UNIVERSITY OF CALIFORNIA,
IRVINE

3D Modeling From Images and Video Streams

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Miguel Sainz

Dissertation Committee:
Professor Nader Bagherzadeh
Professor Glenn Healey
Professor Falko Kuester
Professor Gopi Meenakshisundaram
Professor Renato Pajarola
Professor Antonio Susin

2003

The dissertation of Miguel Sainz
is approved and is acceptable in quality
and form for publication on microfilm:

_____

_____

_____

_____

_____

_____
Committee Chair

University of California, Irvine
2003

DEDICATION

To my parents and friends for their support.
To Yak.
To my lovely wife for being there.

TABLE OF CONTENTS

Page

LIST OF FIGURES ix

LIST OF TABLES xii

ACKNOWLEDGEMENTS xiii

CURRICULUM VITAE xiv

ABSTRACT OF THE DISSERTATION xv

CHAPTER 1: OVERVIEW 1

1 Introduction 1

2 Proposed solution 5

2.1 Data acquisition 6

2.2 Calibration 7

2.3 Scene Reconstruction 7

2.4 Modeling 8

2.5 Rendering 9

CHAPTER 2: CAMERA CALIBRATION 11

1 Introduction 11

2 Structure from motion 12

2.1 A note on the correspondence problem 15

2.2 Occlusions 17

2.3 Critical motions 17

2.4 SFM Techniques 18

3 Proposed calibration solution 22

4 Calibration without occlusions 23

4.1 Projective Reconstruction 26
4.2 Metric Upgrade 31
4.2.1 Closed form solution for the autocalibration equations 36
4.3 Refinement of the metric solution 38
5 Calibration with occlusions and outliers 44
5.1 Sequence division 45
5.1.1 2D homography determination 47
5.2 3D Inlier determination 50
5.2.1 Affine reconstruction 51
5.3 Subsequence Calibration 52
5.4 Sequence merging 53
5.5 Post-processing 55
6 Complete calibration algorithm 56
7 Conclusion 58
CHAPTER 3: SCENE RECONSTRUCTION 60
1 Introduction 60
2 Single and Multi-view Stereo vision 61
3 Volumetric Reconstruction 64
3.1 Volumetric intersection 64
3.2 Voxel Coloring 67
3.3 Surface Reconstruction 69
4 Proposed solution 70
4.1 Space carving theory 71
4.2 Our approach 79
4.2.1 Occlusion map generation 81
4.2.2 Determination of surface voxels in slice 84
4.2.3 Footprint calculation 85

4.2.4 Color consistency criterion 87
4.2.5 Background segmentation 91

5 Conclusion 93

CHAPTER 4: MODELING 94

1 Introduction 94

2 Model representations 95
2.1 Image based models 95
2.2 Point based models 97
2.3 Geometry based models 98
2.3.1 Surface reconstruction from 3D spatial point data 98
2.3.2 Surface reconstruction from 3D volume data 100

3 Model preprocessing 101
3.1 Point data extraction 102
3.2 Surface data extraction 103

4 Proposed model representations 108
4.1 Object Splat 110
4.1.1 Multiresolution hierarchy 111
4.1.2 Splat size calculation 112
4.1.3 Splat-size determination 113
4.2 DMesh Objects 116
4.2.1 Quadtree Triangulation 117
4.2.2 Construction of a depthmesh 119
4.2.3 Depth-mesh segmentation 123
4.2.4 Complete model description 126
4.3 MTMesh 127
4.3.1 Active vertices 128
4.3.2 Quality factors 130
4.3.3 Data encoding 131

4.4 Standard mesh model 132

5 Conclusion 134

CHAPTER 5: RENDERING 135

1 Introduction 135

2 Rendering approaches 136

2.1 Image based rendering 136

2.2 Point based rendering 138

2.3 Geometry based rendering 139

2.4 Hybrid approaches 141

3 Rendering approaches 143

3.1 Point based rendering 143

3.1.1 LOD selection 144

3.1.2 Visibility splatting 144

3.1.3 Normalization 150

3.2 DMesh rendering 151

3.2.1 Depth-mesh selection and triangulation 154

3.2.2 Rendering pass 156

3.2.3 Compositing and normalization 158

3.3 Multitextured Mesh Rendering 159

3.3.1 Reference view selection 161

3.3.2 Render pass 161

3.4 Image Normalization 162

3.4.1 Software 163

3.4.2 Texture shaders 164

3.4.3 Fragment programs 166

3.5 Model re-lighting 168

4 Conclusion 170

CHAPTER 6: RESULTS AND CONCLUSIONS 172

1 Introduction 172

2 Pipeline walkthrough 172

3 Persepolis 179

4 Other examples 182

5 Conclusions and future work 186

REFERENCES 192

LIST OF FIGURES

		Page
Figure 1.1:	Proposed image based modeling and rendering pipeline.	6
Figure 2.1:	Pinhole camera model.	14
Figure 2.2:	Example of sequence calibration without occlusions.	25
Figure 2.3:	Rank-4 approximation graphical interpretation.	30
Figure 2.4:	Sparse structure of the Jacobian and Normal Equations.	42
Figure 2.5:	Measurement pattern of a sequence with occlusions.	46
Figure 2.6:	Example of a sequential merging.	55
Figure 2.7:	Example of sequence calibration.	58
Figure 2.8:	Example of sequence calibration with occlusions.	59
Figure 3.1:	The inferred visual hull.	65
Figure 3.2:	Difference between visual hull and voxel carving.	75
Figure 3.3:	Illustration of the carving non-photo subset consistency Lemma.	76
Figure 3.4:	Standard configuration for a plane sweep carving algorithm.	79
Figure 3.5:	Shadow projection of previous consistent voxels.	82
Figure 3.6:	Occlusion map generation.	84
Figure 3.7:	Two ways of calculating the footprint of the voxels.	86
Figure 3.8:	Two different ways of calculating the visual hull.	92
Figure 4.1:	Cuberille and Marching Cubes patterns.	101
Figure 4.2:	Ordered iterations over the voxel data structure.	104
Figure 4.3:	Mesh cube and surface mesh generation.	105
Figure 4.4:	Laplacian mesh relaxation.	106
Figure 4.5:	Displacement calculation during relaxation.	107
Figure 4.6:	Reconstructed mesh example.	109
Figure 4.7:	Elliptical surface elements covering a smooth curved 3D surface.	111
Figure 4.8:	Point-octree generation data flow.	112
Figure 4.9:	Projection of points onto tangent plane.	114
Figure 4.10:	Ellipse coverage determination.	115

Figure 4.11: Splat based rendering example. 116
Figure 4.12: Recursive quadtree subdivision and triangulation. 118
Figure 4.13: Cracks in unrestricted quadtree subdivision. 118
Figure 4.14: Grid of depth-image pixels. 119
Figure 4.15: Adaptively triangulated terrain using the DMesh approach. 122
Figure 4.16: Rubber sheet artifacts. 123
Figure 4.17: Rubber sheet triangles in the depth-mesh. 124
Figure 4.18: Segmentation flags. 125
Figure 4.19: Propagation of segmentation. 126
Figure 4.20: Normal orientation test. 129
Figure 4.21: Quality map of for a given reference view. 131
Figure 5.1: Point rendering algorithm block diagram. 145
Figure 5.2: Generic triangle modeling a surfel. 146
Figure 5.3: Distribution of surfels. 146
Figure 5.4: Blending kernels. 148
Figure 5.5: Mapping of blending kernel as α-texture onto a generic triangle. 149
Figure 5.6: Splatted model of a reconstructed cup. 152
Figure 5.7: Depth-mesh rendering pipeline. 155
Figure 5.8: Selection of reference views. 156
Figure 5.9: DMesh rendering pipeline example. 159
Figure 5.10: MTMesh rendering pipeline. 160
Figure 5.11: MTMesh rendering example. 162
Figure 5.12: Normalization based in OpenGL texture shaders. 166
Figure 5.13: Example of re-lighting of cup model. 170
Figure 6.1: Input data for monster dataset. 174
Figure 6.2: Calibrated sequence and reconstructed scene for the monster dataset. 177
Figure 6.3: Rendering examples. 178
Figure 6.4: Reconstruction of *two-headed-horse* dataset. 181
Figure 6.5: *head* dataset. 184
Figure 6.6: Example of space carving based reconstruction. 185

Figure 6.7: Some of other reconstructed objects. 185

LIST OF TABLES

		Page
Table 6.1:	Calibration statistics for monster dataset.	173
Table 6.2:	Scene reconstruction for the monster dataset.	175
Table 6.3:	Computation time of the different model representations.	176
Table 6.4:	Average frames per second for the different representations.	179
Table 6.5:	Calibration results for the *two-headed-horse* dataset.	182
Table 6.6:	Calibration results for the *head* dataset.	183

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Professor Nader Bagherzadeh and my co-advisor Professor Antonio Susin who have helped me with their resources and guidance to complete this work. Without their persistence this dissertation would not have been possible.

