), \(\sigma_y\) is the variance of \(y\), \(\sigma_{xy}\) is the covariance of \(x\) and \(y\), \(c_1 = (k_1L)^2\) and \(c_2 = (k_2L)^2\) are constants used to prevent division by zero, where \(L\) is the dynamic range of the images and by default \(k_1 = 0.01\) and \(k_2 = 0.03\).
3.4 Results and Evaluation

Storage Reduction

Image sequences from each camera are encoded into a video stream \( \{I(i,t)\}_{i=1}^{N_T} \rightarrow I_V(i) \) where \( \{I(i,t)\}_{i=1}^{N_T} \) is the image sequence and \( I_V(i) \) is the resulting video stream of length \( N_T \) from the \( i^{th} \) camera. The total storage for a 4D video sequence is therefore the sum of the video streams for all \( N_C \) cameras as shown in Equation 3.11.

\[
s_v(N_C) = \sum_{i=1}^{N_C} I_V(i)
\]  

(3.11)

where \( s_v \) is the total storage for a 4D video sequence, \( N_C \) is the number of cameras and \( I_V(i) \) is the video stream of the \( i^{th} \) camera.

The total storage for a 4D video sequence represented by the MLTR is calculated using the layer texture sequence encoded into a video stream \( \{L_t(j,t)\}_{t=1}^{N_T} \rightarrow L_V^t(j) \) where \( \{L_t(j,t)\}_{t=1}^{N_T} \) is original and \( L_V^t(j) \) is the video stream for the \( j^{th} \) layer. The total storage is therefore the sum of these plus all camera assignment maps stored as grey scale images for \( n_L \) layers, as shown in Equation 3.12.

\[
s_M(n_L) = \sum_{j=1}^{n_L} L_V^t(j) + \sum_{j=1}^{n_L} \sum_{t=1}^{N_T} L_a(j,t)
\]  

(3.12)

where \( s_M(n_L) \) is the storage for \( n_L \) layers, \( L_V^t(j) \) is the \( j^{th} \) layer sequence video stream and \( L_a(j,t) \) is the camera assignment of the \( j^{th} \) layer at frame \( t \).

The storage reduction can therefore be calculated as shown in Equation 3.13.

\[
r(n_L,N_C) = 1 - \frac{\sum_{j=1}^{n_L} s_M(j)}{s_v(N_C)}
\]  

(3.13)

where \( r(n_L,N_C) \) is the storage reduction for \( n_L \) layers and \( N_C \) cameras, \( s_M \) is the storage requirement for the MLTR when using \( n_L \) layers stored in a video stream and \( s_v \) is the storage requirement of the original captured data stored in a video stream.
3.4.2 Rendering Quality

To evaluate rendering quality, the model was rendered using the FVVR [95] and proposed MLTR method. An arbitrary viewpoint was chosen between two capture cameras, as this represents the hardest rendering case. Rendering quality was evaluated by comparing the FVVR and MLTR render using SSIM, presented in Section 3.4.1. Rendering quality results for evaluation datasets are presented in Figure 3.8 and rendered examples shown in Figure 3.9.

![Graphs showing rendering quality evaluation](image)

Fig. 3.8 Render quality evaluation when varying $n_L$ for different sizes. (a) Dan, (b) Roxanne, (c) Cloth and (d) Face datasets.

The results show in the case of Dan and Roxanne datasets that much of the appearance information is captured in the first three layers, $n_L = 3$, of the MLTR. In the case of the Cloth and Face datasets, the quality increases until $n_L = N_C$. This is caused by a difference in the camera arrangement used to capture the subject. The Cloth and Face datasets used 5 cameras with 180 degree view of a near flat surface meaning all the surface is visible in all cameras. While Dan and Roxanne datasets used a 360 degree, 8 camera setup with approximately 45 degree separation between cameras. This means that on average each polygon is seen by 3-4
cameras on average due to self occlusions. A small difference between the two approaches is how the visibility of the surface is calculated. The FVVR computes visibility online on a per fragment basis while MLTR computes this offline on a per polygon basis. This leads to a minor difference at boundaries and self occlusions. Differences are particularly noticeable at the edges of the arms and under the chin in Figure 3.9c highlighted by the heatmap.

Fig. 3.9 Comparison of FVVR (left) with MLTR (right) and the difference highlighted with heatmap (centre) of RGB colour difference. (a) Face, (b) Roxanne and (c) Dan datasets. All from an example frame and viewpoint between capture cameras.
3.4.3 Storage

This section compares the storage footprint of the original captured data against the MLTR. The comparison is made between the sum of total captured data when compressed in a video stream against the MLTR when varying $n_L$. Results for each of the evaluated datasets are presented in Table 3.2 with storage reductions shown in Figure 3.10.

Table 3.2 MLTR storage requirements v.s. captured video streams encoded with the same video codec for varying numbers of layers $n_L$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Video Size (MB)</th>
<th>MLTR</th>
<th>$n_L$ v.s. Storage (MB)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloth</td>
<td>906</td>
<td>512</td>
<td>14.9 31.4 47.3 62.8 76.2 - - -</td>
<td>1024</td>
<td>40.4 84.2 125.3 165.3 198.8 - - -</td>
<td>2048</td>
<td>106.4 219.3 324.3 426.4 511.2 - - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>57</td>
<td>512</td>
<td>1.3 2.9 4.5 5.6 5.8 5.9 5.9</td>
<td>1024</td>
<td>3.3 7.1 10.9 13.6 13.9 14.0 14.1 14.2</td>
<td>2048</td>
<td>8.0 17.3 26.2 32.2 33.1 33.4 33.7 34.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Face</td>
<td>386</td>
<td>512</td>
<td>9.7 20.4 32.3 43.9 53.7 - - -</td>
<td>1024</td>
<td>26.5 55.6 86.1 114.1 137.5 - - -</td>
<td>2048</td>
<td>83.8 172.9 258.3 332.2 394.2 - - -</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roxanne</td>
<td>61</td>
<td>512</td>
<td>1.4 3.2 5.1 6.7 6.9 7.0 7.0</td>
<td>1024</td>
<td>3.4 7.5 11.8 15.3 15.9 16.0 16.1 16.2</td>
<td>2048</td>
<td>8.3 18.0 27.8 35.3 36.7 37.1 37.4 37.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It can be seen that the MLTR offers a large storage reduction in all datasets. In the case of the Dan and Roxanne datasets, the use of $n_L > 3$ did not significantly increase the visual quality, but adds to the storage footprint due to $\geq L(4,t)$ containing appearance information. However, the layers are ordered by visibility and therefore $\geq L(4,t)$ does not make a significant contribution to the final view dependent texture. This is an advantage of storing the appearance information sorted by visibility. In the case of the Cloth and Face dataset, there is less of a storage reduction as there is more texture information present in $\geq L(4,t)$ due to the capture setup. As expected, layer size also affects the storage reduction. When using a 2048 sized texture map, the storage reduction is significantly lower. This is caused by the original data being scaled larger and therefore requires more storage space. It can also be seen that there is no increase in visual quality and in some cases, increasing the size to 2048 can negatively affect the visual quality.
3.4 Results and Evaluation

Fig. 3.10  Ratio of storage reduction when varying $n_L$ for different sizes expressed as a percentage. (a) Dan, (b) Roxanne, (c) Cloth and (d) Face datasets.

3.4.4 Memory

Another important consideration is the amount of GPU memory required by the rendering method. Here, a comparison is made between the FVVR and MLTR of the GPU memory requirements to render a single 4D video frame.

The FVVR crops the camera images and silhouettes into a 512x512 texture, one for each camera. A 32-bit floating point depth map is also rendered offscreen for each camera. This requires an additional 512x512 texture per camera. The FVVR GPU memory requirements are shown in Equation 3.14.

$$m_f(N_C) = N_C \frac{512^2 \cdot 8 \cdot 4}{8 \cdot 1024^2} + N_C \frac{512^2 \cdot 32}{8 \cdot 1024^2}$$  (3.14)
where $m_f$ is the GPU memory requirements in MB for the FVVR with $N_C$ cameras using four channel textures, an 8-bit colour representation for camera images and silhouettes, and a 32-bit single channel depth map.

In comparison, the MLTR requires only the texture layers and camera assignment to be loaded into a four channel texture per layer per frame with no additional buffers required for offscreen rendering. This is shown in Equation 3.15.

$$m_m(n_L, s_L) = n_L s_L^2 \cdot 8 \cdot 4 \cdot \frac{8}{1024^2}$$  (3.15)

where $m_m(n_L, s_L)$ is the GPU memory requirements in MB of the MLTR rendering method for a given number of layers $n_L$ and layer size $s_L$.

Memory reduction is calculated as shown in Equation 3.16.

$$r_m(N_C,n_L,s_L) = 1 - \frac{m_m(n_L,s_L)}{m_f(N_C)}$$  (3.16)

where $r_m(N_C,n_L,s_L)$ is the memory reduction between the two methods for $N_C$ cameras and $n_L$ layers of size $s_L$.

Figure 3.11 shows how the memory reduction of the MLTR changes with layer size $s_L$ and number of layers used $n_L$. Using the Dan or Roxanne dataset, $N_C = 8$, $n_L = 3$ $s_L = 1024$,

![Fig. 3.11 MLTR GPU memory reduction v.s. FVVR, expressed as a percentage.](image)

results in $m_f(8) = 16$ MB and $m_m(3, 1024) = 12$ MB, a reduction of 25% and no requirement for offscreen rendering. A layer size of 2048 significantly increases memory requirements, but this has been shown to have no effect on the visual quality. Therefore, the larger layer size is not a practical consideration.
3.4 Results and Evaluation

3.4.5 Dan Dataset

Here, examples are shown for the Dan dataset. Figure 3.12 shows an example of four texture layers from frame 0 of the walk sequence. Figure 3.13 show rendering examples from a variety of motions from arbitrary chosen viewpoints.

Fig. 3.12 MLTR example from Dan dataset. (a) Texture map layers 1-4. (b) Camera assignment layers 1-4, represented in colour.

Fig. 3.13 Rendering examples from arbitrary viewpoints of the Dan dataset using the MLTR. (a) JumpLow frame 10, (b) JumpLow frame 2 and (c) Walk frame 15.
3.4.6 Roxanne Dataset

Here, results are presented for the Roxanne dataset. Figure 3.14 shows frame 15 from the walk motion sequence represented using the MLTR. Figure 3.15 shows a subset of frames from the walk sequence rendered from a randomly chosen viewpoint.

Fig. 3.14 MLTR example from Roxanne dataset. (a) Texture map layers 1-4. (b) Camera assignment layers 1-4, represented in colour.

Fig. 3.15 Rendering examples from Roxanne dataset walk sequence using the MLTR.
3.4 Results and Evaluation

3.4.7 Cloth Dataset

Here, an example frame from the Cloth dataset represented using the MLTR is shown in Figure 3.16. Rendering examples from randomly chosen frames and viewpoints are shown in Figure 3.17.

Fig. 3.16 MLTR example from Cloth dataset. (a) Texture map layers 1-5. (b) Camera assignment layers 1-5, represented in colour.

Fig. 3.17 Rendering examples from Cloth dataset using the MLTR.
3.4.8 Face Dataset

Here, an example frame from the Face dataset represented using the MLTR is shown in Figure 3.18. Rendering examples from randomly chosen frames and viewpoints are shown in Figure 3.19.

Fig. 3.18 MLTR example from Face dataset. (a) Texture map layers 1-4. (b) Camera assignment layers 1-4, represented in colour.

Fig. 3.19 Rendering examples from the Face dataset using the MLTR.
3.5 Conclusions

This chapter has introduced a novel multiple layer texture representation which significantly reduces the storage requirements of 4D video. An evaluation has shown that this reduction can be > 90% with respect to the captured video when encoded using the same H.264 video compression. Accompanying the texture representation is a novel rendering scheme that avoids all offscreen rendering and additional buffers as well as reduce view dependent rendering to a simple texture look up. The representation was quantitatively evaluated against a typical state-of-the-art free viewpoint video rendering technique [95] and was shown to produce almost identical results as measured by the SSIM.

In the case of an eight camera 360 degree set up, used to capture the Dan and Roxanne datasets, it was shown that $n_L = 3$ at 1024x1024 resolution is able to achieve a storage reduction of > 90% without a significant impact on visual quality. In these datasets, using $n_L > 3$ does not improve the visual quality but adds to the storage and memory requirements. This is due to the layers $n_L \geq 4$ containing little appearance information and what is present does not significantly contribute to the view dependent surface appearance.

The contribution of this chapter opens up the possibility for view dependent rendering to be implemented on lighter weight platforms. This goal was achieved by Imber et al. [55] who demonstrated that the MLTR method was able to run at interactive rates on an Amazon Fire tablet.

In the presented method, cameras are assigned to the polygons in the texture layers based only on the angle between the polygon normal and camera direction vector. This gives rise to a jagged assignment of cameras over the surface of the mesh and high frequency flickering over time. Reducing these artefacts would allow a further storage reduction when using video compression due to increased spatial and temporal redundancies in the MLTR layer sequences. The MLTR also suffers from the same artefacts that affect Starck et al. [95] namely blurring and ghosting which occur due to errors in geometry and camera calibration. In the next chapter, methods are introduced to improve spatio/temporal assignment of cameras to polygons allowing a greater storage reduction and alignment to overcome texture artefacts caused by geometric and calibration errors.
Chapter 4

Optimal Representation of 4D Video

4.1 Introduction

In the previous chapter, a novel multiple layer texture representation (MLTR) was introduced which significantly reduced storage requirements of 4D video and enabled computationally efficient view dependent rendering. The storage requirement of 4D video was reduced by resampling the appearance into a hierarchical set of texture maps ordered by surface visibility, removing much of the redundant information. View-dependent rendering was made computationally efficient by eliminating depth testing, off screen rendering and additional buffers. This reduced view dependent rendering to a simple texture look-up and blending based on the selected viewpoint. However it was observed that within the MLTR, the camera assignment of polygons changed significantly over the mesh surface and flickered over time. This meant that video compression algorithms were not able to reduce the storage to the full potential due to the changes in camera assignment. The representation was also shown to suffer from the visual artefacts that affect view dependent rendering when using approximate geometry models and camera calibration.

In this chapter, methods are introduced to optimise the camera assignment of the MLTR in order to maximise the size of assignment patches over the surface of the mesh while minimising changes over time. The primary objective is to further reduce the storage footprint of the MLTR as well as minimise spatial and temporal artefacts. The hypothesis is that this optimisation will result in a greater storage reduction due to an increase in both spatial and temporal redundancy. The method is quantitatively evaluated to ensure the optimisation of camera labelling results in a storage reduction and has no negative impact.
on rendering quality. Another contribution of this chapter is an optical flow based multiple
camera alignment method. This is used when extracting the MLTR from a 4D video frame
and was developed to reduce blur and ghosting artefacts that are a result of using approximate
geometry and camera calibration. This reduces small errors that occur because a 3D point
will project to slightly different points on the real surface and ensures that the same surface
point is mapped to the same UV location in the texture layers of the MLTR. These two
contributions are required to satisfy the ideal properties of the *Optimal Representation of 4D
Video* which is also introduced in this chapter.

4.2 Background

First, an overview of several techniques from computer vision are given that form the
foundation of the work presented in this chapter. These techniques are graph-cuts, Markov
random fields (MRF) and optical flow. A review of state-of-the-art methods for texture
alignment of multiple camera data is also given.

4.2.1 Graph-Cuts in Computer Vision

Graph-cuts have been used extensively in computer vision for tasks including segmentation
[38], stereo [83] and 3D reconstruction [75, 88]. Problems tackled with graph-cuts utilise a
graph structure $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ which consist of vertices $v_i \in \mathcal{V}$ and edges $\{v_i, v_j\} \in \mathcal{E}$, where
each edge has a non-negative capacity $c(v_i, v_j)$. Vertices are used to represent any entity in the
problem domain and edges represent the dependencies between them. In image processing
problems, $\mathcal{V}$ typically represents image pixels or a group of pixels and $\mathcal{E}$ represent the local
neighbourhood surrounding each pixel or pixel group.