I would like to thank my committee members, Professors Healey, Kuester, Gopi and Pajarola whose comments and supervision have contribute to guarantee the highest level of quality of this work.

Additionally a strong thank you to Professor Pajarola, who has helped me in several aspects of this research. He and his enthusiasm for Computer Graphics have provided me a great opportunity to learn, introducing me to several interesting ideas that surely will impact my future in research.

A big thank you to Professor Rangel and Pete Balsells for the excellent Fellowship program that started this project.

Last but not least, I would like to thank Dr. Jose Najera for providing me wise advice of my career as well as for being there when the big decisions were made.

Financial support for this research was provided by the University of California, Irvine, NSF Grants (CCR-9877171, CCR-0083080), by the Comissio Interdepartamental de Recerca i Innovacio Tecnologica (CIRIT) Gaspar the Portola Collaboration Grants 00-01, 01-02, 02-03 and the Balsells Fellowship.

CURRICULUM VITAE

Miguel Sainz

Education

1999 - 2003	**University of California, Irvine.** Doctor of Philosophy in Computer Engineering. The Henry Samueli School of Engineering, Dept. of Electrical Engineering and Computer Science.
1994 - 1996	**Cybernetics Institute (IC), Barcelona, Spain**. Master of Science degree in Electrical and Electronics Engineering.
1988 - 1994	**Polytechnical University of Catalonia (UPC), Barcelona, Spain**. Bachelor degree in Electrical Engineering.

Appointments

07/03 - 07/04	**Faculty Researcher and Lecturer**. School of Information and Computer Science. University of California, Irvine.
09/99 - 06/03	**Graduate Research Assistant**. Electrical and Computer Engineering, The Henry Samueli School of Engineering, Univ. of California, Irvine.
12/98 - 07/99	**Java Instructor**, Trainning Services Department of Sun Microsystems, Barcelona, Spain
09/98 - 01/99	**Lecturer**. Multimedia engineering and computer graphics. La Salle School of Engineering, Ramon Llull University, Spain
07/96 - 09/98	**Research Engineer**. Technical University of Catalonia (UPC), Computer Graphics Section and the Robotics and Informatics Institute, Barcelona, Spain
01/93 - 06/96	**Research Student**, Instituto de Cibernética (IC), Barcelona, Spain.

Awards

07/03 - 07/04	**UC Faculty Fellowship Award**. School of Information and Computer Science. University of California, Irvine.
04/03 - 06/03	**Dissertation Fellowship Award**. Department Electrical and Computer Engineering, The Henry Samueli School of Engineering, University of California, Irvine.
09/99 - 09/00	**Balsells Fellowship**, Department Electrical and Computer Engineering, The Henry Samueli School of Engineering, University of California, Irvine, CA
01/94-07/96	**IC Undergraduate Fellowship**, Technical University of Catalonia (UPC), Robotics and Informatics Institute, Barcelona, Spain

ABSTRACT OF THE DISSERTATION

3D Modeling From Images and Video Streams
By
Miguel Sainz
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Irvine, 2003
Professor Nader Bagherzadeh, Chair

Image Based Modeling and Rendering (IBMR) is a fairly new field of research and experimentation with the goal of reducing the complexity of use and the execution time of the modeling of photorealistic 3D graphics. These new techniques can be used to develop methods and tools for acquiring and modeling objects, using pictures and/or video sequences as source data. In this work we present a novel pipeline to reconstruct full 3D photorealistic model reconstruction of real world objects from a set of images. The reconstruction process begins with the acquisition of an unrestricted video walkthrough inside or around the scene to be modeled. Robust digital image processing and computer vision techniques allow the use of low cost acquisition systems such as photo or video cameras. After acquiring and digitizing the frames, a tracking algorithm extracts significant 2D information of the video sequence. Then, a calibration process automatically calculates the 3D position of the tracked features, the 3D position and orientation of cameras and their internal parameters, such as the focal length. We have successfully developed novel algorithms that perform these calculations robust and computationally efficiently. We perform a spatial carving operation on a piece of virtual material containing the scene using a voxel carving methodology highly optimized to use the computation power of the actual videocards. Once this carving is finished, a

modeling process obtains a representation suitable to be rendered. We propose three different representations which are a trade-off of different factors with the goal of achieving fast rendering: the final application might require from the 3D model representation several characteristics such as support for level of details (LODs), collision detection, interactive illumination and photorealistic quality. The proposed pipeline targets applications in areas such as Virtual Reality, Computer Gaming and Special Effects. All these environments require high quality 3D models for increased realism. Our approach allows obtaining such data directly from real world objects using images in a fast and automatic way.

CHAPTER 1 OVERVIEW

1 INTRODUCTION

Since the very beginning of computer technology, the possibility of reproducing the real world for simulation purposes has been a primary goal of researchers. Computer technology has evolved tremendously in past years, and one of the features that has been improved the most, even challenging Moore's Law, is the computer graphics capabilities. Showing computer simulation results by visualizing a complex 3D scene in real time is now possible thanks to powerful graphics hardware. Fields such as medical imaging, training simulators, robotic guidance, etc. have seen great improvements in their development because of the graphics revolution. Other new fields such as virtual reality, computer games, have grown from this new technology and they are now ones of the most active business and research fields. Furthermore, entertainment industry in

general and film and broadcasting in particular use extensively the power of graphics to delight the audiences.

In parallel to the growth of computer graphics technology an important demand for more complex and realistic 3D content has emerged. However, even though the supporting tools for 3D model creation are more powerful (but also more expensive and complex to use), obtaining realistic models is still difficult and time consuming.

Image Based Modeling and Rendering (IBMR) is a new field of research and experimentation with the goal of reducing the task complexity and duration of the modeling process of photorealistic 3D graphics. These new techniques can be used to develop tools for acquiring and modeling objects using pictures and/or video sequences as input data. However, the existing IBMR techniques are still in their early stages and require painful calibration procedures and a large amount of user interaction to verify and guide the different algorithms during the reconstruction process.

There is an increasing demand for flexibility in IBMR tools, to allow their use under less restrictive conditions taking benefit of automatic calibration procedures, and minimizing user guidance. Furthermore, some of the existing systems are built around specialized hardware (e.g. laser range finders [Delt] or stereo rigs) increasing the cost and difficulting their use in unconstrained environments. New commercial applications start to use low cost acquisition systems such as standard digital photo and video cameras due to the increasing quality of the images they provide. In the past few years, many techniques have been developed to acquire 3D models of real objects, and several of them do not require more than a camera and a PC computer.

In a more formal way, 3D reconstruction techniques can be divided in two main groups based on the type of sensors used. On one hand there are the *active* techniques, which control the lighting of the scene (e.g. structured light, laser illumination…) simplifying the problem but restricting the applicability due to the sensor requirements, environment suitability and high cost. Some examples of active techniques are the grid projection approach, which is able to extract dynamic textured 3D shapes (this technique is commercially available in [Snap]) and the shadow-based approach proposed by Bouguet and Perona [BoPe98].

On the other hand there are the *passive* techniques that impose less requirements in the sensing devices (video cameras, and photo cameras), but are computationally more expensive and dependent on the structure of the scene. One of the main differences between the available solutions are the calibration process and the amount of required user interaction. The presented work and literature revision are focused on this type of techniques.

The original methodology to reconstruct objects from images is called photogrammetry [Slam80] which deals with the extraction of high accurate measurements from pictures. The calibration must be very precise and the process is completely manual: the user constructs the model from what he sees and the measures he takes from the images. Therefore the acquisition of detailed models is very time consuming. Some software tools to assist the user are commercially available (e.g. [Phot]).

Researchers in the computer vision community have tried to both reduce the requirements for calibration and augment the automation of the reconstruction with the ultimate goal of being able to reconstruct a scene or object by freely moving a camera around it.

An early approach, proposed by Tomasi and Kanade [ToKa92], was used for robotic vision. They used an affine factorization method to extract 3D information from uncalibrated image sequences. An important restriction of this system was the assumption of orthographic projection, but latterly was extended to a paraperspective projection [PoKa93], which is a linearized approach of the perspective projection. However the results do not provide the enough visual quality nor a complete reconstruction of the objects in the scene.

A set of techniques have been proposed to reconstruct 3D surface models from a sequence of images taken with off-the-shelf cameras. The systems use perspective cameras and do not require prior known models. The most successful methods are [SzTo91], [AzHP93], [Poll99] and [HaKa00] among others.

Other approach, the project *Facade* by Debevec *et al.* [DeTM96], uses approximated 3D models created by the user and interactively refine the model based on images. The advantage is that less images are required. However a preliminary model must be built and the geometry should not be too complex to avoid build the complete model manually. This approach has been used with good results in some commercial products such as [Cano] and [Imag].

In summary, the IBMR techniques are in their early stages and, most of these algorithms generally require extensive manual preprocessing in order to obtain accurately calibrated images and realistic geometric approximations of the target objects. Moreover, most of these algorithms heavily rely on user interaction for camera calibration and image registration or require expensive equipment such as calibrated gantries and 3D scanners.

The proposed research is focused on developing new techniques to achieve new goals that will get us closer to the ultimate goal of a full automated tool for 3D photorealistic model reconstruction of objects.

2 PROPOSED SOLUTION

The main goals of the proposed project is to develop a software tool, based on image based modeling techniques, that will allow the automatic reconstruction of physical objects with all their properties (shape, color and texture) properly recovered. Based on different representations of the reconstructed models, the appropriate rendering algorithms will be provided to maximize the utilization of the recovered data. Figure 1.1 illustrates the block diagram of the process for the reconstruction of 3D models from images:

In the following subsections more detail is given about the different stages involved in the proposed pipeline. The ultimate goal of such a tool will be to automate the complete process, however, as we will see throughout this document, there are many different problems involved that are still very difficult to solve, therefore, some steps still will require a certain amount of user interaction.

A more realistic set of goals regarding the proposed pipeline is (1) to minimize user input and (2) concentrate the user interactions in a single location of the pipeline, preferably at the beginning, so once the system is properly configured the rest of the pipeline can be considered as a batch process.

2.1 Data acquisition

The input to the proposed system is a video sequence or set of images taken with an off-the-shelf digital camera or camcorder, by moving it around the physical object that is going to be reconstructed. The reconstruction algorithm should guarantee that the reconstructed model will match closely the original images when seen from each of the reference image viewpoints. The more images are provided, and more important, the more coverage of the object's surface, the better the reconstruction will be.

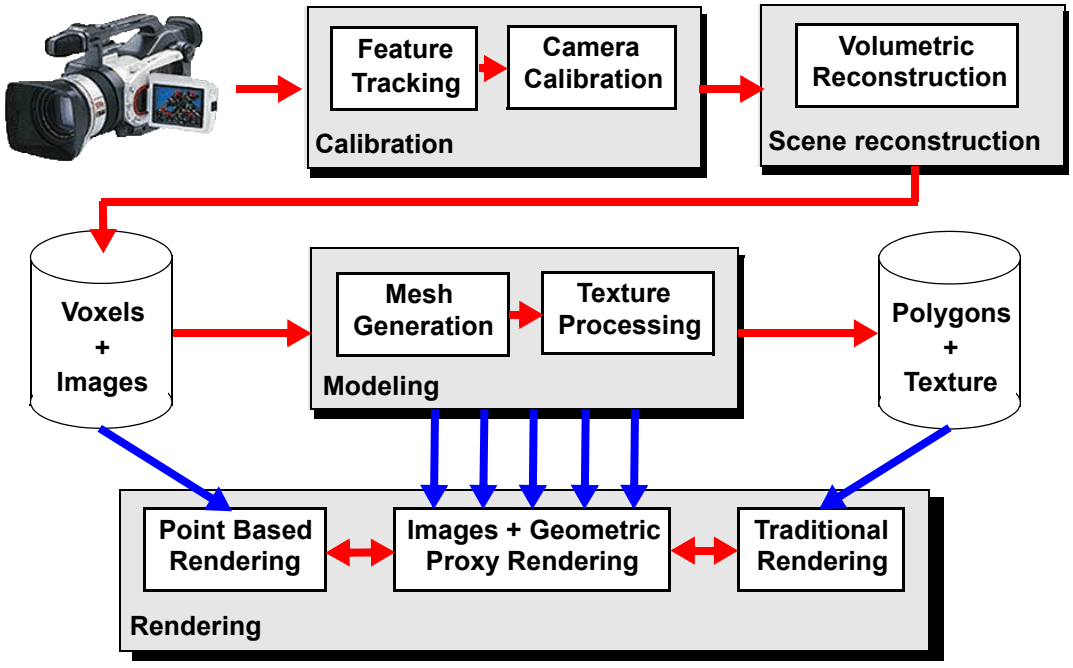


Figure 1.1: Proposed image based modeling and rendering pipeline.

2.2 Calibration

The goal is to reconstruct the 3D structure of the scene and the motion of the camera analyzing how 2D tracked points move in the image sequence. This problem is well known in the community as the *Structure From Motion* problem (SFM).

The first task is to identify significant 2D features in each of the reference views and establish the correspondences between them, that is, for a specific feature in one of the views, where does it appears in the rest of the views. The feature detection on each view can be performed using computer vision techniques for corner, line or curve detection. The correspondence determination can be done using tracking algorithms but it becomes a difficult problem when the number of images decreases and the pose variation between two consecutive frames is large.

The second task is to recover the camera parameters (intrinsic and extrinsic) as well as the structure of the scene (the 3D coordinates of the detected features). There are several approaches to solve this very difficult problem but so far none of them has achieved good and robust results. The different solutions are computationally expensive and noise sensitive and one of the goals is to obtain more reliable methods based on hybrid combination of linear techniques and non-linear bundle adjustment methods.

2.3 Scene Reconstruction

The goal is to extract 3D geometry of the target objects in the scene, based on given camera locations and their respective images. Different approaches such as photogrammetry, stereo vision, contour and/or shadow analysis techniques work with similar assumptions.

The visual hull [Laur94], including the implementation shown in [Lok01], is suitable for real-time volume reconstruction. Although the visual hull is a good approximation of the real object, local concave surface features on the target object cannot be recovered unless they are reflected in the silhouette of the blob. Recently a set of volumetric techniques based on spatial carving algorithms ([KuSe00], [BrDC01], [CuMS99]) was used to reconstruct complex object shapes with excellent results. The common characteristic for these approaches is that they carve a piece of voxelized virtual material that contains the object, similar to an artist sculpting a raw block of marble. The proposed method is inspired in the space carving theory [KuSe00] and addresses the extended computational cost, which is a common trade-off for all these carving methods.

2.4 Modeling

The modeling stage consists of determining the best representation for a model of a reconstructed object. There are different ways of modeling an object based on the input data, the rendering algorithm and the final use of the model.

The proposed model representations are:

- Single mesh. We proposed a mesh generation method based on a cuberille mesh representation with a feature-preserving relaxation step. Then two options are presented for the color information: a color per vertex dataset and the use of the original images as overlapping textures on the mesh surface.
- Multiple meshes. A polygonal textured mesh is generated for each of the reference views. We present a method to use the depth buffer resulting of rendering the

voxelized scene reconstruction as a source for an image aligned triangulation method.

- Colored oriented points. This is a very simple and obvious representation that does not require further processing after the scene reconstruction except determining a combination of the different contributing colors from the reference images for each of the voxels present on the object. Furthermore we present a level-of-detail (LOD) data structure for efficient rendering that holds the point information.

2.5 Rendering

The last stage of the proposed model reconstruction pipeline is to use the reconstructed models of physical objects in a realtime rendering pipeline with the best possible quality. Several approaches for rendering can be taken based on the available representation of the object, the interactivity degree of the renderer and the desired quality.