The goal in graph-cuts is to partition a graph into two disjoint subsets, $\mathcal{S}$ and $\mathcal{I}$, such that
$\mathcal{S} = \mathcal{V} - \mathcal{I}$. Two additional vertices, known as the source $v_s$ and the sink $v_t$, are introduced into
the graph. Finding the maximum flow between $v_s$ and $v_t$ was shown by the *min-cut/max-flow
theorem* to be equivalent to finding the minimum capacity cut in $\mathcal{G}$ that separates $v_s$ from $v_t$
such that $v_s \in \mathcal{S}$ and $v_t \in \mathcal{I}$. The capacity of the cut is given by $c(\mathcal{S}, \mathcal{I}) = \sum_{\{v_i \in \mathcal{S}, v_j \in \mathcal{I}\}} c(v_i, v_j)$.
The *min-cut/max-flow* algorithm forms the basis of many other graph based algorithms.
4.2 Background

4.2.2 Markov Random Fields

MRF is a graph based technique in which a discrete set of $N_L$ labels $\{l_i\}_{i=1}^{N_L} = \{l_1, l_2, ..., l_{N_L}\}$ are assigned to graph vertices, $v_i \in V$ of an undirected graph. A labelling, $\mathcal{L} : v_i \rightarrow l_i$, consisting of node/label pairs, is sought in order to minimise an energy function of the form shown in Equation 4.1.

$$e(\mathcal{L}) = e_D(\mathcal{L}) + e_S(\mathcal{L})$$  \hspace{1cm} (4.1)

where $e(\mathcal{L})$ is the total energy for a given labelling $\mathcal{L}$ consisting of the data term $e_D(\mathcal{L})$ and smoothness term $e_S(\mathcal{L})$.

The data term, also known as the unary term, ensures that the labelling is consistent with the observed data, shown in Equation 4.2.

$$e_D(\mathcal{L}) = \sum_{v_i \in V} e_d(v_i, l_i)$$  \hspace{1cm} (4.2)

where $e_D(\mathcal{L})$ is the energy of the data term given a labelling $\mathcal{L}$, $e_d(v_i, l_i)$ defines an energy of assigning label $l_i$ to the $i^{th}$ graph vertex $v_i$ from the vertex set $V$.

The smoothness term is used to represent the energy between connected graph vertices. These dependencies are defined by the graph edges $\{v_i, v_j\} \in E$, shown in Equation 4.3.

$$e_S(\mathcal{L}) = \sum_{\{v_i, v_j\} \in E} e_s(v_i, l_i, v_j, l_j)$$  \hspace{1cm} (4.3)

Problems of this form are known to be NP-hard and therefore, a globally optimal solution cannot be obtained. However, a strong local minima can be obtained through graph-cut based optimisation algorithms such as $\alpha\beta$-swap or $\alpha$-expansion.

The $\alpha$-expansion algorithm iteratively separates $\alpha$ labelled nodes from non-$\alpha$ labelled nodes using the min-cut/max-flow algorithm [13]. At each iteration, an attempt is made to expand the $\alpha$ region. The expanded region is only accepted if it lowers the total energy. The label represented by $\alpha$ is changed at each iteration and the expansion process is repeated until no expansion move lowers the total energy. At this point, the algorithm is said to have converged.
Kolmogorov and Zabih [63] presented a comprehensive study of the properties of energy functions that can be minimised with graph-cuts. The authors showed that each pairwise term in the energy function must be shown to be a metric for it to be minimised by the \(\alpha\)-expansion algorithm. A metric is defined as having the following properties:

- **Co-incidence Axiom:**
  \[ e_s(l_\alpha, l_\beta) = 0 \iff l_\alpha = l_\beta \quad (4.4) \]

- **Symmetry and non-negativity:**
  \[ e_s(l_\alpha, l_\beta) = e_s(l_\beta, l_\alpha) \geq 0 \quad (4.5) \]

- **Triangular inequality:**
  \[ e_s(l_\alpha, l_\beta) \leq e_s(l_\alpha, l_\gamma) + e_s(l_\gamma, l_\beta) \quad (4.6) \]

\(\alpha\beta\)-swap requires the smoothness term be a semi-metric and therefore, it does not require the triangular inequality condition to be fulfilled. This allows more general types of energy functions be solved with \(\alpha\beta\)-swap [14]. The \(\alpha\)-expansion algorithm has been shown to minimise to a known factor of the global minimum [14]. In this thesis, \(\alpha\)-expansion is used as it has been shown to converge faster and obtain a more optimal solution in many kinds of problems [14, 68, 99].

### 4.2.3 Optical Flow

Optical flow is a class of computer vision algorithms which finds corresponding features between two images. A key assumption that underpins optical flow algorithms is the brightness constancy assumption, Equation 4.7. This assumes that a pixel will remain the same colour between frames.

\[ I(x, y, t) = I(x + d_x, y + d_y, t + d_t) \quad (4.7) \]

The output of these algorithms is either sparse [73] or dense [15, 37] flow vectors. Sparse flow vectors give correspondence only for a set of interesting points between the two images whereas dense flow vectors give a correspondence for every pixel in the images. In this thesis, optical flow fields are visualised using either the HSV colour space, encoding both magnitude
and direction, or by overlaying coloured arrows which define the source and destination of the flow vectors. Figure 4.1 shows an example of both visualisations.

![Visualisation Methods](image)

Fig. 4.1 Methods for visualising optical flow fields. (a) HSV colour wheel is used to visualise optical flow as it shows magnitude and direction. (b) Overlay of arrows showing the source and destination of the flow vector.

Optical flow techniques have been used successfully to perform multiple camera texture alignment [21, 35]. Some state-of-the-art approaches are discussed in the next section. Results presented in this thesis use the OpenCV implementation (C++ version 2.4.10) of the Farnebacke [37] and Brox [15] optical flow algorithms.

### 4.2.4 Multiple Camera Texture Alignment

Methods for camera calibration and geometry reconstruction from wide baseline camera setups contain some degree of error. These errors are ultimately visible as texture artefacts, *e.g.* blur and ghosting, in the final rendering. These artefacts are caused by errors in geometry and calibration which result in reprojection errors occurring when a 3D point is projected to different points on the real surface in different camera views. There are several methods that attempt to reduce texture artefacts both online [23, 35] and offline [1, 101].

Eisemann *et al.* [35] introduced the concept of *Floating Textures* that reduced texture artefacts by performing optical flow correspondence in the rendered screen domain. This was achieved by rendering the model from a virtual camera viewpoint and projectively texturing the model using each camera which contributed to texturing. Optical flow was then performed between
the rendered images. The final texture was achieved by sampling the camera images that were adjusted by interpolated optical flow vectors based on the virtual viewpoint. This would effectively float the camera images over the mesh surface making small local adjustments. The computational complexity of floating textures, caused by running optical flow within the render loop, is a significant computational limitation for the practical application of this algorithm.

Casas et al. [23] extended the Floating Textures approach to allow texture synthesis between two 4D video sequences. The 4D Video Textures approach computed optical flow between two rendered images obtained by texturing a model using the appearance information from two 4D video frames. This allowed real-time parametric control of both the geometry and appearance of 4D video characters. As with Floating Textures, this method is limited by its computational complexity requiring optical flow calculations to be computed within the render cycle. Subsequent research introduced 4D model flow [21], which precomputed optical flow alignment between pairs of models removing the need for online computation of optical flow, but required additional floating point textures to be stored and loaded for rendering.

Aganj et al. [1] proposed to apply a non-linear warping to the captured images in order to make the images consistent with the mesh. The approach used sparse features and thin slice splines to generate a dense correspondence field to warp the captured images. In contrast, Takai et al. [101] presented Harmonised Texture Maps which adjusted the geometry against the camera images.

The work presented in this thesis takes inspiration from Floating Textures by computing the optical flow using the projectively textured models. However, as the rendering viewpoint is not determined, optical flow correspondence can not be computed in the rendered screen domain. Instead it is computed in the domain of each captured camera with respect to all other cameras. These multiple optical flow fields are then used to build a correspondence field between all the capture cameras as opposed to only the subset used for rendering.

4.3 Optimal Representation of 4D Video

In this section, an ideal set of properties for the Optimal Representation of 4D Video are outlined. The idea is driven by the need to store and use 4D video in a more efficient way that maintains the original spatial and temporal changes of the captured video. Within a 4D
video frame, there is large amount of data that is not used as well as huge amounts of spatial and temporal redundancy which must be exploited in order to minimise storage requirements. The ideal properties of a 4D video texture representation are as follows:

- Minimal loss of information due to image resampling from the captured multiple view video.

- Efficient representation of information across multiple views to minimise data size for storage, transmission and rendering.

- Spatial coherence of view sampling in the texture domain such that adjacent texels are sampled from the same view to minimise switches in viewpoint.

- Temporally coherent representation such that the texel corresponding to the same surface point is sampled from the same camera over time to minimise switches in camera.

- Ordered sampling of multiple view observations of the same surface point according to visibility to allow efficient and scalable rendering.

- Texture layer alignment such that multiple observations of the same surface point at a time-instant have the same texel location.

Several of these properties are illustrated in Figure 4.2, e.g. a texture element (red) representing a point on the true surface, remains in the same place between texture layers and over time.

![Figure 4.2: Ideal texture representation properties. View-dependent appearance is split between multiple texture maps. Texture elements remain in the same position within texture layers of the same frame and in the same positions over time.](image-url)
These properties motivate the work in this chapter. By using a temporally consistent mesh representation with a single set of UV coordinates, texture elements remain in the same position over time. Spatio-temporal optimisation of camera labels ensure spatial and temporal coherence between texels, Section 4.4. Finally, multiple camera alignment ensures that UV location between layers are sampled from the same true surface point despite errors in geometry and calibration, Section 4.5

4.4 Optimisation of Camera Assignment

The naive approach of assigning camera images to mesh polygons solely based upon the polygon normal, as presented in Chapter 3, leads to a jagged assignment of cameras to polygons over the mesh surface and high frequency flickering over time. Therefore, it is desirable to maximise the size of camera assignment patches in each frame while minimising the changes in each polygon over time. These properties will allow video compression algorithms to further reduce the storage requirements of the MLTR through increased spatial and temporal redundancy.

Formally, this optimisation can be approached as a labelling problem mapping camera images to mesh polygons. An undirected graph, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, is constructed consisting of vertices $\mathcal{V}$ and edges $\mathcal{E}$. $\mathcal{V}$ are used to represent the set of mesh polygons which are expressed as $p^t_i \in \mathcal{V}$. Graph edges $\mathcal{E}$ form a local one-neighbour connectivity between mesh polygons with a common edge, as shown in Figure 4.3, and are expressed as pairs of vertices in the set of edges $\{p^t_i, p^t_j\} \in \mathcal{E}$. Camera images are represented by labels $\{c_i\}^{N_C}_{i=1} = \{c_1, c_2, \ldots, c_{N_C}\}$. $\mathcal{G}$ is constructed for each frame in a 4D video sequence and a temporal edge is added between corresponding polygon nodes over time. This results in a three-dimensional graph structure in space and time.

The labelling $\mathcal{L}$ can formally be defined as shown in Equation 4.8.

$$\mathcal{L} = \{\{p^t_i : c^t_i\}^{N_P}_{i=1}\}^{N_F}_{t=1}$$  \hspace{1cm} (4.8)

where $\mathcal{L}$ represents the labelling, $p^t_i$ and $c^t_i$ are the $i^{th}$ polygon/camera pair at frame $t$, $N_P$ is the number of polygons in the mesh and $N_F$ is the total number of frames.
4.4 Optimisation of Camera Assignment

An optimal labelling is found by minimising Equation 4.9 using the \(\alpha\)-expansion algorithm described in Section 4.2.1.

\[
e(\mathcal{L}) = e_D(\mathcal{L}) + \lambda_S e_S(\mathcal{L}) + \lambda_T e_T(\mathcal{L})
\]  

(4.9)

where \(e(\mathcal{L})\) is the total energy for a given labelling \(\mathcal{L}\), \(e_D(\mathcal{L})\) is the data term, \(e_S(\mathcal{L})\) is the spatial smoothness term and \(e_T(\mathcal{L})\) is the temporal smoothness term. \(\lambda_S\) and \(\lambda_T\) are weightings for the spatial and temporal smoothness terms, respectively. In this thesis, weights are kept equal, \(\lambda_S = \lambda_T = 1\). This assumes an equal importance to both spatial and temporal information, but could be used to prioritise either spatial or temporal consistency.

![Graph Vertex, \(p_t^i \in V\)  
Graph Edge, \(\{p_t^i, p_t^j\} \in E\)  
Mesh Vertex  
Mesh Edge  
Mesh Polygon](image)

Fig. 4.3 Mesh structure represented as an undirected graph.

The data term \(e_D(\mathcal{L})\) penalises deviation from the camera which has the best visibility for each polygon independently. The most direct camera to each polygon is based on the angle between the polygon normal and normalised camera direction vector. This assignment is traded off against the smoothness terms. Visibility constraints are enforced with the introduction of a null camera label, \(c_N\). The null camera label is given a score such that it is significantly higher than the score any visible camera/polygon pair can achieve but significantly lower than a camera considered to be not visible. This means that \(c_N\) is always selected when no cameras are considered to be visible and ensures visibility constraints are respected. These conditions are shown in Equations 4.10 and 4.11.

\[
e_D(\mathcal{L}) = \sum_{p_t^i \in \mathcal{V}} e_d(p_t^i, c_i)
\]  

(4.10)
Optimal Representation of 4D Video

\[
e_d(p_t^i, c_i) = \begin{cases} 
1 - c_D(c_i) \cdot p_n(p_t^i) & \text{if } (c_i \neq c_N) \text{ and } p_t^i \text{ is visible in } i^{th} \text{ camera} \\
1000 & \text{if } (c_i = c_N) \\
\infty & \text{otherwise}
\end{cases} \tag{4.11}
\]

where \(e_d(p_t, c_i)\) is the energy of assigning camera \(c_i\) to polygon \(p_t\), \(p_n(p_t)\) is the polygon normal associated to polygon \(p_t\), and \(c_D(c_i)\) is the normalised direction vector associated with camera \(c_i\).

The spatial smoothness term encourages pairs of polygons to have the same camera assignment over the mesh surface. This helps to increase spatial redundancy by reducing sharp changes in the layered texture maps caused by changes in camera assignment. Spatial smoothness is considered for all possible pairs of polygons that share a common edge in the mesh domain. The conditions for spatial smoothness are shown in Equation 4.12 and 4.13.

\[
e_s(L) = \sum_{\{p_t^i, p_j^j\} \in \mathcal{E}} e_s(p_t^i, c_i, p_j^j, c_j) \tag{4.12}
\]

\[
e_s(p_t^i, c_i, p_j^j, c_j) = \begin{cases} 
0 & \text{if } c_i = c_j \\
\cos^{-1}(c_D(c_i) \cdot c_D(c_j))/\pi & \text{if } (c_i \neq c_N) \text{ and } (c_j \neq c_N) \\
1 & \text{otherwise}
\end{cases} \tag{4.13}
\]

where \(e_s(p_t^i, c_i, p_j^j, c_j)\) is the spatial smoothness term assigning polygon \(p_t^i\) to camera label \(c_i\) while also assigning polygon \(p_j^j\) to camera label \(c_j\), and \(c_D(c_i)\) gives the camera direction vector for a given camera label.