The proposed rendering systems for each of the model representations are:

- Single mesh / multiple textures. The proposed approach is a view dependent rendering algorithm that combines the significant reference images to texture a triangular mesh in real time.
- Single colored mesh. The rendering method is the most common one used in computer graphics. Additionally to the computation of a single mesh, a weighted combination of colors will be calculated for each of the vertices of the mesh in order to generate a vertex colored mesh that can be used in almost any 3D graphics pipeline.

- Multiple meshes / multiple textures. We present an algorithm that takes a set of view dependant selected non-overlapping meshes and blends the final result into a seamless image of the object.
- Point based objects. A point splat rendering algorithm with LOD capabilities is presented.

The present document is structured in a set of chapters, one for each stage of the pipeline. Therefore, Chapter 2 presents the literature review on camera calibration as well as the proposed solution. In Chapter 3, a solution to perform volumetric reconstructions from a set of calibrated images is presented. Chapter 4 discusses the different model representations and how to generate them efficiently. In Chapter 5 we present the rendering paradigms for the different model representations. Chapter 6 is a walkthrough the complete pipeline showing detailed results for different datasets. Finally, an additional chapter with some conclusions is presented.

CHAPTER 2 CAMERA CALIBRATION

1 INTRODUCTION

The first requirement in order to extract spatial information from a set of images of a scene is to have perfectly characterized the camera position and orientation as well as the internal parameters such as the focal length that were used during the acquisition process. When part or all of that information is missing a camera calibration process is required in order to recover it.

Although previous results on this area come from researchers in the computer vision field, recent interest on the problem has raised because of the straightforward applications in building 3D models for virtual reality, video conference or medical applications. Also, in the film and video production industry, there has been a proliferation of demand for computer-graphics based special effects consisting of the combination of 2D digital image sequences with 3D computer graphics that require a

perfect synchronization only obtained with a calibration procedure. The common denominator of all these applications is that they require an accurate 3D scene reconstruction from the 2D source images.

One easy way of performing such calibration process is to use some hardware devices or fiducial marks in a controlled environment or by using calibrated rigs to mount and move the camera/s around. However due to the implied cost and the lack of flexibility of such systems, we will focus our attention on the use of low-cost and readily available sensor devices such us digital still cameras or camcorders. Moreover we will assume that no prior knowledge of the camera nor its relative motion in the scene is known. Therefore, the calibration process is based entirely in measurements taken from the input images.

This section provides a review of the different approaches to solve the camera calibration problem based on image measurements. Also the final adopted solution is presented, emphasizing the different technical difficulties and the way they have been addressed.

2 STRUCTURE FROM MOTION

The *Structure From Motion* (SFM) refers to the problem of recovering the three-dimensional structure and motion of a scene from its two-dimensional projection onto the image plane of a moving camera. No information about the camera nor the scene is known a-priori and the only requirement regarding the scene is that it will be assumed to be rigid.

The SFM analysis is based in preprocessing the set of reference views to consistently extract and label 2D features in the scene. Such features consist of corners of objects, lines along edges or curves along contours. These features are detected on each image and then associated with their corresponding instantiations in the other images. The correspondence between points of different frames is assumed to be given in the general SFM.

Two dominant approaches to the SFM problem can be distinguished according to the number of points tracked along the image sequence. We can use a correspondence for a small set of points or for all the pixels in the sequence which are known as *sparse* or *dense* correspondence methods. The later methods are based on determining the optical flow between frames, which limits the baseline or distance between each reference image. Moreover, it is argued in [BrFA98] that the determination of the optical flow is an ill-posed problem due to inherent differences between 2-D motion field and intensity variations. It is reflected in [Chen00] that none of the optical flow based techniques produce low error and high density correspondences in all testing cases.

The presented work and the reviewed literature is focused in sparse correspondence methods such as the ones presented in [ChMe99], [MaHe00], [HaKa00], [HeBS99], [Trig96], [Poll99], [AzPe95] and [JeAP99]. A good review of projective reconstruction can be found in [Trig96] and for the euclidean reconstruction in [JeAP99] and [Fusi00].

The SFM problem can be stated more formally considering m uncalibrated views with n 3D points and the respective 2D measurements:

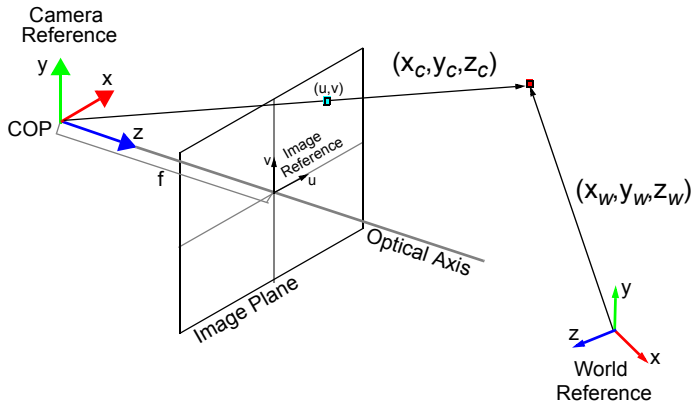


Figure 2.1: Pinhole camera model.

$$\begin{bmatrix} \lambda_{11} \cdot \begin{bmatrix} u_{11} \\ v_{11} \\ 1 \end{bmatrix} & \cdots & \lambda_{1n} \cdot \begin{bmatrix} u_{1n} \\ v_{1n} \\ 1 \end{bmatrix} \\ \cdots & \cdots & \cdots \\ \lambda_{m1} \cdot \begin{bmatrix} u_{11} \\ v_{11} \\ 1 \end{bmatrix} & \cdots & \lambda_{mn} \cdot \begin{bmatrix} u_{mn} \\ v_{mn} \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} K_1 \cdot [R_1 | T_1] \\ \cdots \\ K_m \cdot [R_m | T_m] \end{bmatrix} \cdot \begin{bmatrix} X_1 & \cdots & X_n \end{bmatrix} \qquad \text{(eq 2.1)}$$

which can be expressed in short as $W = PX$, where W is the *3m x n measurement* matrix, P is the *3m x 4 perspective* matrix and X is the *4 x n shape* matrix. λ_{ij} are the *projective depths*, the non-zero scale factors that relate a 3D point in camera coordinates and its perspective projection. X_i are the unknown homogenous world coordinates of the tracked features.

A commonly used camera model is the *pinhole* perspective camera (Figure 2.1) that consists in an image plane where the 3D features project with perspective rays along the center of projection (COP). The origin of coordinates of the camera frame is

traditionally located at the COP and the focal length f_i is the distance from the COP to the image plane along the principal axis, which aligned with the z axis of the frame.

A more complete parametrization of the intrinsic parameters of the camera includes the principal point (u_{0i},v_{0i}), the aspect ratio α_i, and the skew β_i. In addition there can be other nonlinear effects such as lens distortion, but they are neglected, specially for applications where high accuracy is not a requirement. The matrix notation for the camera parameters is the following

$$K_i = \begin{bmatrix} f_i & \beta_i & u_{0i} \\ 0 & \alpha_i f_i & v_{0i} \\ 0 & 0 & 1 \end{bmatrix}, R_i = \begin{bmatrix} i_i^T \\ j_i^T \\ k_i^T \end{bmatrix}, T_i = \begin{bmatrix} T_{xi} \\ T_{yi} \\ T_{zi} \end{bmatrix}, i = 1 \ldots m \qquad \text{(eq 2.2)}$$

Summarizing, the SFM problem consists of solving Equation 2.1 with the only knowledge of the pairs u,v. If no other information is added, the problem becomes non-linear, and as it turns out, is still a very difficult unsolved problem in computer vision. Moreover, in case that a solution can be computed, it will be correct up to a scale factor due to the depth-speed ambiguity, since it is impossible to determine whether a given image motion is caused by a nearby object with relative slow motion or a distant object with fast relative motion.

2.1 A note on the correspondence problem

The focus of this work is to solve the automatic reconstruction of objects from multiple reference views. During the camera autocalibration process it is required a set of corresponded features that will be assumed as given throughout the presented work,

although the detection and matching of features is a difficult computer vision problem that cannot be dismissed in practical implementations.

However, in order to perform validation experiments of the algorithms, a set semiautomatic tools have been used to obtain the measurements. Depending on the type of reference views used in the calibration process, we can classify these tools in the following groups:

- **Synthetic imagery**. This corresponds to the use of synthetic models and usually the environment is perfectly characterized. The autocalibration algorithms can be tested by calculating the 2D projection of the vertices of the objects to reconstruct. We have used this tools to evaluate the calibration sensitivity under noise in the input data.
- **Sparse set of reference views**. Some of the datasets have been obtained using a digital still camera and the pose variation of the objects between frames is large, making very difficult to automatically establish correspondences between the different frames. A semi-automatic tool has been used to detect features on each reference image and the correspondences are determined manually.
- **Video sequences.** For this type of input an adapted implementation of the KLT tracker [ShTo94] has been used to find features and track them along the frames. The tool does not filter outliers (detected points that do not belong to the object and distort the results) so a manual postprocessing, combined with an automatic statistical analysis during calibration, must be carried out in order to remove the outliers.

Overall, two point detection and tracking algorithms have been used, [ShTo94] and [SmBr97], and a robust outlier detection algorithm has been integrated based on RANSAC ([HaZi00]) in order to achieve a full automation of the calibration process.

2.2 Occlusions

Camera autocalibration is based in selecting correspondences between features, and most of the methods in the literature obtain a solution performing a global minimization on all the feature measurements simultaneously. When the pose of the objects change radically between reference images is very common that some of the tracked features are occluded and new ones appear. None of the most successful approaches propose a method that has a built-in support for occlusions. Nevertheless there are some solutions [FiZi98], [GCH+02] that implement a divide and conquer approach by splitting the sequence of images in time contiguous chunks where the occlusion problem can be ignored. Each of the fragments is solved and a final complete solution is obtained performing a bundle adjustment.

2.3 Critical motions

Not all the sets of reference views are suited for an autocalibration process because the extracted measurements present ambiguities and the motion or camera parameters are coupled giving a non unique solution. Some obvious cases are a pure rotation in the optical axis of the camera, pure planar translation perpendicular to the optical axis or pure zooming.

Several analysis have been conducted in [Trig97], [KaTr99], [Poll99] and Sturm in [Stur97] and [Stur02] provides a catalogue of critical motions based on the unknown calibration parameters.

To avoid falling into degenerate cases, it is recommendable to enforce as many constraints as possible and to perform a general enough global motion around the scene. According to [Poll99] by fixing all intrinsic parameters except the focal length, the critical set of motions is reduced to a pure rotation, where no reconstruction is available, a pure forward motion where focal length and translation are coupled, or hyperbolic or elliptic motion were an extra solution is generated.

2.4 SFM Techniques

Recently new approaches based on the idea of *stratification* have been introduced. The goal is to obtain an initial projective reconstruction which can be computed from the set of correspondences. Then, depending on the constraints and additional information available, an upgrade to affine or metric (Euclidean) calibration is calculated.

In order to obtain an initial projective reconstruction it is necessary to calculate a set of projective depths λ_{ij} in such a way that the measurement matrix W can be factorized in a motion matrix P and a shape matrix X. Furthermore, as stated in [Trig96], the matrix W has to be rank-4 if it is the matrix associated to a projection of a set of real points. Consequently, for points in general positions, a rank-4 factorization of the scaled matrix produces a projective reconstruction of the points.

Due to the unknown projective depths the solution can not be reached directly, but an iterative approximation can be constructed. There exist different approaches ([ChMe99], [MaHe00], [HaKa00], [HeBS99], [Trig96], [StTr96]) to construct an iterative algorithm that converges to a rank-4 decomposition of the measurement matrix.

In [Poll99] a different approach is suggested that extracts a projective reconstruction based on epipolar constraints that adds one new frame at a time, relating it to the previous frame. A local minimization is performed using an Extended Kalman Filter technique and, when all reference views have been added, a global bundle adjustment is performed using a Levenberg-Marquardt minimization. This method allows to handle occlusions properly although the convergence is largely dependent on an accurate initial solution obtained by determining the fundamental matrices between the frames, which is normally a constrained non-linear minimization problem that may incur numerical instability.

Once a projective reconstruction is available, the SFM problem is reduced to find the right projective transformation that upgrades the projective motion matrix and shape matrix to metric space

$$W = \hat{P} \cdot \hat{X} = \hat{P} \cdot H \cdot H^{-1} \cdot \hat{X} = P \cdot X \qquad \text{(eq 2.3)}$$

Almost all existing autocalibration algorithms are based on the invariant property of the absolute conic (or the equivalent projected absolute quadric) under rigid motion. Triggs presents in [Trig97] a method that uses an estimation of the absolute quadric to rectify the projective reconstruction to a scaled Euclidean one. An efficient nonlinear

optimization technique is used (quadratic programming), but a quasi-linear solution is also proposed. A limitation is that the camera intrinsic parameters are assumed constant.

Pollefeys in [Poll99] proposes a method with fixed intrinsic camera parameters except the focal length, that can vary between the reference images. An initial solution of the absolute quadric is calculated using a linear method, and then it is refined with a non-linear maximum likelihood approach.

An Euclidean reconstruction is a special form of projective reconstruction, related by a *4 x 4* collineation called *Projective Distortion Matrix* (PDM), that is represented as *H* in Equation 2.3. As shown in [Chen00] and [SaBS02], it can be proved that this PDM contains the elements of the absolute quadric which relates these approaches to the existing ones. However, by avoiding dealing with the absolute conic/quadric these two approaches are conceptually much simpler.

Kanade's group has been very prolific in developing reconstruction algorithms using linear factorization methods. The first approach solves the SFM problem for the orthographic camera case and is proposed in [ToKa92]. An SVD based factorization is performed to obtain an initial projective reconstruction and a set of linear orthographic constraints is imposed on the shape and motion matrices to obtain the PDM and recover the scene. In a later work [PoKa93], the camera model is upgraded to a paraperspective model, which is Taylor's first term approximation of a perspective camera. A similar methodology is applied to obtain the PDM and the reconstructed scene. Han and Kanade propose in [HaKa00] a full perspective reconstruction method based on the factorization approach and impose again a set of linear constraints on the PDM. The recovery of the

PDM is performed by solving a set of linear problems without requiring an initial solution. Three different sets of constraints are proposed based on the available knowledge of the camera intrinsic parameters. These sets of linear-based approaches are very attractive since they provide a closed solution without any initial guess. However it is still an approximation and it is not clear if there are any restrictions on the potential cases they can solve.

A completely different set of approaches of the structure from motion problem use the Extended Kalman Filter (EKF) (see [AlBW01]) as a non-linear estimator of the camera intrinsic and extrinsic parameters given the measurements. The different solutions attack the problem directly to the Euclidean perspective level in an iterative way without any stratification.

A first approach is introduced in [BrCe86] and [BrCC90] where they assume a single fixed camera and suppose the variation on the measurements corresponds to a rigid object with relative motion. A state vector is constructed by considering the position of a coordinate system centered in the object, its translation and angular velocities, their derivatives and the relative 3D coordinates of the points in the object reference since the points are constant in such a coordinate system.

A motion equation is described, and the rotation and translation, as well as the object points are calculated evaluating an Iterative Extended Kalman Filter (IEKF). The method does not handle occlusions, since all points must be visible in every view to define the current state vector. Moreover, the method converges when an initial close to correct solution is provided using the batch method of [BrCe86] for the first frames.

In [AzPe95], and republished later in [JeAP99], a similar method is presented using the EKF with a downsized state vector: translation and rotation of the object reference frame, focal length of the camera and distance of each point to the image plane. The method obtains good results even with a trivial initial solution. Although Kalman filter based methods are an interesting solution, there is usually no guarantee of convergence without and initial solution and occlusion handling is not supported.

3 PROPOSED CALIBRATION SOLUTION

In this section we present the proposed method [SaSB03] to calibrate and extract a set of key-frames from a video sequence that contain enough three dimensional information to be used as the reference views for an image based reconstruction algorithm. However, for such applications, a requirement to obtain a robust reconstruction is that the image sequence must show the object from different perspectives, therefore self-occlusions are constantly present increasing the difficulty of the problem.