If camera labels are the same, the score is zero which encourages the same camera labelling between pairs of polygons. This is an important property to ensure the energy function is regular and can be minimised using \(\alpha\)-expansion. Proof of regularity is shown in Section 4.4.1. If both labels are different and are not equal to \(c_N\), the associated cost is proportional to the angle between the two cameras measured by the direction vectors \(C_D(c_i)\) and \(C_D(c_j)\). This value is normalised against the maximum value, \(\pi\), meaning the final cost will range from zero to one. Finally, if either camera label equals \(c_N\) or is considered to not be visible, a score of one is assigned.
4.4 Optimisation of Camera Assignment

The temporal term is used to minimise changes in a polygon assignment over time as shown in Equation 4.14 and 4.15. This uses the Potts model which assigns a cost of zero if the camera labels are the same and a cost of one if labels are different.

\[
e_T(\mathcal{L}) = \sum_{\{p'_i, p_i^{t+1}\} \in \mathcal{E}} e_t(p'_i, c_i, p_i^{t+1}, c_j) \tag{4.14}
\]

\[
e_t(p'_i, c_i, p_i^{t+1}, c_j) = \begin{cases} 
0 & \text{if } c_i = c_j, \\
1 & \text{otherwise} 
\end{cases} \tag{4.15}
\]

where \(e_t(p'_i, c_i, p_i^{t+1}, c_j)\) is the temporal cost of assigning camera label \(c_i\) to polygon \(p'_i\) while also assigning camera label \(c_j\) to polygon \(p_i^{t+1}\). This is only considered for the same polygon at different time instants. The energy is zero when camera labels are the same and one if camera labels are different. This encourages a polygon to have the same labelling over time.

In this thesis, two optimisation schemes are tested against the method presented in Chapter 3, these schemes are:

- **No Optimisation (NO)**
  NO is the method presented in Chapter 3 in which cameras are assigned to polygons based solely on the polygon normal. These results can also be achieved using the MRF framework presented in this chapter by considering only the data term \(e_D()\) in the energy formula and by not adding spatial or temporal edges to the graph.

- **Spatial Optimisation (SO)**
  This result is achieved by constructing the graph for every frame (with no temporal edges) and removing the temporal term \(e_T()\) in the energy function.

- **Spatial/Temporal Optimisation (STO)**
  STO uses both spatial and temporal terms to find a labelling that minimises the energy function over all frames simultaneously.

The optimisation is solved once per layer sequence with visibility constraints updated between each solve to exclude the previous assignment from future results. This ensures that no appearance data is duplicated between layers at each time instance.
4.4.1 Proof of Regularity

To ensure that the energy function can be minimised by the \(\alpha\)-expansion algorithm, it must be shown to satisfy the regularity condition [63]. In order to do this, every pairwise term must be shown to be a metric as defined by the conditions listed in Section 4.2.1. For convenience, these conditions are restated below using appropriate variables:

\[
e_{s}(c_\alpha, c_\beta) = 0 \iff c_\alpha = c_\beta \tag{4.16}
\]

\[
e_{s}(c_\alpha, c_\beta) = e_{s}(c_\beta, c_\alpha) \geq 0 \tag{4.17}
\]

\[
e_{s}(c_\alpha, c_\beta) \leq e_{s}(c_\alpha, c_\gamma) + e_{s}(c_\gamma, c_\beta) \tag{4.18}
\]

First, it is shown that the spatial smoothness term \(e_{s}()\) in Equation 4.13 is a metric. It can be seen that Equations 4.16 and 4.17 hold true due to the \(c_i = c_j\) condition. By definition, the scalar product between two normalised 3D vectors fall within the range \([-1, 1]\) with the inverse cosine of these values in the range \([0, \pi]\). This result is then normalised with respect to \(\pi\). It therefore only remains to prove that the triangular inequality condition, Equation 4.18, holds true in all possible cases of \(e_{s}()\).

Fig. 4.4 Proof of regularity: Triangular inequality conditions for non-null camera labels
4.4 Optimisation of Camera Assignment

The six potential cases that can arise are evaluated individually, while the five non-null scenarios are illustrated in Figure 4.4.

1. \( c_\alpha = c_\beta = c_\gamma \)
   In this case, Equation 4.4 assigns a cost of zero to all components and the triangular inequality condition is satisfied. This is illustrated in Figure 4.4 shown in red.

2. \( c_\alpha \neq c_\beta \neq c_\gamma \) and \( c_\alpha \neq c_N \) and \( c_\beta \neq c_N \) and \( c_\gamma \neq c_N \)
   In this case, all three labels are different and \( \neq c_N \). It can be seen that any combination where all three labels are different cannot break the triangular inequality condition. This is illustrated in blue shown in Figure 4.4.

3. \( c_\alpha = c_\beta \) and \( c_\alpha \neq c_\gamma \) and \( c_\beta \neq c_\gamma \) and \( c_\alpha \neq c_N \) and \( c_\beta \neq c_N \) and \( c_\gamma \neq c_N \)
   In this case, Equation 4.18 evaluates to \( e_s(c_\alpha, c_\beta) \leq e_s(c_\alpha, c_\gamma) + e_s(c_\gamma, c_\alpha) \) substituting \( c_\alpha \) for \( c_\beta \). Given Equation 4.4 and 4.5, it follows that \( 0 \leq 2e_s(c_\alpha, c_\gamma) \), with \( e_s(c_\alpha, c_\gamma) \) between \([0, 1]\). Therefore Equation 4.18 is satisfied. This is illustrated in green shown in Figure 4.4.

4. \( c_\beta = c_\gamma \) and \( c_\beta \neq c_\alpha \) and \( c_\gamma \neq c_\alpha \) and \( c_\alpha \neq c_N \) and \( c_\beta \neq c_N \) and \( c_\gamma \neq c_N \)
   In this case, the triangular inequality equation is equivalent to \( e_s(c_\alpha, c_\beta) \leq e_s(c_\alpha, c_\beta) + e_s(c_\beta, c_\beta) \) with \( c_\beta \) substituted for \( c_\gamma \). \( e_s(c_\beta, c_\beta) = 0 \), so \( e_s(c_\alpha, c_\beta) \leq e_s(c_\alpha, c_\beta) \) remains. Equation 4.18 is therefore satisfied. This is illustrated in green shown in Figure 4.4.

5. \( c_\alpha = c_\gamma \) and \( c_\alpha \neq c_\beta \) and \( c_\gamma \neq c_\beta \) and \( c_\alpha \neq c_N \) and \( c_\beta \neq c_N \) and \( c_\gamma \neq c_N \)
   In this case, the triangular inequality equation is equivalent to \( e_s(c_\alpha, c_\beta) \leq e_s(c_\alpha, c_\alpha) + e_s(c_\alpha, c_\alpha) \) with \( c_\alpha \) substituted for \( c_\gamma \). \( e_s(c_\alpha, c_\alpha) = 0 \), so \( e_s(c_\alpha, c_\beta) \leq e_s(c_\alpha, c_\beta) \) remains. Equation 4.18 is therefore satisfied. This is illustrated in Figure 4.4 shown in magenta.

6. \( c_\alpha = c_N \) or \( c_\beta = c_N \) or \( c_\gamma = c_N \)
   In the case that \( c_\alpha \) is assigned the null camera label, Equation 4.18 reduces to \( 1000 \leq 1000 + e_s(c_\beta, c_\gamma) \) which holds true as \( e_s(c_\beta, c_\gamma) \geq 0 \). This is also the case when \( c_\beta = c_N \).
   In the case that \( c_\gamma \) is assigned the null camera label, Equation 4.18 reduces to \( e_s(c_\alpha, c_\beta) \leq 2000 \). As \( e_s(c_\alpha, c_\beta) \) will always fall into the range of \([0, 1]\) for non-null camera labels Equation 4.18 is satisfied.

All possible cases that can arise for \( e_s() \) have been shown to satisfy the triangular inequality condition and therefore \( e_s() \) has been shown to be a metric. The temporal term \( e_t() \) in Equation 4.15 uses the Potts models with respect to camera labels. This function is well known to be a metric. This completes the proof.
4.5 Multiple Camera Texture Alignment

In view dependent rendering, simple projection and blending of camera views using approximate mesh geometry and erroneous camera calibration leads to blurring and ghosting artefacts. These artefacts are caused by misalignment between overlapping camera images projected onto the mesh surface. In order to minimise these artefacts, as with previous work presented in Section 4.2.4, optical flow-based image warping is used to correct misalignments before sampling into the texture domain. This requires alignment of the texture data within the overlapping regions of the camera views. This process ensures that pixels representing the same point on the true surface are aligned in the MLTR domain. This is one of the ideal properties as defined in Section 4.3.

Wide camera baselines used in the capture setup means optical flow cannot be directly applied to the camera images. To establish optical flow between camera views, the geometry is rendered from the viewpoint of camera $c_i$ and projectively textured using the image of camera $c_j$ for all $N_C$ cameras. This results in $N_C^2$ rendered images, $R(i, j)$, which denotes the image rendered from the $i^{th}$ camera viewpoint using the $j^{th}$ camera image for texturing. A complete example frame is shown in Figure 4.5. An optical flow correspondence field, $O(i, j)$, is computed between the reference rendered image $R(i, i)$ and $R(i, j)$ where $i \neq j$ for all cameras, as shown in Figure 4.6. Optical flow is known to give unreliable flow vectors in the presence of occlusions where it is undefined. To mitigate such errors, a binary confidence score is assigned to each flow vector based on both depth discontinuities (taken from a rendered depth map of the geometry) and occlusions (computed by identifying vertices visible in $c_i$ but occluded in $c_j$). Figure 4.6 indicates such artefacts in black, resulting in zero confidence regions within the resulting field of scores $B(i, j)$.

A correction vector is applied to the projected point in the camera domain to take into account the projection error. The magnitude of the correction vector is given by the weighted average of all visible and high confidence flow vectors on the surface:

$$O(i) = \sum_{j=1}^{N_C} \omega(j)B(i, j)O(i, j).$$

(4.19)

where $O(i)$ is the field of correction vectors for the $i^{th}$ camera, $\omega(j)$ is a scalar weight such that $\sum_{j=1}^{N_C} \omega_j = 1$. Multiplication of fields occur on a per-pixel basis. Results in this thesis use uniform weighting, however this could be varied to prioritise particular cameras.
Fig. 4.5 Multiple camera projective texture example. $R(i, j)$ in which $i$ is rows and $j$ is columns. Images highlighted with a red box are used as the reference image when computing optical flow.
Fig. 4.6 Multiple camera optical flow. Optical flow field $O(i, j)$ computed between rendered images $R(i, i)$ and $R(i, j)$. Reference cameras highlighted in red boxes produce zero flow when computing optical flow with themselves. Flow vectors are represented using the HSV colour space found in Figure 4.1a with a maximum length of 5 pixels, with black pixels representing unreliable flow vectors due to occlusions.
4.6 Results and Evaluation

In this section, results and an evaluation of the MLTR optimisation schemes and optical flow-based texture correction are presented. The evaluation consists of a comparison of processing times, rendering quality and storage requirements versus the free viewpoint video renderer (FVVR) presented by Stark et al. [95], as performed in Chapter 3.

All tests were performed on a Dell Optiplex 9010 desktop computer running Ubuntu 12.04 with an Intel i7 processor, 8GB of RAM and a Nvidia GeForce FX 640 graphics card. The optimisation algorithm was implemented using the Middlebury MRF 2.2 library [13, 14, 63]. The extraction algorithm was implemented using C++11, OpenCV 2.4.11 computer vision library, OpenGL 4.3 and OpenGL Shading Language 1.5. The rendering algorithm was implemented using C++11, OpenGL Shading Language 1.5 and OpenSceneGraph 3.2.1 graphics library.

4.6.1 Error Metric

In order to evaluate this approach, the storage and image quality error metrics presented in Section 3.4.1 are used in addition to camera assignment flicker defined below.

**Temporal Camera Assignment Flicker**

Temporal flicker is defined as the number of polygon which change assignment between consecutive layered texture maps. The average percentage of polygons which change assignment in consecutive frames over a sequence is given in Equation 4.20.

\[
f(\mathcal{L}) = \frac{100}{N_p(N_T - 1)} \sum_{t=1}^{N_T-1} \sum_{i=1}^{N_p} f_p(c_t^i, c_{t+1}^i)
\]  

(4.20)

where \( f(\mathcal{L}) \) is the percentage of polygons which change assignment between consecutive frames for a given labelling, \( N_p \) is the number of polygons in the mesh, \( N_T \) is the number of frames in the sequence and \( f_p(c_t^i, c_{t+1}^i) \) compares the assignment between the \( p \)th polygon at frame \( t \) and \( t + 1 \) and returns 0 if they are the same and 1 if they are different, as shown in Equation 4.21.

\[
f_p(c_t^i, c_{t+1}^i) = \begin{cases} 
0 & \text{if } c_t^i = c_{t+1}^i, \\
1 & \text{otherwise}
\end{cases}
\]  

(4.21)
4.6.2 Optimisation Results

Table 4.1 presents processing times for each of the optimisation schemes on the evaluation datasets, as well as including the number of frames, cameras and mesh complexity. The NO method is the fastest to compute as it is essentially a sort function comparing the angle between polygon normal and all visible capture cameras. The SO method takes longer as this has to take into account the assignment of three local neighbours of every polygon. The STO method takes the longest to compute as this is taking into account three spatial neighbours as well as up to two temporal neighbours. Table 4.1 also includes results of processing time from the MIT Samba dataset [108]. These are included to demonstrate that the algorithm scales well with sequence length and mesh structure and converges within a reasonable amount of processing time.

Table 4.1 Summary of processing times using MRF framework

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Frames</th>
<th>Cameras</th>
<th>Polygons</th>
<th>NO (s)</th>
<th>SO (s)</th>
<th>STO (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloth</td>
<td>322</td>
<td>5</td>
<td>768</td>
<td>16</td>
<td>62</td>
<td>77</td>
</tr>
<tr>
<td>Dan</td>
<td>28</td>
<td>8</td>
<td>5330</td>
<td>20</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>Face</td>
<td>355</td>
<td>5</td>
<td>5248</td>
<td>134</td>
<td>255</td>
<td>340</td>
</tr>
<tr>
<td>Roxanne</td>
<td>31</td>
<td>8</td>
<td>4950</td>
<td>20</td>
<td>23</td>
<td>34</td>
</tr>
<tr>
<td>Samba</td>
<td>175</td>
<td>8</td>
<td>19938</td>
<td>475</td>
<td>494</td>
<td>902</td>
</tr>
</tbody>
</table>

Figure 4.7, 4.8, 4.9, 4.10 and 4.11 show results of the optimisation schemes for Dan, Roxanne, Samba, Cloth and Face datasets, respectively. The benefit of optimising camera labelling can clearly be seen. NO simply assigns the most direct camera to each polygon resulting in many changes over the mesh surface at each time instance as well as over time. SO creates large areas of the same assignment which is good for increasing spatial redundancy. However, without knowledge of past and future frames, large areas change assignment from frame-to-frame. STO attempts to maximise the size of consistent camera assignment patches while minimise changes in assignment over time. This results in a trade off between spatial and temporal smoothness.

In order to quantify the stability of camera assignment in the three presented methods, temporal flicker is assessed as presented in Equation 4.20. This measures the percentage of polygons that change in consecutive frames. Table 4.2 presents the temporal flicker of the evaluated datasets when processed using the proposed optimisation schemes. These results demonstrate that the STO method produces the least amount of change in polygon assignment over a sequence.
Table 4.2 Average percentage of polygons which change camera assignment between consecutive frames.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Polys</th>
<th>Opt</th>
<th>Average % of polygon changes per layer per frame(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Cloth</td>
<td>768</td>
<td>NO</td>
<td>8.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>8.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>0.45</td>
</tr>
<tr>
<td>Dan</td>
<td>5330</td>
<td>NO</td>
<td>9.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>16.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>2.49</td>
</tr>
<tr>
<td>Face</td>
<td>5248</td>
<td>NO</td>
<td>1.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>0.00</td>
</tr>
<tr>
<td>Roxanne</td>
<td>4950</td>
<td>NO</td>
<td>11.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>16.44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>2.71</td>
</tr>
<tr>
<td>Samba</td>
<td>19938</td>
<td>NO</td>
<td>7.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>11.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>4.14</td>
</tr>
</tbody>
</table>

Fig. 4.7 Optimisation results for Dan dataset. NO (top row), SO (middle row) and STO (bottom row) for frames 1 to 5 of walk sequence of layer 1.
Fig. 4.8 Optimisation results for Roxanne dataset. NO (top row), SO (middle row) and STO (bottom row) for frames 1 to 5 of walk sequence of layer 1.