We present an approach based on a divide and conquer strategy to fully calibrate a long sequence of images with a high degree of feature occlusions such as video sequences of objects on rotating platforms. The complete sequence is automatically divided into subsequences and, in each of them, a set of key-frames is selected and calibrated using an improved version of the algorithm presented in [HaKa00], recovering both camera parameters and structure of the scene. When the different subsequences have been successfully calibrated a merging process groups them into a

single set of cameras and reconstructed features of the scene. A final non-linear optimization is performed in order to reduce the overall reprojection error.

One advantage of the presented approach is that it allows to recover an Euclidean reconstruction of the scene without any initial solution or prior information, which is one of the drawbacks of most of the existing methods. Another important feature is that the entire calibration process is based on solving linear systems using the SVD decomposition algorithm. The knowledge of the geometric meaning and rank properties of the different transformations represented by the matrices of the process allows to enforce a valid Euclidean reconstruction.

The proposed solution is designed to be versatile in respect to the input data allowing the use of (1) automatically tracked video sequences, (2) manually tracked sequences which usually contain less frames or (3) a set of still images with features and correspondences manually selected.

In the next sections we will present the theoretical background as well as the details final algorithm that attempts to solve the structure from motion problem.

4 CALIBRATION WITHOUT OCCLUSIONS

Before addressing the occlusion and outliers determination problems, we present the basic theoretical analysis and a proposed solution to calibrate image sequences were the significant features are present in all the frames with no exception. This algorithm will be the basic building block of our complete calibration solution. We use a stratification approach to recover both camera parameters and structure of the scene. and the idea is to obtain a projective reconstruction and then upgrade it to a metric/Euclidean structure.

One advantage of the presented calibration approach is that it allows to recover an Euclidean reconstruction of the scene without any initial solution, which is one of the drawbacks of most of the existing methods. Another important feature is that the entire process is based on solving linear systems using the SVD decomposition algorithm. The knowledge of the geometric meaning and rank properties of the different transformations represented by the matrices of the process allows to enforce a valid Euclidean reconstruction. As shown in [SaBS02] high accuracy can be obtained when synthetic data is used, and when noise is added or real data is used, the precision is still acceptable. Figure 2.2 shows an example of calibration of a synthetic sphere with different levels of noise in the 2D measurements. As one can see, a perfect reconstruction can be achieved when no noise is present (a) and the under large amounts of noise (c) the reconstruction is good enough, showing the robustness of the method.

A first projective reconstruction is performed using an iterative approach similar to the proposed in [Trig96] and [Chen00]. Initially all the projective depths are initialized to one, and the algorithm calculates W_4 as a rank-4 approximation of the measurement matrix W using the Singular Value Decomposition (SVD) algorithm [GoVa96], [PTVF92]. Then the λ_{ij} values of W are rescaled with the coefficients of W_4 to make W closer to rank-4. This process is iterated until the solution stabilizes, that is, when an additional iteration will not make significant changes in the rank of W. When the algorithm converges, a close to rank-4 factorization of W in P and X is available with the corresponding projective depths.

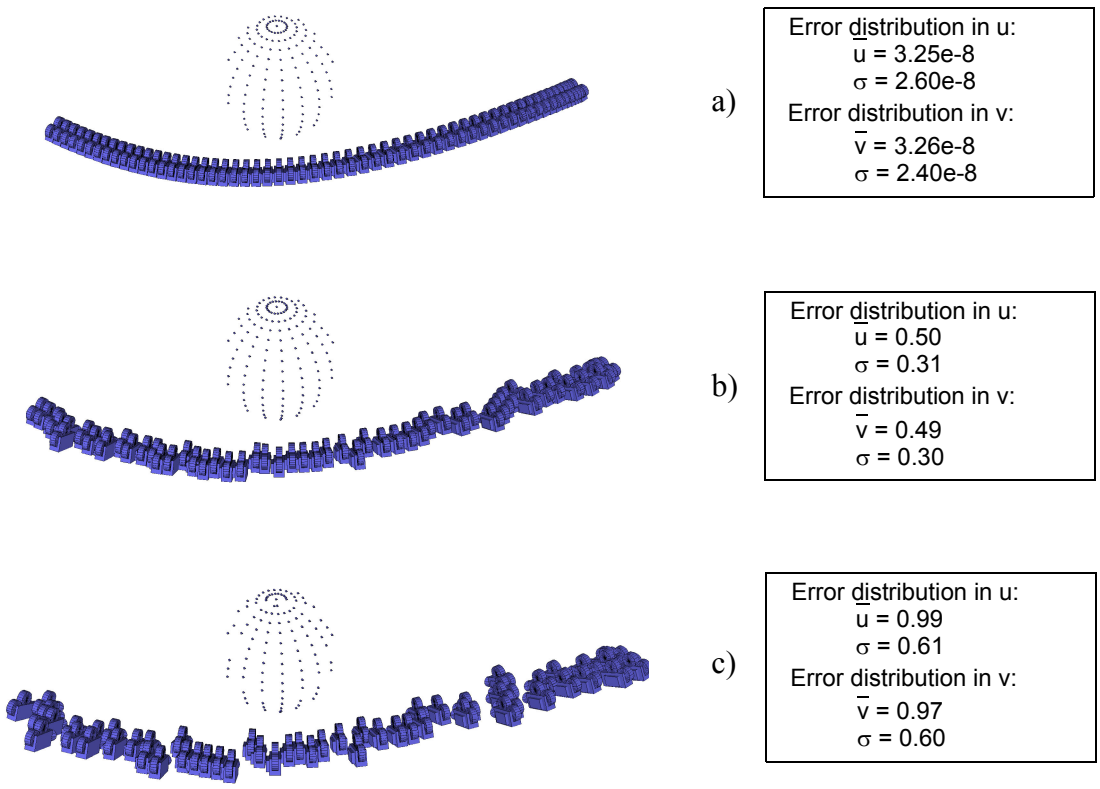


Figure 2.2: Example of sequence calibration without occlusions. A synthetic sphere is calibrated under different noise conditions. Case a) is the original data with no noise. Case b) shows the same set up model with random uniform noise of 2 pixels of amplitude. Case c) is the same as before with 4 pixels of noise amplitude. All the mean errors and standard deviations are calculated as 2D reprojection errors and are expressed in pixels.

Although there is no theoretical proof of convergence, the algorithm has given excellent results with synthetic datasets, recovering the correct solution with errors close to zero ($\sim 10^{-6}$). In presence of noise, or with real data, a close to rank-4 solution is obtained. We have observed that with ideal data the method requires a large amount of iterations, and when noise is added, it stabilizes with much less iterations. This is because when the data is accurate a very accurate solution can be achieved, taking more processing time. We consider this to be a good behavior of our system.

The second part consists of a metric upgrade based on the method proposed by Han and Kanade [HaKa00] and further improved in [SaBS02] and [SaSB03], which calculates a projective transformation that will rectify the projective reconstruction to a metric one. This is done by solving a set of overdetermined linear systems using the SVD algorithm. We improve the original method by adding an enforcing condition to the reconstruction process obtaining overall better calibrated results (measured in 2D reprojection error) and is able to solve cases that were degenerated for the original solution.

The following subsections will introduce the detail of the different stages of this stratified calibration algorithm.

4.1 Projective Reconstruction

The projective factorization method is a generalization of the factorization method which was first developed in [ToKa92],[PoKa93] and [Trig96] for the orthographic and the paraperspective projection models respectively. It provides a more general framework to recover 3D shape and motion from multiple view images up to an unknown projective transformation.

More formally, the goal is to recover the 3D projective structure and motion from m uncalibrated perspective projected images of a scene and a set of n 3D points that are static to the scene. Let $X_j = (X_j, Y_j, Z_j, 1)^T$, $j = 1, \ldots, n$, be the unknown homogeneous 3D point vectors, P_i, $i = 1, \ldots, m$ the unknown 3 x 4 image projections of a pinhole camera model, and $x_{ij} = (u_{ij}, v_{ij}, 1)$ the measured homogeneous image point vectors.

We define the **projective depths** as the non-zero scale factors λ_{ij} relating the world points and its projections as follows

$$\lambda_{ij} x_{ij} = P_i \cdot X_j, \quad i = 1, \ldots, m \quad j = 1, \ldots, n \qquad \text{(eq 2.4)}$$

The points on the scene are defined up to a scale transformation and when correctly normalized points and projections, the λ factors become the true optical depths ([Trig96]).

This decomposition using the projective depths can be stated, in a very similar expression as Equation 2.1, as the following matrix equation

$$W = \begin{bmatrix} \lambda_{11}x_{11} & \ldots & \lambda_{1n}x_{1n} \\ \ldots & \ldots & \ldots \\ \lambda_{m1}x_{m1} & \ldots & \lambda_{mn}x_{mn} \end{bmatrix} = \begin{bmatrix} P_1 \\ \ldots \\ P_m \end{bmatrix} \begin{bmatrix} X_1 & \ldots & X_n \end{bmatrix} = P \cdot X \qquad \text{(eq 2.5)}$$

where W is the $3m \times n$ scaled measurement matrix, P is the $3m \times 4$ perspective matrix and X is the $4 \times n$ shape matrix. The values of the projective depths depend on the 3D structure, which in turn derives from the depths themselves. This coupling of variables reveals Equation 2.5 as a non-linear equation system.

W has to be a rank-4 matrix because it is associated to projections of sets of real points, and both the perspective and shape matrices have a maximum rank of 4 (in the columns of P and the rows of X). Denoting by $W_i = \lambda_{ij}x_{ij} \quad j = 1,\ldots,n$, the $3 \times n$ measurements corresponding to frame i, it can be obtained from Equation 2.5 the equality $W_i = P_i X$, that is equivalent to consider each W_i as a member of the vector subspace spanned by the rows (columns) of X (it also admits a column version) being P_i a linear application that maps between the subspaces.

```
begin
   initialize depths
   do
      normalize depths
      multiply depths by measurements λW
      rank 4 decomposition λW = UDV^T
      calculate scale factors
      update depths
      calculate relative error
   while(relative error > threshold)
   calculate P = UD, X = V^T
end
```

Algorithm 2.1: Projective reconstruction algorithm

This translates into a set of linear constraints that need to be satisfied by any set of image projections that is assumed to be the projections of a 3D point. Moreover, if one can assume that either X or P_i is known, Equation 2.5 can be used to obtain, first, an approximation of the projective homogenous coordinates and, second, to recover the projective depths of the measurements. Conversely, if the projective depths are known, one can recover X and P as a factorization of the measurement matrix, W.

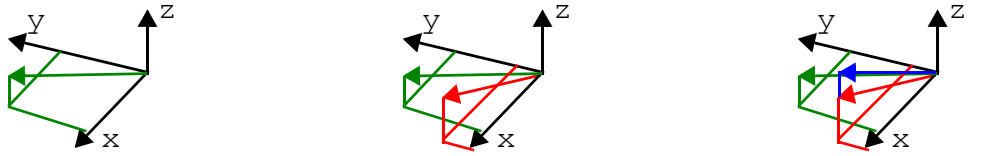
To overcome this nonlinear problem, an iterative projective reconstruction algorithm is proposed based on the Singular Value Decomposition that obtains a valid rank-4 factorization of W. There exist different approaches ([Trig96] and [Chen00]) to construct an iterative algorithm that converges to a rank-4 decomposition of the measurement matrix. Inspired in [Chen00] we have developed a convergent iterative algorithm for successively approximating X and λ_{ij} (see Algorithm 2.1). The different steps involved in the iterations are

1. First set $\lambda^{(0)}_{ij} = 1$, for $i = 1,...,m$ and $j = 1,...,n$ as initial conditions. This can be assumed because the depth values (essentially for the first image) can not be determined uniquely. In fact they can be chosen arbitrarily in the linear subspace generated by the rows of X. As is stated in [MaHe00] the final algorithm is robust w.r.t. initial conditions. Moreover, according to [HaZi00], if the true depths of the 3D points in X remain approximately constant during a sequence, the assumption of $\lambda^{(0)}_{ij} = 1$ can be considered a good first approximation.

2. An initial SVD factorization of $W^{(k)}$, with $W^{(0)} = W$, is computed. We use the standard notation (see [GoVa96]) $W^{(k)} = UDV^T$, where U is a $3m \times n$ matrix which their columns are an orthogonal basis of the output (range) subspace of $W^{(k)}$. D is a $n \times n$ diagonal matrix, their elements σ_i, are known as the singular values of $W^{(k)}$, and finally, V is a $n \times n$ matrix containing an orthonormal basis corresponding to the input (co-kernel) of $W^{(k)}$. A first approximation $P^{(k)} = U_4$ is computed, where U_4 means the submatrix obtained from U using only the 4 first columns (the ones associated to the 4 larger singular values).

Analogously, $X^{(k)} = D_4 V^T_4$ and from that we compute the following estimate $\tilde{W}^{(k)} = P^{(k)} X^{(k)}$. This choice guarantees (according to [GoVa96], page 72) that we get the best rank-4 approximation of $W^{(k)}$, and the spectral distance (using $\| \|_2$) from the subspace of the rank 4 is exactly σ_5.

If the measurement matrix corresponds to a projection of a 3D real points then, with the correct depth values, it must be a rank-4 matrix. Because the depths are unknown, the matrix $W^{(k)}$ can be far from a rank-4 matrix. The value of σ_5 can be used as a



a) $\lambda_{ij}^{(k)} W_{ij}^{(k)}$ b) $\lambda_{ij}^{(k)} \tilde{W}_{ij}^{(k)}$ c) $\lambda_{ij}^{(k+1)} W_{ij}^{(k)}$

Figure 2.3: Rank-4 approximation graphical interpretation.The present depths a) are corrected according the 4-dimensional projection b) and used to recompute the new depths c).

measure of this distance and it should have zero limit if the iterative process is supposed to converge to the solution.

3. Once the rank-4 approximation $\tilde{W}^{(k)}$ is computed, we get an estimate of the 3D coordinates, $X_j^{(k+1)}$ of the points (living in a rank-4 space) based on the present depth value for each point $\lambda_{ij}^{(k)}$. As this value is not the correct one, for each point we can improve the previous estimation of the depth using the new 3D coordinates. To preserve the projection of the real points (the measurements) x_{ij}, we only need to scale $\lambda_{ij}^{(k)}$ along the visual line, the line from the center of projection through x_{ij}. The new depth $\lambda_{ij}^{(k+1)}$ is chosen to coincide with the projection of $X_j^{(k+1)}$ into the visual line. In this way we are forcing each depth to be as close as possible to the ideal present rank-4 decomposition. The previous condition can be summarized with the following projective depth update formula:

$$\lambda_{ij}^{(k+1)} = \lambda_{ij}^{(k)} \cdot \frac{\tilde{W}_{ij}^{(k)T} \cdot W_{ij}^{(k)}}{W_{ij}^{(k)T} \cdot W_{ij}^{(k)}} \qquad \text{(eq 2.6)}$$

A graphical interpretation of the rank-4 approximation and the computation of Equation 2.6 is shown in Figure 2.3.

4. After computing the new value of the depth matrix we get an update measurement matrix $\tilde{W}_i^{(k+1)} = x_{ij}\lambda_{ij}^{(k+1)},\ j = 1, \ldots, n$ and the process is repeated until the value of the corresponding $\sigma_5^{(k+1)}$ is either small enough or it is stabilized. When noise is present in the projection measurements, the values of $\sigma_5^{(k+1)}$ can reach an small stable value but different from zero. Before proceeding to the next iteration is strongly encouraged to perform a normalization of the measurements so that the projective depths are as close to unity as possible, improving the numerical stability and convergence of the algorithm. This is performed as proposed in [StTr96] in two steps, first normalizing the columns of *W* and applying the same factors to the columns of *X*, and then normalizing by rows and applying the same factors to the rows of *P*. This process is performed in this way because the projective depths are unique up to a set of scale factors, as stated in the following equation

$$(\alpha_i\beta_j\lambda_{ij})x_{ij} = (\alpha_i P_i)\cdot(\beta_j X_j), \quad i = 1, \ldots, m \quad j = 1, \ldots, n$$

4.2 Metric Upgrade

The previous factorization algorithm decomposes Equation 2.5 and recovers the motion of the camera and shape of the scene up to a linear projective transformation *H* known as the *Projective Distortion Matrix* (PDM):

$$W = PX = PHH^{-1}X = P^M X^M \qquad \text{(eq 2.7)}$$

The main goal of this PDM is to upgrade from a projective space to a metric space by adding constraints on some entities well defined in projective geometry: The plane at infinity π_∞ and the absolute conic Ω_∞. By fixing the plane at infinity we transform the projective space into an affine one that contains more structure such as the property of

parallelism. This geometric concept can be seen as the plane where two parallel lines intersect. Regarding the absolute conic Ω_∞, it is a more abstract concept than the plane at infinity, and it can be seen as an imaginary invariant circle located in the plane at infinity. Once it is determined, we can establish the properties of relative distances and angles, upgrading the affine space to metric. We suggest [HaZi00] as an excellent source for all the theoretical background of projective geometry.