Fig. 4.9 Optimisation results for Samba dataset. NO (top row), SO (middle row) and STO (bottom row) for frames 1 to 5 of layer 1.
Fig. 4.10  Optimisation results for cloth dataset. NO (top row), SO (middle row) and STO (bottom row) for frames 0, 5, 10, 15 and 20 of layer 1.

Fig. 4.11  Optimisation results for face dataset. NO (top row), SO (middle row) and STO (bottom row) for frames 1 to 5 of layer 1.
4.6.3 Rendering Quality versus Optimisation

The rendering quality between the NO, SO and STO optimisation schemes are compared to the output of the FVVR [95] using the SSIM measure [111], defined in Section 3.4.1. This was performed for all evaluation datasets.

Figure 4.12 shows the effect of the optimisation schemes on rendering quality when changing the number of layers used for rendering, \( n_L \).

![Graph (a)](image)

![Graph (b)](image)

![Graph (c)](image)

![Graph (d)](image)

Fig. 4.12 The effect of optimisation on rendering quality evaluated for (a) Dan, (b) Roxanne, (c) Cloth and (d) Face datasets. All results use a layer size of 1024x1024 pixels.

It can be seen that the optimisation scheme does not affect the rendering quality on a per frame basis. This can be expected particularly when using \( n_L > 1 \) as the same data exists in all three optimisation schemes but has simply been reordered. Small differences, often negligible, in rendering quality between the optimisation schemes occur when using \( \leq 2 \) layers. Yet as \( n_L \) is increased, the difference becomes insignificant.
4.6.4 Storage

Storage requirements are compared by taking the total storage size of the MLTR when compressed into a video format along with the greyscale camera assignment images against the original video when compressed in the same video stream format, as described in Section 3.4.1. Table 4.3 presents the storage results for each of the evaluated datasets for different MLTR sizes, number of layers $n_L$ and optimisation schemes developed in this chapter. Figure 4.13 shows the storage reduction given by the MLTR for the evaluated datasets using the different optimisation schemes.

![Graphs](image)

Fig. 4.13 The effect of optimisation on MLTR storage size evaluated for (a) Dan, (b) Roxanne, (c) Cloth and (d) Face datasets. All results use a layer size of 1024x1024 pixels.

The results show that the STO method is able to achieve the highest storage reduction of all the presented optimisation schemes. This is due to the video compression algorithm taking advantage of the increased spatial and temporal redundancy in the layer sequences due to the optimised camera labelling. In the case of the Dan (Figure 4.13a) and Roxanne (Figure 4.13b) datasets, the STO method is able to give an additional 5% storage reduction compared to the NO method.
Table 4.3 Storage requirements of MLTR with the different optimisation schemes applied

<table>
<thead>
<tr>
<th>Video (MB)</th>
<th>MLTR Size</th>
<th>Opt</th>
<th>Storage for varying $n_L$ (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Cloth</td>
<td>906</td>
<td>NO</td>
<td>14.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>14.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>13.7</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>NO</td>
<td>40.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>39.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>37.6</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>NO</td>
<td>106.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>104.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>100.4</td>
</tr>
<tr>
<td>Dan</td>
<td>57</td>
<td>NO</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>NO</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>NO</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>7.3</td>
</tr>
<tr>
<td>Face</td>
<td>386</td>
<td>NO</td>
<td>9.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>NO</td>
<td>26.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>24.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>24.1</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>NO</td>
<td>83.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>79.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>78.7</td>
</tr>
<tr>
<td>Roxanne</td>
<td>61</td>
<td>NO</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>NO</td>
<td>3.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>NO</td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SO</td>
<td>7.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STO</td>
<td>7.7</td>
</tr>
</tbody>
</table>
4.6 Results and Evaluation

In the case of the Cloth (Figure 4.13c) and Face (Figure 4.13d) datasets, which are significantly longer, an additional 5% storage reduction was achieved using the STO method compared to the NO. This gives a real saving in the file size of 20MB and 45MB in the Face and Cloth datasets, respectively, from effectively reorganising the appearance data.

4.6.5 Multiple Camera Optical Flow-based Texture Alignment

This section presents the results from the multiple camera optical flow alignment method presented in Section 4.5. This was developed to reduce blur and ghosting artefacts. A comparison between results is shown using a heat map visualisation. Heatmaps are a convenient way to represent the difference between two datasets. In this section, heat maps are used to show the difference between the RGB colours of corresponding pixels. This difference in colour is normalised against the maximum possible difference of the RGB colour space and mapped to the jet colour space. Figure 4.14 shows the first layer from a frame of the character Dan walk sequence. The multiple camera optical flow alignment has improved the alignment of features at the boundary between cameras. This can clearly be seen in the hair line (red box), face region where the eyebrows are in alignment (green box), and also in the striped pattern on the character’s jumper (blue box).

![Fig. 4.14](image)

Fig. 4.14 Multiple camera optical flow alignment result: Dan MLTR layer 1. (left) No correction, (right) correction applied and (centre) close up of selected patches with a heat map to highlight differences in RGB colour values.

Figure 4.15 show results of an extracted texture layer from the Face dataset. It can clearly be seen that the texture alignment between cameras has been improved, most noticeably in and around the iris where the geometry contains errors. This error is corrected in the aligned version and the boundary between the two cameras is now difficult to see.
Fig. 4.15 Multiple camera optical flow alignment Face dataset result: Face MLTR layer 1. (left) No correction, (right) correction applied and (centre) close up of selected patches with a heat map to highlight differences in RGB colour values.

Figure 4.16 shows a view dependent rendering of the Cloth dataset. Three areas have been highlighted to show the blur and ghosting artefacts in the FVVR rendering and the reduction using the multiple camera alignment algorithm combined with the MLTR. This shows that the texture details have become clearer such as the white pattern displayed at the top (red box) and bottom (magenta box). Ghosting artefacts, shown in the middle (green box), are reduced by bringing the camera views into better alignment.

Fig. 4.16 Multiple camera optical flow alignment Cloth dataset result rendered using MLTR. (left) No correction, (right) correction applied and (centre) close up of selected patches with a heat map to highlight differences in RGB colour values.
4.7 Conclusions

A MRF based optimisation has been introduced which optimises the camera assignment in the MLTR over the mesh surface and over time. This optimisation was developed to increase the spatial and temporal redundancies in layer sequences to allow video compression algorithms to make further storage reductions. Proof of regularity for the energy function has also been demonstrated. This is important as it provides proof that the function is suitable to be minimised by the $\alpha$-expansion algorithm. This optimisation was able to help achieve a further reduction in the required storage.

A multiple camera optical flow based alignment method was also introduced. This alignment method was able to effectively reduce texture artefacts caused by approximate geometry and camera calibration errors. Currently, optical flow algorithm fail when applied to wide baseline camera setups. However, a dense correspondence was computed between all possible camera pairs by projectively texturing the geometry from the viewpoint of a pose camera using the texture from an alternative camera. This brings the appearance of the character into a common frame of reference and allows the computation of accurate optical flow correspondence. Blur and ghosting artefacts were reduced which enhanced the visual appearance of the final rendered models and ensured that appearance details between layers were aligned.

Finally, this chapter defined the ideal properties of the optimal representation of 4D video that a multiple camera texture representation should have. The work in this chapter has contributed the optimal representation of 4D video by optimising camera labelling and improving the spatial alignment between texture layers using an optical flow based alignment method.
Chapter 5

Spatio-Temporal Shape Optimisation

5.1 Introduction

View dependent rendering was developed to allow photo-realistic rendering of static scenes using approximate geometric models [30]. By adapting the appearance of the rendered model based on the virtual camera viewpoint, small errors in the geometric proxy and camera calibration are concealed. However, in order to facilitate high quality view dependent rendering and avoid ghosting and blurring artefacts, important geometric features must be preserved. Shape reconstructions from dense, narrow baseline capture setups have been able to achieve millimetre scale accuracy [32, 42]. However, shape reconstruction from sparse, wide baseline camera setups is a more challenging problem. Methods to resolve these artefacts have been proposed but require offline computation and storage of correspondence [21] or online computation of optical flow [23, 35]. However, these methods do not address the underlying problem of errors in the geometry.

This chapter addresses the problem of refining a given 3D shape to reintroduce details lost in a model free reconstruction pipeline. This chapter presents two approaches to shape optimisation. The goal in both approaches is to adapt a temporally consistent mesh to better reflect the true surface captured in the camera images. The first method approaches this as a surface registration problem using a non-rigid iterative closest point (ICP) algorithm [7] to deform the temporally consistent mesh (TC). In this approach, the strengths of two shape reconstruction techniques are leveraged to re-introduce details lost in the processing pipeline: visual hull (VH) reconstruction and multiple view stereo (MVS). The second approach establishes surface correspondence using optical flow between wide baseline cameras. These
refined correspondences are then used in a bundle adjustment framework to minimise the reprojection error of mesh vertices. This is achieved by optimising both camera calibration and surface geometry.

## 5.2 Background

This section reviews shape refinement methods that start with a coarse reconstruction and refine the shape to better reflect the characteristics of the scene. Miller et al. [75] introduced a global shape refinement technique which refined a depth map for each camera in projective ray space using a min-cut/max-flow formulation. This approach used rims, represented as pixel chains, as hard constraints in the depth maps. Rims are points on the 3D surface where the camera rays graze the surface tangentially. These are considered as points on the VH that intersect the true surface. This method resulted in a RGB+D shape representation per camera and allowed view dependent rendering of the refined depth maps.

More recently, Nobuhara et al. [77] presented a real-time view dependent shape optimisation technique that optimised the virtual camera depth map to maximise photo-consistency. The key idea behind this method is that local refinement of geometry leads to a more photo-consistent result as opposed to creating a single global geometric proxy. This work achieved real-time frame rates using a GPU based implementation of loopy belief propagation to optimise a discrete set of depth offsets applied to the virtual camera depth map. The main disadvantages of this method are the computational complexity and that the result of the current frame is heavily influenced by the result of the previous viewpoint. While this allows smooth transitions between viewpoints, it means that the same viewpoint can have two different results depending upon the path it took to reach that viewpoint. Both of these approaches optimise and render using a depth map representation of the subject. This representation is only suitable for replay of captured data and would not be appropriate for current mesh based animation techniques.

Shading cues have also been used to refine multiple view geometry [119, 120] as well as depth maps from RGB+D sensors [121, 123]. Wu et al. [119] were able to add true surface details by resolving shading cues. The multiple view surface was segmented into areas of similar albedo and estimations of incident illumination were made. The method presented in [120] required the subject to be captured from > 20 viewpoints and therefore cannot be applied in this instance. The approach described in [119] requires a finely tessellated mesh
structure containing approximately 80k vertices which would make real time application of mesh based animation algorithms restrictive due to mesh complexity. Both of these methods make the assumption of uniform albedo which limits the application of these methods to subjects that are not richly textured.

Photometric bundle adjustment [32] refined an initial coarse reconstruction of a subject along with the camera calibration parameters. Methods based on shape evolution [33, 34, 44] have also been proposed that minimise the error between the estimated scene and captured images in a Bayesian framework. Both of these methods are based on complex optimisations, computationally expensive and assume a higher number of cameras viewing the subject.

5.3 Approach 1: Non-Rigid ICP Shape Optimisation

5.3.1 Overview

Reconstruction pipelines often combine multiple shape reconstruction methods and use the strengths of one method to overcome the limitations of another [26, 77, 88, 94]. This is also the case with the 4D reconstruction pipeline developed at the University of Surrey. The VH provides an initial reconstruction that is refined through MVS. Given a set of unstructured MVS meshes, a TC mesh can then be computed.

VH reconstruction contains surface points which intersect with the real surface, commonly referred to as rims. However, VH reconstructions are coarse, do not reconstruct concavities, and contain erroneous volumes in the presence of self occlusion and uneven camera distribution. MVS produces geometry that is as photo-consistent as allowed by the camera calibration. However, errors in the camera calibration cause geometric error and MVS fails when no photometric features can be found, e.g. textureless areas. MVS can also result in reconstructions that are smaller than the true shape if a strong regularisation term is used. TC geometry often lacks high frequency surface detail found in the MVS reconstruction. One reason this occurs is that the algorithms often aim to represent the mesh sequence at a lower resolution than the raw reconstructions and error between pairwise alignment naturally contain some degree of error. However, temporal consistency is an important property as it enables the reuse of 4D video to produce novel animations [12, 22, 23].

Figure 5.1 shows a profile view of the geometric proxy of a frame in the Dan dataset obtained using VH, MVS and TC. It can be seen that the nose of the character is smoothed out leading
to an unconvincing rendering result from profile views. This is a typical scenario in model free approaches and occurs due to limited capture resolution. This could be improved by increasing the number of cameras, camera resolution and/or reducing capture volume size. However, this may not be a practical solution given available equipment, cost and space requirements, or the use of previously captured data.

5.3.2 Problem Formulation

The first method proposed in this chapter for shape refinement approaches the task as a registration problem. The method takes in mesh sequence reconstructions of: VH [65] converted into a mesh surface, MVS [97] and TC [17], denoted as \( \{ M_V(t) \}_{t=1}^{N_T}, \{ M_S(t) \}_{t=1}^{N_T} \) and \( \{ M_T(t) \}_{t=1}^{N_T} \), respectively, where sequences are \( N_T \) frames in length and \( t \) is a time instance. The method uses a non-rigid ICP algorithm [7] to deform the TC to match important features from the VH and MVS at each time instance. By approaching the problem in this way, it gives access to the true surface points of the VH, photo-consistency of the MVS and the temporal consistency of the TC. An overview of the approach is shown in Figure 5.2.

Firstly, \( M_T(t) \) is sub-divided to increase the mesh resolution giving \( M_T^H(t) \). This is to allow sufficient geometric detail to be represented. The aim is to sub-divide \( M_T(t) \) to have approximately 10k vertices evenly distributed over the surface. Approximately 10k vertices have been used in previous work [26] to represent fine details of the human body. Furthermore, 10k vertices does not add significant overheads and complexity to existing
5.3 Approach 1: Non-Rigid ICP Shape Optimisation

mesh based animation techniques [22, 23]. A mid-point sub-division scheme is used [25] to increase mesh resolution while maintaining temporal consistency. Mid-point sub-division introduces a new vertex at the middle point of every edge and splits every polygon into four, shown in Figure 5.3.

![Diagram of non-rigid ICP shape refinement approach](image)

**Fig. 5.2** Overview of non-rigid ICP shape refinement approach. Input/output shown in blue and processes are shown in gold.

![Images of temporally consistent mesh](image)

**Fig. 5.3** Temporally consistent mesh with increased resolution using mid-point sub-division.
Secondly, rims are extracted from the VH, Section 5.3.3, representing true surface points. Each vertex of the VH is then assigned a confidence score, detailed in Section 5.3.4. A correspondence search is undertaken to find two corresponding vertices for each vertex in the $M_H^A(t)$, one from VH and one from MVS. Based on the confidence score, these two vertices are traded off to give a final 3D correspondence position for every vertex in $M_H^A(t)$.

Finally, $M_H^A(t)$ is deformed using non-rigid ICP [7] according to this correspondence, Section 5.3.6. This results in a refined mesh sequence $M_R(t)$ that has the geometric properties from all three reconstruction techniques.

5.3.3 Rim Computation

VH reconstruction produces the maximal volume consistent with a set of silhouettes. This leads to coarse geometric reconstructions particularly with relatively few cameras. A strength of this reconstruction technique is that, providing accurate silhouettes and camera calibration can be obtained, the resulting shape will enclose the true shape and will consists of points that lie on the true surface consistent with the silhouettes. Although these true surface points exist, which are commonly referred to as rims, it can be a challenging task to identify them.

Several techniques have been proposed to identify and represent rims, e.g. stereo correspondence along silhouette contours for pixel chains in depth maps [75] or 3D points using the rim mesh algorithm [66]. The work presented in this chapter employs a mesh based min-cut/max-flow formulation to compute the rims for a given mesh. Rims are represented in 3D as a subset of mesh edges that form a continuous line in the viewpoint of each camera and lie on the silhouette foreground/background boundary. This is denoted as $\{v_i, v_j\} \in \{R_k\}_{k=1}^{NC}$ where vertex $v_i$ and $v_j$ form a mesh edge in the rim $R_k$ of the $k^{th}$ camera. An overview of the approach is shown in Figure 5.4.

The VH reconstruction, represented as a triangular mesh surface, is converted into a bi-directional graph structure. Nodes in the graph represent mesh vertices and edges represent the connectivity between them. The capacity of each edge is proportional to the angle of the vector perpendicular to the camera direction ray and the average of the normal direction from the vertices that make up the edge, shown in Equation 5.1.

$$c(\vec{n}_i, \vec{n}_j, \vec{c}_D) = \sin\left(\frac{\pi}{2} - \cos^{-1}\left(\vec{c}_D \cdot \frac{\vec{n}_i + \vec{n}_j}{2}\right)\right)^2$$  (5.1)
5.3 Approach 1: Non-Rigid ICP Shape Optimisation

Fig. 5.4 Rims are computed using a min-cut/max-flow approach. Visible vertices are connected to the source node (green) and non-visible vertices to the sink (red). Vertices whose normal falls within an angular threshold, $a$, of the camera view direction are rim candidates (blue).

where $\vec{n}_i$ and $\vec{n}_j$ are the normalised vertex normal vectors of the $i^{th}$ and $j^{th}$ vertex, respectively, and $\vec{c}_D$ is the normalised camera direction vector. The squared sin function causes the cost to rise more rapidly as the angle deviates from the perpendicular of the view direction. This property makes the graph cut favour edges in which vertices are as close as possible to be perpendicular to the camera direction vector. This is done with respect to the camera direction vector $\vec{c}_D$. If the direction vector was computed on a per vertex basis, it would add a small bias to rim candidates towards the visible side of the mesh.

All vertices considered visible, as defined by a depth test, are connected to the source node, shown as green in Figure 5.4. Non-visible vertices are connected to the sink node, shown in red. Vertices, visible or non-visible, whose normal is within a threshold, $a$, of the camera direction vector are not connected to source or sink node, but are considered to be rim candidates, shown in blue. The minimum cut in the graph that separates the source and sink is positioned along the edges which are perpendicular to the camera.

To ensure that these edges on the graph-cut relate to a valid rim, the results are filtered so that all selected edges lie on a silhouette boundary. This is achieved by checking both vertices are on the silhouette boundary within a 3x3 pixel window. This process results in a subset of mesh edges used to represent the rims and is computed for each camera that views the subject. An example of rims extracted from a frame in the Dan dataset is shown in Figure 5.5.
5.3.4 Visual Hull Confidence

Although VH reconstruction contains true surface points, it can also contain artefacts known as phantom volumes. This is a result of misclassification of voxels due to self occlusions and camera distribution. To prevent phantom volumes causing shape estimation errors, a confidence score is assigned to each vertex of the VH. The confidence score is based on the Hausdorff distance between the VH and MVS mesh at each time frame, as defined in Equation 5.2.

\[
d_H(M_V(t), M_S(t)) = \max_{\bar{v}_i \in M_V(t)} \{ \min_{\bar{v}_s \in M_S(t)} \{ d(\bar{v}_i, \bar{v}_s) \} \} \tag{5.2}
\]
where \( d_H(M_V(t), M_S(t)) \) is the Hausdorff distance between the VH mesh \( M_V(t) \) and MVS mesh \( M_S(t) \), and \( \vec{v}_v \) and \( \vec{v}_s \) are vertices that exist in mesh \( M_V(t) \) and \( M_S(t) \), respectively. \( d(\vec{v}_v, \vec{v}_s) \) is the Euclidean distance between vertex \( \vec{v}_v \) and \( \vec{v}_s \).

The Hausdorff distance can also be calculated for each vertex of the mesh. This is the distance between a vertex \( \vec{v} \) and a given mesh \( M \), shown in Equation 5.3.

\[
d_H(\vec{v}, M) = \max \left( \min_{\vec{v}_X \in M} \{d(\vec{v}, \vec{v}_X)\} \right)
\]

where \( d_H(\vec{v}, M) \) is the Hausdorff distance for vertex \( \vec{v} \) for a given mesh \( M \).

The confidence assigned to a vertex \( \vec{v} \) is given by normalising the Hausdorff distance with respect to the Hausdorff distance between the mesh \( M_V(t) \) and \( M_S(t) \)

\[
\delta(\vec{v}, M_V(t), M_S(t)) = 1 - \frac{d_H(\vec{v}, M_S(t))}{d_H(M_V(t), M_S(t))}
\]

where \( \delta(\vec{v}, M_V(t), M_S(t)) \) is the confidence score for vertex \( \vec{v} \) given \( M_V(t) \) and \( M_S(t) \).

The confidence score ranges from zero to one, where zero is the lowest confidence and one is the highest confidence. Any vertices which form part of a rim, defined in the previous section, are assigned a confidence score of 1. Examples of the Hausdorff distance are shown in Figure 5.6 with phantom volumes clearly visible.

Fig. 5.6 Hausdorff distance is used to measure the visual hull confidence for (a) Roxanne dataset and (b) Dan dataset.
5.3.5 Correspondence Search

A key processing step of the method is the correspondence search which finds a corresponding 3D position for every vertex in the $M^H_A(t)$. The correspondence is used in the non-rigid ICP stage to deform $M^H_A(t)$ to match surface features. This search is conducted by finding the point that intersects the surface of $M_S(t)$ as well as high confidence surface regions of $M_V(t)$. The surface intersection point is found by moving a vertex along its associated vertex normal to the point where the Euclidean distance is minimised, Equation 5.5. This process is illustrated in Figure 5.7.

$$\min_{\vec{v}_i \in V} \min_{-d \leq \theta \leq d} d(\vec{v} + \theta \vec{n}, \vec{v}_i)$$ (5.5)

where $\vec{v}$ is the vertex that is seeking a corresponding intersection point for mesh $M$, $\vec{n}$ is vertex normal associated with $\vec{v}$, $\vec{v}_i$ is the $i^{th}$ vertex of mesh $M$, and $\theta$ is an offset factor between a minimum/maximum distance $d$.

The correspondence search is performed for each vertex in $M^H_A(t)$ resulting in two corresponding points: $(\vec{v}_i \rightarrow \vec{v}_V^i)$ and $(\vec{v}_i \rightarrow \vec{v}_S^i)$ which are obtained from $M_V(t)$ and $M_S(t)$, respectively. The final corresponding vertex $\vec{v}_i \rightarrow \vec{t}_i$ is an interpolation of the two intersection points, $\vec{v}_V^i$ and $\vec{v}_S^i$, based on the VH confidence for $\vec{v}_V^i$, shown in Equation 5.6.

$$\vec{t}_i = \delta_i \vec{v}_V^i + (1 - \delta_i) \vec{v}_S^i$$ (5.6)

where $\vec{t}_i$ is the corresponding point for vertex $v_i$, $\vec{v}_V$ and $\vec{v}_S$ are the positions that intersects the VH and MVS, respectively, and $\delta_i$ is the confidence of the $i^{th}$ vertex of the VH reconstruction. This was given previously by the normalised Hausdorff distance shown in Equation 5.4.

This effectively trades off VH mesh features with the MVS mesh. The weighting scheme ensures that vertices which correspond high confidence features in the visual hull reconstruction, e.g. rims, are given priority over the photo-consistency of MVS. It also ensures that when VH and MVS are close and the confidence is low, e.g. due to phantom volumes, the MVS surface is preferred.

5.3.6 Non-rigid ICP

In this work, the optimal step non-rigid ICP algorithm by Amberg et al. [7] is used to deform the $M^H_A(t)$ to match the corresponding surface point found in Section 5.3.5. The algorithm
5.3 Approach 1: Non-Rigid ICP Shape Optimisation

Fig. 5.7 Correspondence search to find the two points that intersect the VH and MVS mesh surfaces (pink). \( \vec{v} \) (blue) is moved along its normal to find points within a minimum/maximum distance that intersect with \( M_V(t) \) and \( M_S(t) \).

iteratively calculates an optimal affine transformation for each vertex in the \( M^H(t) \) to match the vertex correspondences. Details of the algorithm are given below.

An optimal transformation for each vertex can be found by solving the following sparse linear system given by Equation 5.7.

\[
e(X) = e_d(X) + \alpha e_s(X) + \beta e_l(X)
\]  

(5.7)

where \( e_d \) is the distance term, \( e_s \) is a regularisation term, \( e_l \) is a landmark term, \( \alpha \) is a stiffness weighing, \( \beta \) is used to scale the importance of landmarks, and \( X \) is a set of 4x3 affine transformation matrices as shown in Equation 5.8.

\[
X = [X_1, X_2, \ldots, X_{N_V}]^T
\]  

(5.8)

where \( X_i \) is a 4x3 affine transformation associated with the \( i^{th} \) vertex of the template mesh and \( N_V \) is the number of vertices in the template mesh.

The distance term, Equation 5.9, is used to minimise the difference between the template mesh vertices and the corresponding target positions.

\[
e_d(X) = \sum_{\vec{v}_i \in \bar{V}} \omega_i d(\bar{i}_i, X_i \bar{v}_i)^2
\]  

(5.9)
where \( v_i \) is a template mesh vertex, \( d(\vec{t}_i, \vec{v}_i) \) is the distance between the target vertex \( \vec{t}_i \) and \( \vec{v}_i \), and \( \omega_i \) is a weighing.

The weighting is equal to zero if no corresponding point can be located, and set to one if found. In this work, the correspondence search always results in a corresponding point, so the weighing is always set to one.

The smoothness term is used to ensure that the local neighbourhood of vertices, defined by the mesh connectivity, remain close to one another, shown in Equation 5.10.

\[
e_s(X) = \sum_{\{i,j\} \in N} \| (X_i - X_j)G \|^2_F
\]  

(5.10)

where \( e_s(X) \) is the regularisation term over all unique edge pairs in the mesh \( \{i,j\} \in N \) using the Frobenius norm, while \( X_i \) and \( X_j \) are the transformations applied to the \( i^{th} \) and \( j^{th} \) template vertices that are connected by edges in the local neighbourhood \( N \), and \( G = \text{diag}(1,1,1,1) \). Finally, the landmark term shown in Equation 5.11 defines the initial correspondence.

\[
e_l(X) = \sum_{\{v_i, l_i\} \in L} \| X_i v_i - l_i \|^2
\]  

(5.11)

where \( e_l(X) \) is the landmark term and \( \{v_i, l_i\} \) are pairs of vertex/landmark points in \( L \). In this work, the landmark term is not required as \( M_H(t) \) is close to its target shape and only requires the transfer of surface details. Sparse landmarks are generally used to drive large scale deformations.

An optimal affine transformation is found by solving Equation 5.7 in an iterative fashion. Stiffness weights are decreased from an initial stiffness \( \alpha^1 \) to a minimum stiffness \( \alpha^N_i \) where \( N_i \) is the number of iterations. \( \epsilon \) defines the convergence threshold criteria. The algorithm is solved as follows:

- Initialise \( X^0 \) with all transforms set to zero
- for each stiffness \( \alpha^i \in \{ \alpha^1, \alpha^2, ... \alpha^N_i \} \), \( \alpha^i > \alpha^{i+1} \)
  - Until \( \|X^i - X^{i-1}\| < \epsilon \)
  - Determine \( X^j \) as the optimal deformation
5.3.7 Summary

This section has presented a shape optimisation approach based on non-rigid ICP. An evaluation of this approach is presented in Section 5.5. The TC mesh is deformed at each frame to match rims, extracted from a VH reconstruction and the MVS which is as photo-consistent as allowed by the geometry. However, if VH and MVS reconstruction have underlying problems due to errors in the calibration, these will propagate through to the refined shape. In order to resolve this, a shape optimisation framework that uses bundle adjustment is proposed in Section 5.4. This formulation allows geometry and calibration to be jointly optimised to minimise reprojection errors.

5.4 Approach 2: Bundle Adjustment Shape Optimisation

5.4.1 Overview

The first shape optimisation approach presented in this chapter trades off shape details between the VH and the MVS reconstructions. If the camera calibration is inaccurate, errors from the VH and MVS reconstruction techniques will propagate to the non-rigid ICP shape optimisation. In this section, a second shape optimisation method is presented that approaches the task as a bundle adjustment problem, refining both geometry and camera parameters.

In Chapter 4, it was demonstrated that optical flow could be used to correct texture misalignment in the camera domain across wide baseline camera setups. These reprojection errors, caused by camera calibration and geometric inaccuracies, gave rise to texture artefacts, e.g. blurring and ghosting. Other work [21, 23, 35] has also used optical flow to correct texture artefacts, but these methods fail to address the underlying issue of geometric and calibration inaccuracies. This is the objective of the approach presented in this section.

The proposed algorithm uses the optical flow correspondences between the wide baseline cameras and minimises the reprojection error with respect to both camera calibration parameters and scene geometry. To simplify the problem, vertex positions are constrained to lie along the associated vertex normal giving one parameter per vertex to optimise. An overview of the approach is shown in Figure 5.8.
Firstly, as with the previous approach, the TC mesh \( \{ M_A(t) \}_{1}^{N_f} \) is sub-divided using the mid-point sub-division scheme \( \{ M_A^H(t) \}_{1}^{N_f} \) to create a higher resolution mesh. Secondly, in order to find a dense surface correspondence between wide baseline cameras, the method previously proposed in Section 4.5 is used. This method computes the dense optical flow correspondence between the projectively textured model from the viewpoint of a reference camera textured with its camera image and textured with an alternative camera image. Ideally, texture features should be aligned across views and should result in zero flow vectors. However, due to calibration and geometric errors, the texture slides across the surface. Performing optical flow on these images identifies these errors in alignment and is able to calculate the displacements in pixel coordinates. This process is carried out for all \( N_C \) cameras resulting in \( N_C^2 \) optical flow correspondence fields per frame in the overlapping areas between camera views, \( \{ \{ O(i,j,t) \}_{i=1}^{N_C} \}_{j=1}^{N_C} \). Due to visibility constraints, some cameras will not contain any valid flow vectors and can be discarded. The vertex reprojection and bundle adjustment stages are now described.

### 5.4.2 Vertex Reprojection

Given a \( M_A^H(t) = \{ V \} \) consisting of vertices \( V \) with a set of calibrated cameras \( \{ C_i \}_{i=1}^{N_C} = \{ \pi(i,t), I(i,t), R_D(i,t) \} \) where \( I(i,t) \) and \( R_D(i,t) \) are the camera images and rendered depth maps, respectively, and \( O(i,j,t) \) is a set of optical flow correspondence fields between the \( i^{th} \) reference camera and \( j^{th} \) texture camera at frame \( t \). The process for establishing a refined vertex correspondence between a pair of cameras is as follows:
1. Project 3D mesh vertex $\vec{v}$ into image domain of the $i^{th}$ camera, resulting in 2D pixel location $\vec{p}_i$.

2. Adjust $\vec{p}_i$ in image domain of the $i^{th}$ camera by optical flow vector $\vec{o}_{i,j}$ obtained from pixel location $\vec{p}_i$ in optical flow field $O(i, j, t)$ between $i^{th}$ and $j^{th}$ cameras.

3. Back project 2D pixel $\vec{p}_i + \vec{o}_{i,j}(\vec{p}_i)$ onto 3D surface in world coordinate system using depth map $R_D(i, t)$, resulting in 3D surface point $\vec{v}_j$.

4. Project $\vec{v}_j$ into the camera image domain of the $j^{th}$ camera, resulting in 2D pixel location $\vec{p}_j$.

This process results in two corresponding pixel locations between the $i^{th}$ and $j^{th}$ cameras, $\vec{p}_i$ and $\vec{p}_j$. The process is illustrated in Figure 5.9. Full correspondence is obtained by performing this for every vertex in all pairs of cameras where the vertex is evaluated by a depth test to be visible. An example of this correspondence and the magnitude of the correction vectors is shown in Figure 5.10.

Fig. 5.9 Vertex reprojection process.
Fig. 5.10 Vertex reprojection based on optical flow computed with respect to camera 2 (b). Camera 3 (a) and camera 1 (c) show reprojected vertices across camera views. Only vertices common to all three views are displayed.
5.4.3 Bundle Adjustment

In this step, the pixel correspondences computed in the previous vertex reprojection step are used to optimise the camera parameters and vertex positions using a bundle adjustment solver [2]. The optimisation assumes a pin-hole camera model using four intrinsic camera parameters and six extrinsic. The four intrinsic parameters used in the camera model are: horizontal and vertical focal lengths, and the 2D principal point. Extrinsic parameters include three parameters for translation and three for rotation using the axis-angle representation. Shape parameters are a scalar per mesh vertex which translates a vertex along its normal. This is done to simplify the optimisation and ensure that vertices remain evenly distributed on the mesh surface as in the template. This results in \( 10N_C + N_V \) parameters per frame, where \( N_C \) is the number of cameras used to capture the subject and \( N_V \) is the number of vertices. An optimal calibration and geometry can be found by minimisation of Equation 5.12.

\[
\min_{\{C_i\}_{i=1}^{N_C}, \{\theta_k\}_{k=1}^{N_V}} \sum_{k=1}^{N_C} \sum_{i=1}^{N_V} \sum_{j=1}^{N_C} v(k, i, j) \left( |\tilde{p}_k^j - \tilde{p}(\bar{v}_k + \bar{n}_k \theta_k, C_j)|^2 \right)
\]

(5.12)

where \( v(k, i, j) \) is a binary visibility function that returns 1 if the \( k^{th} \) vertex is visible in both the \( i^{th} \) and \( j^{th} \) cameras and 0 otherwise. \( \tilde{p}_k^j \) is the flow adjust 2D position of the \( k^{th} \) vertex reprojected into the \( j^{th} \) camera view and is treated as the measured, \( \tilde{p}(\bar{v}_k + \bar{n}_k \theta_k, C_j) \) is the predicted 2D location of vertex \( \bar{v}_k \) translated by a factor of \( \theta_k \) along the vertex normal \( \bar{n}_k \) in camera \( C_j \).

This approach is evaluated in Section 5.5.
5.5 Results and Evaluation

In this section, the two proposed shape optimisation approaches are evaluated in terms of silhouette overlap and reprojection error, defined in Section 5.5.1. All tests were performed on a Dell Optiplex 9010 desktop computer running Ubuntu 12.04 with an Intel i7 processor, 8GB of RAM and a Nvidia GeForce FX 640 graphics card. The non-rigid ICP algorithm by Amberg et al. [7] was implemented using Eigen version 3.2.5. Min-cut/max-flow was performed using the publicly available implementation by Boykov and Kolmogorov [13], GCO version 3.01. Optical flow was performed using the method of Brox et al. [15] and Farneback [37] from the OpenCV library version 2.4.11. Bundle adjustment was implemented using Ceres Solver library [2]. Mid-point sub-division was performed using the MeshLab application version 1.3.0 [25]. The rendering application was implemented using C++11, OpenGL 4.3, and OpenGL Shading Language 1.5.

The non-rigid ICP algorithm used $\alpha^1 = 2.5$ and decreased in steps of 0.05 until $\alpha^{N_i} = 2.3$ with $\varepsilon = 1.0$ as the convergence criteria. These parameters were chosen experimentally as they allow the algorithm to recover surface details while maintaining a relatively smooth surface. A low $\alpha^1$ results in a noisy surface and a high $\alpha^1$ does not allow small details to be recovered.

5.5.1 Error Metrics

The evaluation of multiple view shape reconstruction is a challenging problem in itself. Rigid objects can be laser scanned and geometrically compared to the reconstructed object as a ground truth comparison. However, for non-rigid dynamic objects, such as people, it is not currently possible to obtain ground truth geometry. A quantitative evaluation is performed using the silhouette overlap ratios [87] and reprojection error as measured by optical flow [82]. Details of both error metrics are given below.

Silhouette Overlap Ratios

Silhouette overlap is measured by three ratios: the hit ratio, background ratio and overlap ratio, shown in Equations 5.13, 5.14 and 5.15, respectively. All ratios use pixel counts resulting from Boolean operations on the extracted silhouettes and geometry masks rendered
from the point of view of the respective camera. The extracted silhouette is seen as the ground truth to which the rendered masks are evaluated against.

The hit ratio describes the ratio of correct pixel in the render mask to the ground truth silhouette, shown in Equation 5.13.

\[
 r_H(S(i,t),R(i,t)) = \frac{R(i,t) \cap S(i,t)}{S(i,t)} 
\]  

(5.13)

The background ratio measures the ratio of false positive pixels against the resulting pixels, shown in Equation 5.14.

\[
 r_B(S(i,t),R(i,t)) = \frac{|S(i,t) - R(i,t)|}{R(i,t)} 
\]  

(5.14)

Finally, the overlap ratio measures the overall correctness using the ratio of the intersection to the union of the silhouette and rendered mask, shown in Equation 5.15.

\[
 r_O(S(i,t),R(i,t)) = \frac{R(i,t) \cap S(i,t)}{R(i,t) \cup S(i,t)} 
\]  

(5.15)

where \(S(i,t)\) and \(R(i,t)\) are the extracted silhouette and rendered mask of the \(i^{th}\) camera in frame \(t\), respectively.

Accurate render masks will result in a high overlap and hit ratio and a low background ratio. Results in this chapter are expressed as percentages using the mean for each camera over a sequence. The results of these measures are visualised using the following colour scheme: common overlapping pixels (white); pixels in silhouette but not mask (green); pixels in mask but not silhouette (red).

**Reprojection error**

Reprojection error is measured by projectively texturing the geometry from the viewpoint of a camera using the texture from a single alternative camera and computing an optical flow correspondence field between the projectively textured rendered image and the camera image. This gives a measure of how much texture misalignment exists between the reconstructed and real captured data. The flow vectors are only analysed for common surface between the real and rendered data. An accurate reconstruction would result in zero flow vectors between cameras. Therefore the lower the value, the closer the reconstruction is to the
true surface. Values are expressed as mean, standard deviation and maximum value of the magnitude.

### 5.5.2 Silhouette Overlap Evaluation

Table 5.1 presents an evaluation of the silhouette overlap ratios for the evaluation datasets. The results present the average overlap, hit and background ratios for each camera over all sequence frames. Figure 5.12 and 5.13 visualise these results. It can be seen that the VH reconstruction method produced the highest overlap ratios of all the techniques. This is because the objective of VH reconstruction is to produce geometry consistent with a set of silhouettes. However, VH reconstruction is based on the assumption that both camera calibration and matting are accurate.

From the proposed approaches, the non-rigid ICP based method was able to almost match the VH in terms of overlap ratios. This comes from using the rims, derived from the silhouette, as surface features which are then used in the non-rigid ICP as target points. The MVS and TC meshes have lower overlap ratios, highlighting that these are smaller meshes compared to the VH. Reducing the stiffness parameter gives a better fit to the corresponding points, but results in surface noise. Figure 5.11 demonstrates that this method was able to add shape detail to the face which was smoothed out by the model-free shape reconstruction pipeline. In contrast, the BA approach makes small changes to the overall geometry and was unable to improve the silhouette overlap ratios which remain close to their starting value. However, the BA method does not use rim constraints, derived from silhouette information in its formulation. Instead, it relies solely on minimising reprojection error.

![Image](image.png)

**Fig. 5.11** Geometry of profile view of Dan character after refinement. (a) Heat map of difference between BA geometry and TC geometry, (b) refined geometry using BA method, (c) TC geometry, (d) refined geometry using non-rigid ICP method, and (e) heat map of difference between non-rigid ICP geometry and TC geometry.
Table 5.1 Silhouette overlap measured by overlap ratio (O), hit ratio (H) and background ratio (B) comparing visual hull (VH), multiple view stereo (MVS) and temporally consistent (TC), proposed non-rigid ICP (ICP) and bundle adjustment (BA) shape reconstruction methods. The best reconstruction method is highlighted in bold for each ratio in each dataset.

<table>
<thead>
<tr>
<th>Camera</th>
<th>VH</th>
<th>MVS</th>
<th>TC</th>
<th>ICP</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean overlap ratios per camera (%)</td>
<td>Mean (%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Dan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>O</td>
<td>B</td>
<td>H</td>
<td>O</td>
</tr>
<tr>
<td>VH</td>
<td>96.9</td>
<td>95.9</td>
<td>95.3</td>
<td>95.0</td>
<td>93.1</td>
</tr>
<tr>
<td>O</td>
<td>97.0</td>
<td>96.2</td>
<td>95.6</td>
<td>95.8</td>
<td>97.0</td>
</tr>
<tr>
<td>B</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0.9</td>
<td>4.2</td>
</tr>
<tr>
<td>MVS</td>
<td>93.3</td>
<td>92.2</td>
<td>91.6</td>
<td>91.9</td>
<td>91.9</td>
</tr>
<tr>
<td>O</td>
<td>93.4</td>
<td>92.3</td>
<td>91.7</td>
<td>92.2</td>
<td>94.5</td>
</tr>
<tr>
<td>B</td>
<td>0.0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>2.9</td>
</tr>
<tr>
<td>TC</td>
<td>91.4</td>
<td>90.0</td>
<td>89.1</td>
<td>88.6</td>
<td>90.3</td>
</tr>
<tr>
<td>O</td>
<td>91.8</td>
<td>90.2</td>
<td>89.5</td>
<td>89.5</td>
<td>93.1</td>
</tr>
<tr>
<td>B</td>
<td>0.4</td>
<td>0.4</td>
<td>0.5</td>
<td>1.2</td>
<td>3.2</td>
</tr>
<tr>
<td>ICP</td>
<td>95.7</td>
<td>94.6</td>
<td>93.9</td>
<td>93.2</td>
<td>92.3</td>
</tr>
<tr>
<td>O</td>
<td>96.2</td>
<td>95.1</td>
<td>94.3</td>
<td>94.4</td>
<td>96.3</td>
</tr>
<tr>
<td>B</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5</td>
<td>1.3</td>
<td>4.3</td>
</tr>
<tr>
<td>BA</td>
<td>91.7</td>
<td>90.3</td>
<td>89.2</td>
<td>88.7</td>
<td>90.6</td>
</tr>
<tr>
<td>O</td>
<td>92.1</td>
<td>90.6</td>
<td>89.6</td>
<td>89.7</td>
<td>93.2</td>
</tr>
<tr>
<td>B</td>
<td>0.5</td>
<td>0.4</td>
<td>0.5</td>
<td>1.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Roxanne</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>O</td>
<td>B</td>
<td>H</td>
<td>O</td>
</tr>
<tr>
<td>VH</td>
<td>95.1</td>
<td>94.1</td>
<td>91.0</td>
<td>90.3</td>
<td>92.2</td>
</tr>
<tr>
<td>O</td>
<td>95.9</td>
<td>94.9</td>
<td>92.7</td>
<td>92.4</td>
<td>93.6</td>
</tr>
<tr>
<td>B</td>
<td>0.9</td>
<td>1.0</td>
<td>2.2</td>
<td>2.5</td>
<td>1.7</td>
</tr>
<tr>
<td>MVS</td>
<td>92.0</td>
<td>91.6</td>
<td>88.4</td>
<td>87.5</td>
<td>89.6</td>
</tr>
<tr>
<td>O</td>
<td>92.3</td>
<td>91.8</td>
<td>89.3</td>
<td>88.6</td>
<td>90.3</td>
</tr>
<tr>
<td>B</td>
<td>0.3</td>
<td>0.3</td>
<td>1.3</td>
<td>1.6</td>
<td>0.8</td>
</tr>
<tr>
<td>TC</td>
<td>90.3</td>
<td>89.7</td>
<td>86.5</td>
<td>85.4</td>
<td>88.2</td>
</tr>
<tr>
<td>O</td>
<td>90.8</td>
<td>89.9</td>
<td>87.4</td>
<td>86.5</td>
<td>89.0</td>
</tr>
<tr>
<td>B</td>
<td>0.5</td>
<td>0.2</td>
<td>1.3</td>
<td>1.5</td>
<td>1.0</td>
</tr>
<tr>
<td>ICP</td>
<td>94.9</td>
<td>93.7</td>
<td>92.8</td>
<td>91.9</td>
<td>92.1</td>
</tr>
<tr>
<td>O</td>
<td>96.0</td>
<td>94.7</td>
<td>93.9</td>
<td>93.1</td>
<td>93.6</td>
</tr>
<tr>
<td>B</td>
<td>1.2</td>
<td>1.0</td>
<td>1.3</td>
<td>1.4</td>
<td>1.7</td>
</tr>
<tr>
<td>BA</td>
<td>90.2</td>
<td>89.8</td>
<td>86.4</td>
<td>85.3</td>
<td>88.0</td>
</tr>
<tr>
<td>O</td>
<td>90.7</td>
<td>90.0</td>
<td>87.5</td>
<td>86.5</td>
<td>88.9</td>
</tr>
<tr>
<td>B</td>
<td>0.6</td>
<td>0.3</td>
<td>1.4</td>
<td>1.6</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Fig. 5.12 Silhouette overlap for reconstructed meshes in Dan dataset. (a) VH, (b) TC, (c) non-rigid ICP shape optimisation, and (d) BA shape optimisation. Common overlapping pixels (white), pixels in silhouette but not in render mask (red), and pixels in render mask but not in silhouette (green).
5.5 Results and Evaluation

(a) Visual hull geometry

(b) Temporally consistent geometry

(c) Non-rigid ICP shape optimisation geometry

(d) Bundle adjustment shape optimisation geometry

Fig. 5.13 Silhouette overlap for (a) VH, (b) TC, (c) non-rigid ICP shape optimisation, and (d) BA shape optimisation. Common overlapping pixels (white), pixels in silhouette but not in render mask (red), and pixels in render mask but not in silhouette (green).
5.5.3 Reprojection Error Evaluation

Table 5.2 reports reprojection error results for each camera over the sequence for the different reconstruction and optimisation approaches, expressed as mean, standard deviation and maximum value. Projectively texture results with heatmaps are shown in Figure 5.15 and 5.16 for Dan and Roxanne datasets, respectively, and close ups views from Dan dataset in Figure 5.14. The non-rigid ICP approach achieved an average reprojection error similar to that of the VH. This occurs because the geometry was deformed to match the rim features of the VH. Areas of the surface where the VH and MVS reconstruction yield a similar result remain as a similar surface. However, areas where the VH and MVS deviate from one another and do not occur on a rim, are moved closer to the MVS surface to improve photo-consistency. Figure 5.14 shows the presence of texture artefacts.

These results also show that the BA shape optimisation approach was able to achieve the lowest average reprojection error with the smallest standard deviation, even compared to MVS. This indicates that rendering results using the optimised shape and calibration should contain the least amount of texture artefacts. Figure 5.14 clearly shows that the BA approaches produces the sharper textures and precise alignment of facial features and the stripes in character Dan’s shirt. This is achieved by jointly optimising geometry and camera calibration to minimise reprojection error. However, the maximum reprojection was still high for the BA method which may be caused by unreliable flow vectors at boundaries and depth discontinuities.

![Fig. 5.14 Shape optimisation results for Dan dataset close up: Projectively texture geometry (a-e) and heat map showing colour difference with respect to TC (f-i)](image-url)
Fig. 5.15 Shape optimisation results for Dan dataset: Projectively texture geometry (a-e) and heat map showing colour difference with respect to TC (f-i)
Fig. 5.16 Shape optimisation results for Roxanne dataset: Projectively texture geometry (a-e) and heat map showing colour difference with respect to TC (f-i)
Table 5.2 Reprojection error results are expressed as mean, standard deviation (SD) and maximum (MAX) of all optical flow vectors in the common overlapping surface between all pairs of cameras over all frames. Data is shown for visual hull (VH), multiple view stereo (MVS), temporally consistent (TC), proposed non-rigid ICP based optimisation (ICP) and bundle adjustment (BA) shape optimisation. Best results are highlighted in bold for each dataset.

<table>
<thead>
<tr>
<th></th>
<th>Reprojection Error per Camera (pixels)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Dan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VH</td>
<td>Mean</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>1.14</td>
</tr>
<tr>
<td></td>
<td>MAX</td>
<td>11.15</td>
</tr>
<tr>
<td>MVS</td>
<td>Mean</td>
<td>1.38</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td>MAX</td>
<td>18.22</td>
</tr>
<tr>
<td>TC</td>
<td>Mean</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>1.22</td>
</tr>
<tr>
<td></td>
<td>MAX</td>
<td>14.86</td>
</tr>
<tr>
<td>ICP</td>
<td>Mean</td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>1.27</td>
</tr>
<tr>
<td></td>
<td>MAX</td>
<td>15.78</td>
</tr>
<tr>
<td>BA</td>
<td>Mean</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>1.15</td>
</tr>
<tr>
<td></td>
<td>MAX</td>
<td>11.65</td>
</tr>
</tbody>
</table>

| Roxanne|     |     |     |     |     |     |     |     |      |
| VH    | Mean| 1.59| 1.75| 2.42| 2.51| 2.52| 1.55| 2.07| 1.87| 2.03 |
|       | SD  | 1.39| 1.57| 2.21| 2.17| 2.43| 1.48| 2.06| 1.63| 1.87 |
|       | MAX | 10.43| 16.43| 22.74| 20.31| 20.98| 15.91| 20.12| 14.63| 22.74 |
| MVS   | Mean| 1.11| 1.28| 1.78| 1.90| 1.75| 1.27| 1.26| 1.20| 1.44 |
|       | SD  | 0.98| 1.17| 1.58| 1.67| 1.62| 1.25| 1.16| 1.11| 1.32 |
|       | MAX | 9.69| 12.00| 22.14| 19.71| 17.83| 15.44| 13.33| 11.40| 22.14 |
| TC    | Mean| 1.11| 1.16| 1.58| 1.54| 1.54| 1.20| 1.14| 1.18| 1.31 |
|       | SD  | 1.02| 1.14| 1.48| 1.45| 1.45| 1.21| 1.12| 1.18| 1.26 |
|       | MAX | 10.11| 11.84| 27.20| 19.20| 25.12| 15.30| 12.77| 11.46| 27.20 |
| ICP   | Mean| 1.64| 1.71| 2.35| 2.49| 2.37| 1.65| 1.93| 1.87| 2.00 |
|       | SD  | 1.34| 1.48| 2.08| 2.14| 2.17| 1.54| 1.75| 1.68| 1.77 |
| BA    | Mean| 1.10| 1.12| 1.55| 1.52| 1.48| 1.15| 1.15| 1.19| 1.28 |
|       | SD  | 1.06| 1.10| 1.50| 1.49| 1.43| 1.17| 1.14| 1.23| 1.27 |
|       | MAX | 13.79| 12.52| 23.40| 22.10| 18.05| 13.49| 13.12| 13.68| 23.40 |
5.6 Limitations

This section discusses limitations of the shape optimisation approaches presented in this chapter. The non-rigid ICP approach used VH and MVS reconstruction to add details lost in the reconstruction pipeline. If calibration errors exist, they will have adversely affected the VH and MVS reconstructions. A limitation of the non-rigid ICP based method is that using existing reconstructions information that may have been based on erroneous calibration will result in errors. For this reason, the second approach was proposed that used only the TC geometry in bundle adjustment framework to optimise calibration and geometry.

The current BA implementation does not include any shape regularisation which can result in a noisy surface appearance. Another drawback of the presented implementation is that BA is currently performed on a per frame basis which results in slight variations in the camera calibration from frame-to-frame. In order to make the calibration stable, the process could be run over all frame in a sequence optimising a single set of calibration parameters. The BA approach results in small changes in the shape, this was shown to be the case with the silhouette overlap analysis. This could be improved by adding rim constraint to the formulation. This method is also reliant on the accuracy of the optical flow algorithm, errors in optical flow correspondence will result in geometric. A filtering process could be added to verify each flow-vector based on photo-consistency.

5.7 Conclusion

This chapter has presented two different shape optimisation techniques which have focused on adapting an existing TC mesh representation to better reflect the true shape of the subject. The preservation of TC geometry is essential to allow reuse of the data for animation purposes.

The first method approached the task as a registration problem and combined strengths of VH and MVS reconstruction techniques. This was able to give a higher silhouette overlap than other MVS and TC reconstruction methods with a reprojection error similar to that of the VH reconstruction. This approach was successful in restoring facial features, such as the nose, which can be smoothed out in the MVS and TC processing stage, due to shape regularisation and mesh resolution.
The second approach used a bundle adjustment framework to jointly optimise geometry and camera calibration. This approach was able to reduce the reprojection error and resulted in reduced texture artefacts during view dependent rendering.

Both optimisation methods were compared to the VH [65], MVS [97] and TC [17] reconstructions in terms of silhouette overlap and reprojection error. Reprojection error was measured by optical flow correspondence in the overlapping camera regions when projected to a common shape proxy from a camera viewpoint. The proposed non-rigid ICP approach was able to almost match the silhouette overlap performance of the VH reconstruction. In the case of the Roxanne dataset, it was able to improve the VH reconstruction result. The BA approach was able to achieve a lower reprojection error in both datasets but not able to improve the silhouette overlap metric.

Neither of the proposed approaches is able to completely solve the problem. A simple next step that could be explored is to combine these methods and alternate the optimisation between the methods in an iterative fashion.
Chapter 6

Online Interactive 4D Character Animation

6.1 Introduction

So far this thesis has introduced methods that enable the compact storage and efficient view-dependent rendering of 4D video data. A shape refinement technique was also presented that allows fine scale surface features to be recovered by combining multiple reconstruction modalities. These contributions are key to obtaining high-quality and compact data that can be efficiently visualised. In this chapter, it is shown how these contributions could potentially be brought together to deliver realistic interactive virtual characters via the internet to a WebGL enabled browser. Compact data representations are important to minimise data loading times and enable efficient visualisation to open video-based character animation and free-viewpoint video rendering technology up to as many devices as possible.

This chapter demonstrates, for the first time, that 4D video can be used to provide interactive content to a WebGL based platform. Previous work has utilised WebGL to deliver free-viewpoint video of sporting events, generated from a multiple camera capture [16], to a web browser. However, this framework was limited to simple replay of the captured data. In contrast, the work presented in this chapter builds on previous work by adding additional processing steps to the 4D video data. Existing animation techniques are adapted to allow users to interactively control 4D characters in a WebGL environment.
6.2 Background

6.2.1 Character Animation Techniques

In the last decade, markerless performance capture techniques have been developed to capture geometry, dynamic appearance and motion of the human body from a multiple camera setup [28, 93, 94, 104, 108]. Traditional motion capture (MoCap) techniques only capture skeletal pose that is retargeted to a virtual character. This conveys a sense of realistic motion and is used extensively for authoring animation in games and film. Markerless performance capture [27, 94, 104, 108] aims to reproduce the fine details of human performance which cannot be captured by skeletal pose, including realistic shape, motion and appearance, e.g. wrinkles in clothing, facial expression and the dynamics of hair.

Motion graph techniques [9, 64, 67] are widely used to synthesise new skeletal movements from a database of skeletal MoCap examples. A graph structure is used to represent possible transitions between different motions while the traversal of the graph results in concatenative animations. The production of character animation from 4D videos is an analog to conventional MoCap based character animation. Huang et al. [51] introduced surface motion graphs which used unaligned 3D mesh sequences to produce concatenative animations of human motion. Transitions were determined between 3D mesh sequences by identifying the most similar frames in terms of both 3D shape and motion similarity.

Recent work bridges the gap between skeletal-based animation and markerless motion capture using a hybrid skeletal-surface motion graph [54]. This considered both 3D shape and appearance similarity and using 4D video as input. Seamless transitions were created by linear interpolation of 3D meshes in overlapping frames at identified transitions. This work also provided a means to drive a 4D video based animation with readily available skeletal MoCap data which extends the range of motion that can be produced.

4D parametric motion graphs [22] also used 4D video as input and enabled parametrisation of motions. A parametric motion is defined as a pair of motions with a semantic motion parameter to control the synthesis of a new intermediate motion. For instance, given walk and jog motions, a parameter can be used to create a new motion with any speed between walk and jog. Parametric motion graphs also allow responsive transitions between parametric motions. Transitions are identified on the fly and balance the latency and smoothness of the transition. 4D video textures [23] provide a run-time optical flow based texture warping to seamlessly blend textures at transitions. However, the computational cost is high for run-time transition
6.2 Background

Identification and optical flow based texture warping. Due to the WebGL implementation limits, this work uses fixed transitions between motions for animation control.

Boukhayma and Boyer [12] introduced the essential graph, a data structure to organise and create animations from multiple 4D video sequences of human characters. The approach combined dynamic time warping and variable length blended segments to generate mesh based interpolations at sequence transitions. High level path and motion constraints were defined and graph based search was used to identify the optimal path through the essential graph structure. In contrast, Casas et al. [22] compute optimal transitions on-the-fly based on user interaction.

6.2.2 WebGL

WebGL is a cross-platform, JavaScript based API for rendering 3D graphics natively in a web browser, e.g. Google Chrome, Mozilla Firefox, and is managed by the Khronos group. WebGL currently supports the OpenGL ES 2.0 specification [59] which is becoming increasingly supported by the latest mobile devices, e.g. smart phones and tablets, as well as set-top boxes. The cross-platform nature of WebGL makes it an ideal platform for the delivery of interactive 3D content. It also requires no specialist software, only a standard HTML5 web browser, and therefore opens this technology up to the widest possible audience.

Previously, Budd et al. [16] used WebGL to deliver free-viewpoint video of sport events [16] over the internet. Multiple camera capture was used to generate geometry sequences that were reconstructed offline. The time-varying geometry was transferred via HTTP along with the calibrated camera images. WebGL was then used to projectively texture the geometry based upon the user selected viewpoint using the same pipeline as presented in Section 2.4.1. To handle the large storage, transmission and memory overheads, this work resorted to down sampling the captured images which reduced the visual quality. This framework was also limited to the replay of reconstructed data.

In contrast to previous work, the approach presented in this chapter applies further offline processing steps to create temporally consistent geometry, texture maps and a motion graph with fixed transition points. These preprocessing steps remove the need for online depth testing that requires additional computational resources including texture buffers and multiple off-screen renders. These representations allow the implementation of an efficient interactive WebGL based character animation and rendering engine using 4D video.
6.3 System Overview

The system consists of three main stages: offline data processing, data storage and transfer using an internet connected server, and the client side application. Each stage is described below and an overview of the system is shown in Figure 6.1.

Fig. 6.1 Geometry, textures and motion graph data are hosted on a server. When loaded, the client application sends a HTTP request to the server and downloads the data into the client memory. Once initialised, the character animation engine (CAE) keeps track of the current state of the character based on the motion graph and user interaction. The data required to display the appropriate state given by the CAE is passed to the renderer to be displayed in the WebGL window.
6.3 System Overview

6.3.1 Offline Data Processing

4D Motion Graph Construction

A 4D motion graph with 4D video as input is constructed in order to represent possible inter- and intra-sequence transitions. Analogous to motion graph [64] for skeletal MoCap data, this allows captured motion sequences to be seamlessly concatenated to produce new motions.

To identify potential transitions, the frame-to-frame similarity metric is measured across all sequences. Both geometry and appearance similarity are considered. A 6D Shape-Colour Histogram is extracted for each mesh. This partitions the space into disjoint cells and counts the number of occupied volume elements falling into each bin together with their RGB colour distribution giving a 6D histogram signature [54] for each frame.

A similarity measure \( c(M_i, M_j) \) is defined between two meshes \( M_i \) and \( M_j \) by minimizing the difference between their corresponding bins with respect to rotation about the vertical axis,

\[
c(M_i, M_j) = \min_{\phi} \| h(M_i, 0) - h(M_j, \phi) \| \tag{6.1}
\]

where \( h(M_i, 0) \) and \( h(M_j, \phi) \) denote the extracted 6D shape-colour histogram for \( M_i \) and \( M_j \), respectively. The earth mover’s distance [84] is used to compute the distance between the sparse 6D shape-colour histograms. Equation 6.1 is minimized in a computationally efficient way. Firstly, a fine histogram is generated initially at an order of magnitude higher resolution than the desired vertical bin size. Secondly, the fine histogram is then shifted by the fine bin size and re-binned to a coarse histogram for comparison.

Given a pair of motion sequences (transfer from and transfer to), a transition is determined by a tuple \((m, n, N_O)\), \( m \) and \( n \) for identified location in motion sequences and \( N_O \) for overlap length. The optimal transition can be found by minimising Equation 6.2, given Equation 6.3.

\[
(m_{opt}, n_{opt}, N_{opt}) = \arg \min_{m,n,N_O} \sum_{k=-N_O}^{N_O} \alpha'(k) \cdot C_{m+k,n+k} \tag{6.2}
\]

\[
\alpha'(k) = \min\left(1 - \frac{k + N_O}{2N_O}, \frac{k + N_O}{2N_O}\right) \tag{6.3}
\]
where $\alpha'(k)$ denotes the weighting for linear blending at transitions. This optimisation is performed as an adaptive temporal filtering with window size $2N_O + 1$ and weighting $\alpha'(k)$ on the precomputed similarity matrix $C$.

Parametric motion is supported within 4D motion graphs [22]. A parametric motion node contains two or more motion sequences, e.g. a walk and a jog, parametrised to allow generation of any motion in between, e.g. the speed between a walk and a jog. Due to computational cost, the transition between parametric motions is precomputed rather than on the fly as in previous work [22, 23]. Fixed transitions are precomputed using the same method as previously described. When transferring between motions, the motion parameter will first adjust to reach those transitions and then transfer via them. Although multiple transitions are possible, due to the computational and data transfer cost for a WebGL application, only the best transition between each pair of motion sequences is kept. The surface motion graph is stored in a XML file format for future use. An example of a surface motion graph is shown in Figure 6.2.

![Motion Graph Example](image)

**Fig. 6.2** Motion graph example showing how different motion nodes are connected.

### Texture Data

Texture data is stored as PNG images on the web server and at run time transferred via HTTP and loaded into client memory. WebGL currently has limited support for non-power-of-two textures which restricts the dimensions of textures. The MLTR or other texture representations can be used within this framework. Also, due to implementation limitations, the MLTR texture layers are tiled into a single power-of-two sized PNG image with the single
channel camera assignment maps combined into a three channel RGB image, as shown in Figure 6.3.

![Image](image.png)

Fig. 6.3 Adapting the MLTR for WebGL based rendering. Layers are tiled into a single power-of-two texture with greyscale camera assignment combined into a one RGB image (bottom right).

Demonstrations presented in this chapter used a single texture map per frame. This was created by blending the first three texture layers with the colours weighted proportional to the angle between the polygon normal and capture cameras as defined by the camera assignment map. This results in a reduction in quality due to a loss of view-dependent information and may also result in blurring and ghosting artefacts for inaccurate geometry.

### 6.3.2 Client Application

The application was implemented in JavaScript and consists of two main components: a character animation engine (CAE) and a WebGL-based renderer. The client application starts by downloading all the required mesh, texture and motion graph data. The CAE maintains the current state of the character based on motion graph states, available transitions, and user input. The frame, or pair of frames in the case of parametric motion, required to be displayed are sent to the WebGL renderer.
Character Animation Engine

The CAE is implemented based on the previously constructed parametric motion graph and allows user interactive control over motion transfer and motion parameter adjustment. Figure 6.4 illustrates the flow chart of the CAE. The motion graph and the motion database are first loaded using an XML file format. User interaction results in a queue of motion requests. The CAE calls a traverse function to step through the motion graph and each step points to data in the motion database. Depending on whether a frame needs to be blended, single or multiple pairs of mesh and texture data are passed to the WebGL renderer. Playing will finish when no more motion requests are in the queue. However, if the current motion is defined as a loop motion, playing will continue indefinitely, e.g. a walk cycle or standing idle cycle. While the animation is playing, the user can adjust motion parameters. For the current parametric motion, e.g. when the virtual character is performing a walk/jog motion, the user can speed up or slow down the character, which will immediately change the motion.

WebGL Renderer

The resources of the client device are not known prior to rendering. To ensure that the application can run on devices with different capabilities, only the minimum amount of graphics resources are allocated, e.g. data buffers and textures. Mesh and texture data are stored in client memory and the WebGL buffers are updated as required. The temporally consistent geometry representation allow for only the vertex positions and texture buffers to be updated on a frame-to-frame basis, as the mesh connectivity and UV coordinates remain constant over all frames. An overview of the client memory resources is shown in Figure 6.5.

To increase realism, the rendering engine can handle shadows in the virtual environment. Basic shadowing is achieved by positioning a virtual light source in the scene and rendering a depth map from the light source viewpoint. WebGL currently does not support access to the depth buffer, so depth values are packed/unpacked into a 24bit representation. When rendering the floor plane or scene objects, each vertex is projected into the depth buffer. If a vertex falls into a region occluded by the characters depth map, a shadow effect is applied. This is averaged over a small window to enable realistic shadows with soft edges.
6.3 System Overview

Fig. 6.4 Flow chart of character animation engine showing how motion parameters are used to query the motion graph and motion database based on user input.

Fig. 6.5 Memory arrangement of WebGL renderer.
6.4 Results

In order to evaluate the framework, three 4D video datasets were processed using the described approach. An overview of each dataset processed with this framework is shown in Table 6.1 and a description of each character scenario is given.

The CAE and WebGL renderer were implemented in JavaScript. All tests were performed on a Dell Optiplex 9010 desktop computer running Ubuntu 12.04 with an Intel i7 processor, 8GB of RAM and a Nvidia GeForce FX 640 graphics card. The WebGL application was tested in both Mozilla Firefox (version 35.0) and Google Chrome (version 40.0).

Character Dan

A male character in a red and black sweater, dark jeans and brown shoes was captured performing typical game character motions, e.g. idle, walk, run, and horizontal and vertical jumps [23]. In this demonstration, the user can interactively control the viewpoint, the state of the character using HTML buttons, and motion parameters when in a parametric motion node, e.g. the speed of walk/run, the height of vertical jumps and the length of horizontal jumps. This character requires 39 MB of mesh and texture data. It is made up of eight motions with a total of 254 frames. Once loaded, it requires 220 MB of RAM. Examples of this character are shown in Figure 6.6.

Fig. 6.6 WebGL Character Animation Engine Dan Example. The current motion of the character is controlled using HTML buttons below the viewer. Parametric nodes are controlled using assigned keys and the viewpoint is selected using either the mouse or keyboard controls.
6.4 Results

Character Roxanne

A female character in a green top, camouflage shorts, and brown boots was captured performing typical game character motions including walk, run, tense, hit and stagger. This data is part of the SurfCap dataset [94]. In this demonstration, the user can interactively control the viewpoint, the state of the character, and motion parameters. The complete character consists of 10 motions, requires 51 MB of data to be transferred, and once loaded requires 370 MB of RAM. Examples of this character are shown in Figure 6.7.

![Fig. 6.7 WebGL character animation engine Roxanne example. The character and viewpoint can be interactively controlled by the user. Parametric nodes (e.g. speed of walk/run) are controlled using the keyboard. This example consists of six motions including stand, walk, run, stagger, tense and hit.]

Ballet Character

A female ballet dancer dressed in a black leotard was captured performing nine short (5-10 second) dance segments, starting and ending in a neutral pose. This dataset was captured as part of the RE@CT project and reconstruction was performed by project partners. In this example, high resolution geometry of the character’s face was captured and reconstructed separately [10]. This high resolution face model was then fused to a lower resolution body template and temporal alignment performed [4]. In this demonstration, the user can reorder the small dance segments to create a unique dance. Example frames are shown in Figure 6.8. This is the largest demonstration consisting of 3301 frames, requiring 305 MB of data to be transferred and using 1.7 GB of RAM. To make this demonstration practical, a single texture map was selected and applied to all frames.
Fig. 6.8 WebGL Character Animation Engine Ballet Example. Users can create a unique dance sequence by concatenating short dance sequences together. This demonstration consists of 3301 frames.

Table 6.1 Overview of datasets and storage requirements of captured and reconstructed data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Frames (Motions)</th>
<th>Processed Data (MB)</th>
<th>Runtime</th>
<th>RAM(MB)</th>
<th>Load Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>254 (8)</td>
<td>12</td>
<td>27</td>
<td>39</td>
<td>220</td>
</tr>
<tr>
<td>Roxanne</td>
<td>432 (10)</td>
<td>20</td>
<td>31</td>
<td>51</td>
<td>350</td>
</tr>
<tr>
<td>Ballet</td>
<td>3301 (9)</td>
<td>304</td>
<td>1*</td>
<td>305</td>
<td>1700</td>
</tr>
</tbody>
</table>

* Loading times are dependent of internet connection speed which can be affected by many factors. * Due to the number of frames, a single texture map was applied to all frames.

In all cases, the rendering frame rate ranged from 30 to 70 frames per second (FPS). It was observed that Mozilla Firefox was consistently able to achieve a higher FPS than Google Chrome in all demonstrations. Loading time is dependent on the Internet connection speed which can be affected by many different factors. Dan and Roxanne demonstrations required between 30 and 60 seconds from page request to playing. However, the Ballet example took several minutes to load and in its current form would probably be unsuitable for general use. Examples of the user interfaces are shown in Figure 6.9.
Fig. 6.9 Example of user interface for (a) Roxanne and (b) Ballet character scenarios.
6.5 Conclusions

This chapter has presented a framework for delivering 4D characters to a WebGL enabled web browser. In contrast to previous work, further offline processing was performed that allowed the 4D characters to be interactively controlled in the client side browser application. This was achieved through the development of a JavaScript-based WebGL character animation and rendering engine. Three character demonstrations based on this framework were presented with each character expressing a different set of motions. The motion of a typical game character can be represented in a few hundred frames which equates to between 40-50MB of data. However, the data size was still a limiting factor and therefore the work presented in this chapter would greatly benefit from further research and development particularly focused on geometry and texture compression algorithms as well as switching from an image based to video based storage and transfer of textures. A method that has been employed in other work is to compress both texture and geometry using the H.264 video codec [26]. The H.264 format [118] allows binary data to be encoded alongside the video stream in the network abstraction layer. This has been used to compress texture and geometry into a streamable video format. This could be applied to the work presented in this chapter in the future.
Chapter 7

Conclusions and Future Work

7.1 Introduction

In this section, conclusions are drawn from each of the contributions, their limitations are discussed, and avenues for future research are considered.

7.2 Multiple Layer Texture Representation

A major limitation of multiple camera capture is the inherently large data size. The first contribution of this thesis, presented in Chapter 3, was a method to reduce the storage requirements of multiple view video while preserving the view-dependent and dynamic changes in surface appearance. This was achieved using a temporally consistent mesh representation and mapping the subject’s texture into the UV domain. All existing approaches either generated a single texture map applied to all frames [56], or a texture map per frame [56, 106, 122]. None were able to preserve both the view-dependent and dynamic appearance of a subject. To overcome this, each multiple view frame was decomposed into a set of texture maps ordered by surface visibility. Visibility was determined by depth testing each polygon as well as measuring its directness to the capture cameras based on the polygon normal. This enabled both compact storage and efficient view dependent rendering. Efficient rendering was made possible as the representation precomputed both visibility testing as well as projective texturing, removing the need for offscreen rendering and all additional buffers. The rendering pipeline was shown to be lightweight enough to run in a web browser and on an Amazon
Fire tablet. This approach was quantitatively compared to a state-of-the-art free-viewpoint video renderer [95] and was shown to produce almost identical results, as measured by the structural similarity index measure, with up to a >90% storage reduction.

An observation from the results was that assigning cameras to layers solely based on normals resulted in the camera assignment flickering over time and created an irregular assignment over the mesh surfaces. The contribution of Chapter 4 resolved this by solving the assignment in a MRF framework allowing both spatial and temporal information to be leveraged to optimally assign cameras to polygons in the layer sequences. This increased the storage reduction by promoting spatial and temporal redundancies in the layer sequences. It was also noted that the texture representation suffered from the same artefacts as in traditional view-dependent rendering, e.g. blurring and ghosting. Another contribution of this chapter was an optical flow based, multiple camera alignment method which resolved these artefacts and resulted in clearer and sharper textures. A limitation of this alignment method is that it would be unable to cope with large errors in either geometry or camera calibration. These two contributions lead to the definition of an optimal representation of 4D video and an ideal set of properties were outlined in Section 4.3.

Animation may require non-sequential access of both texture and geometry information. Whilst video compression achieves high storage reductions, it is difficult to randomly access a single frame. This is a limitation of the work presented in Chapter 3 and 4 as it does not easily allow for random access of the data. This is not a limitation of the MLTR, but due to its storage in a video stream. An alternative approach that could be explored in future work may involve using a dimensionality reduction technique such as principal component analysis (PCA). An initial experiment, shown in Figure 7.1, demonstrates this idea. Two hundred texture maps from the Face dataset, a selection are shown in Figure 7.1a, were compressed using PCA with the algorithm set to capture 90% of the variation in the input. This resulted in a mean texture, 22 principal components, and a weighting to reconstruct the input at any frame, shown in Figure 7.1b.

This representation required approximately 14 MB of storage compared to 170 MB for the original texture map sequence. This was implemented using the OpenCV 2.4.11 implementation of PCA. The mean texture and the principal components could then be tiled into a single texture that would represent the entire sequence. At render time, only a set of weightings would be sent to the renderer. This would give random access to all the texture maps in all the input frames. However, it may increase the computational complexity of rendering through an increased number of texture lookups. Improved temporal alignment would result
in less drift in the UV domain and could potentially result in fewer principal components being required.

Fig. 7.1 Texture Map Storage using Principal Component Analysis (PCA). (a) Randomly chosen textures taken from Face dataset. (b) Mean texture and first three principal components (left to right). (c) Reconstructed textures for the randomly chosen frames. (d) Heat map between pixel intensities of input textures against PCA reconstructed textures, small errors occur in the mouth and eye regions (best viewed in digital version).
A simple extension to the MLTR could be encoding the camera-to-polygon assignment as a binary stream within the video container, as was previously done with geometry [26]. The first frame could contain the complete polygon-to-camera assignment then future frames could be based on the difference to the previous frame. This would allow the STO methods to exploit the reduced number of polygon changes over sequence to reduce the amount of data required to be transmitted. Further research could follow many different paths including: using the MLTR as a basis for relighting, estimation of material properties and material editing, improvement of temporal alignment using cues from the UV domain, or computing super resolution texture maps.

7.3 Spatio-Temporal Shape Optimisation

Chapter 5 presented two shape refinement techniques which adapted a temporally consistent mesh representation to better reflect the shape of the subject and facilitate high-quality view dependent rendering. This is necessary as the combination of multiple processes on the mesh structure result in a loss of surface details. The first approach used features from other reconstructions results, visual hull and multiple view stereo, to refine the shape of the temporally consistent geometry. Rims were extracted using a min-cut/max-flow formulation from the visual hull which represented points on the mesh which intersected with silhouette contours. These were prioritised in the framework and where rims could not be found, the framework traded off the visual hull and stereo surfaces. This resulted in surface details more consistent with silhouette observations. However, view dependent rendering clearly showed texture artefacts resulting from errors in camera calibration.

To overcome calibration errors, a methods was proposed based on bundle adjustment. The bundle adjustment approach identified dense surface correspondence between wide baseline camera views using an optical flow based method. These correspondences were used to successfully reduce the reprojection error by jointly adapting shape and camera calibration parameters. This resulted in view dependent rendering with reduced texture artefacts, but had little affect on the overall shape of the subject. In contrast to the previous approach, this methods did not use any prior information, e.g. visual hull or multiple view stereo reconstruction, and only relied on correspondence derived through optical flow from projectively texturing the surface.
Neither of the proposed approaches was able to completely solve the problem. The non-rigid ICP approach adapted the shape to match the silhouette observations at the cost of rendering quality. This was due to errors existing error in the camera calibration. The bundle adjustment approach made small changes to the shape but effectively reduced reprojection errors caused by camera calibration. The bundle adjustment approach used only correspondence derived from optical flow between wide baseline cameras, whereas the non-rigid ICP approach identified correspondences between two other reconstructions and deformed the temporally consistent representation to match. A simple next step that could be explored is to combine these methods and alternate the optimisation between the methods in an iterative fashion.

7.4 WebGL-based Delivery of 4D Characters

Chapter 6 presented the first WebGL-based character animation engine to use 4D video as input. Previous work to deliver free-viewpoint video to a WebGL-based renderer was limited to replay and required the captured frames to be down sampled which significantly reduced the visual quality [16]. In contrast, the work presented in Chapter 6, applied further offline processing steps to compute a parametric motion graph and allowed the user to interactively control the state of the virtual subject. Although parametric motion graphs had been proposed previously [22], they had never been applied in a WebGL-based environment.

A limitation of this work is that the motions which can be reproduced are limited to those which were captured or a linear combination between related pairs. Casas et al. [22] were also able to compute transitions on the fly, whereas this work settled with precomputed transitions due to the added computational complexity. This can result in some latency between state change requests.

Texture blending between motion pairs is handled simply by switching to the closet motion. This could be done as a linear blend between the textures, but would result in ghosting artefacts. Both online and offline solutions to reduce texture artefacts have been proposed [21, 23, 35]. An online solution is 4D video textures [23] which used online optical flow in rendered screen domain from the virtual camera viewpoint. This is computationally expensive and outside of the capabilities of WebGL at present. An offline solution was presented in 4D model flow [21] that pre-computes alignment and stores correspondence in a UV domain. During rendering, the textures are interpolated based on the correspondence to
Conclusions and Future Work

enable efficient real-time rendering with a reduction of artefacts. This offline method could offer a viable solution to the problem, but would add to the data transfer overheads.

Temporally consistent geometry and a single texture map per frame or using the MLTR are effective at reducing the required storage of 4D video. However, data size is still problematic and results in increased loading times. This work would benefit from further research in the area of geometry and appearance compression as presented in Section 7.2 using a PCA texture compression scheme. Future work could also involve further research and development of the JavaScript-based character animation engine and to allow for multiple characters in a real game scenario. Efficiently handling character-character and character-object interaction would also be a key challenge to overcome.

7.5 Conclusion

Human performance capture has fascinated scientists, artists and engineers for over a century. Over the last decade, markerless performance capture technology has emerged to make up for the shortcomings of traditional animation techniques. Markerless performance capture has the potential to revolutionise digital content creation in the creative industries offering greater realism. The technology potentially has far reaching applications outside of entertainment in areas such as: medical imaging, diagnosis and rehabilitation, realistic training and simulation environments in VR/AR. This technology has now matured and is beginning to make its way into mainstream productions. This thesis has introduced methods to efficiently represent the visual appearance of a 4D video character, improve for shape reconstruction in wide baseline camera setups by combining the strengths of different reconstruction techniques and demonstrated that the data can be used to deliver interactive content directly to a standard web browser.
References