Then, in order to upgrade the projective solution to metric, the next step is to calculate an homography H such that, $P_i^M = P_i H$ and $X_j^M = H^{-1} X_j$ are the metric reconstruction of the scene. Using the pinhole camera model, the metric camera matrices can be decomposed in terms of the internal and external parameters as

$$P_i^M = K_i[R_i|t_i] \, , i = 1, \ldots m$$

where the *3 x 3* symmetric matrix K_i is the internal camera parameters matrix, R_i is the euclidean rotation matrix and t_i is the translation vector.

To be more precise, we are not concerned with the absolute scaling, rotation and translation of the reconstructed scene and we can factor this out by choosing the world coordinate reference to coincide with the first camera, so $R_1 = I$ ant $t_1 = 0$. Then the first euclidean camera matrix becomes ${P_1}^M = K_1[I|0]$. We can factor out the same similarity component from the projective reconstruction, having $P_1 = [I|0]$. Then the condition $P_1^M = P_1 H$ becomes $[K_1|0] = [I|0]$ and we can write H as

$$H = \begin{bmatrix} K_1 & 0 \\ v^T & 1 \end{bmatrix}$$

In affine and metric space, the plane at infinity π_∞ has the canonical form $\pi_\infty = (0, 0, 0, 1)^T$, so we can calculate the plane at infinity in projective space from the relation $P^M = PH$ obtaining the following

$$\pi_\infty = H^{-T}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{bmatrix} (K_1)^{-T} & -(K_1)^{-T}v \\ 0 & 1 \end{bmatrix}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -(K_1)^{-T}v \\ 1 \end{pmatrix}$$

If we write the plane at infinity as an usual plane, $\pi_\infty = (p^T, 1)^T$, where $p = -(K_1)^{-T}v$, then the PDM that transforms a projective reconstruction to a metric one, both in canonical form, is

$$H = \begin{bmatrix} K_1 & 0 \\ -p^T K_1 & 1 \end{bmatrix} \qquad \text{(eq 2.8)}$$

From this result we can see that only 8 parameters are necessary to transform a projective reconstruction into a metric one (3 parameters for p and 5 for K_1). This agrees with the geometric arguments: the plane at infinity and the absolute conic (3 and 5 degrees of freedom respectively). Based on these results, we can derive the auto-calibration equations, first we express the rest of the projective cameras distinguishing between the first three and the last columns, $P^i = [A^i | a^i]$, and from $P_i^M = P_i H$ and Equation 2.8 we obtain

$$K_i R_i = (A_i - a_i p^T)K^1, \quad i = 2 \ldots m \qquad \text{(eq 2.9)}$$

which can be rearranged considering $R_i = (K_i)^{-1}(A_i - a_i p^T)K^1, \quad i = 2 \ldots m$, and the orthogonal property of rotation matrices $RR^T = I$, obtaining

$$K^i K^{iT} = (A^i - a^i p^T) K^1 K^{1T} (A^i - a^i p^T), \quad i = 2 \ldots m \qquad \text{(eq 2.10)}$$

On the other hand, the main goal of the autocalibration process is to fix the plane at infinity π_∞, which allows to establish measures of parallelism and intersection between elements of R^3 and the image of the absolute conic ω, which builds on top of the plane at infinity and provides the angle and relative length measurement capabilities. For a single view, we can derive a relation between the calibration matrix K (intrinsic parameters) with the image of the absolute conic by determining the mapping between π_∞ and the camera image plane. Points on π_∞ can be written as $X_\infty = (d^T, 0)^T$ and are projected by a general camera as

$$x = PX_\infty = K[R|t]\begin{pmatrix} d \\ 0 \end{pmatrix} = KRd$$

The conic at the infinite Ω_∞ which is expressed as the identity matrix in π_∞ maps to the image plane forming the image of the dual conic $\omega = (KR)^{-T} I (KR)^{-1} = (KK^T)^{-1}$. For convenience we will consider the dual image of the absolute conic $\omega^* = \omega^{-1} = (KK^T)$.

Taking this into account, we can say that the symmetric matrix $K_i K_i^T$ from Equation 2.10 is the dual image of the absolute conic, ω_i^*, which is related with the *absolute dual quadric*, Q_∞^*, a degenerated dual quadric represented by a *4 x 4* homogenous matrix of rank 3, by

$$\omega_i^* = P_i Q_\infty^* P_i^T \qquad \text{(eq 2.11)}$$

Q_∞^* was introduced by Triggs [Trig97] as a very convenient representation of the autocalibration because its main advantage of encoding both π_∞ and Ω_∞ very concisely.

Furthermore, Q_∞^* has some very interesting properties such as that in an Euclidean frame has the canonical form

$$Q_\infty^* = \begin{bmatrix} I_{3x3} & 0 \\ 0 & 0 \end{bmatrix} = \tilde{I}$$

so in a projective coordinate frame, it will follow the projective transformation rule of for dual quadrics giving

$$Q_\infty^* = H \begin{bmatrix} I_{3x3} & 0 \\ 0 & 0 \end{bmatrix} H^T = H\tilde{I}H^T$$

this means that in an arbitrary projective frame the dual absolute quadric is represented by a symmetric *4 x 4* matrix that has rank-3, its null space is the vector that represents the plane at infinity and it is semi-definite positive or negative depending on a scale factor.

Moreover, using Equation 2.8 the projective reconstruction gives the relation

$$Q_\infty^* = \begin{bmatrix} K_1 K_1^T & -K_1 K_1^T p \\ -p^T K_1 K_1^T & p^T K_1 K_1^T p \end{bmatrix} = \begin{bmatrix} \omega_1^* & -\omega_1^* p \\ -p^T \omega_1^* & p^T \omega_1^* p \end{bmatrix} \qquad \text{(eq 2.12)}$$

That substituted in Equation 2.11 gives again the autocalibration relations of Equation 2.10. This provides the interpretation of Q_∞^* as being a fixed quadric under the

motion of the camera and each of the dual images of the absolute conic ω_i^* are the different images of Q_∞^* for each of the views.

The most important consequence is that assuming certain constraints on ω_i^*, as we show below, we can translate them to Q_∞^* using Equation 2.11 via the known matrices P_i and solve for Q_∞^* in projective space, using the resulting matrices from the projective factorization.

4.2.1 Closed form solution for the autocalibration equations

A linear system can be obtained from and assuming some knowledge on the camera internal parameters:

- Principal point is at the center of the image plane, then $(\omega_i^*)_{13} = (\omega_i^*)_{23} = 0$.
- Zero skew in the pixels implies $(\omega_i^*)_{12} = 0$.
- Equal aspect ratio implies $(\omega_i^*)_{11} = (\omega_i^*)_{22}$.

These are very common assumptions in the computer vision literature that leave just the focal length as a variable parameter for all the available views. Moreover, these conditions provide that 4 linear equations can be derived for each view and that also redefine the shape of Q_∞^* into

$$Q_\infty^* = \begin{bmatrix} q_{11} & 0 & 0 & q_{14} \\ 0 & q_{11} & 0 & q_{24} \\ 0 & 0 & 1 & q_{34} \\ q_{14} & q_{24} & q_{34} & q_{44} \end{bmatrix} = HH^T \qquad \text{(eq 2.13)}$$

The autocalibration equations become an overdetermined linear system of *4 x m* equations generated from the zero entries of the different ω_i^* when applying Equation

2.10 for each view *i*. Additionally, with the parametrization of Q_∞^* shown in Equation 2.13, we can solve the system of equations using the SVD decomposition.

A very important condition must be satisfied, which is the degenerated rank of Q_∞^*. In order to do so, a supplementary non linear condition $det(Q_\infty^*) = 0$ must be added to the least square solution estimation algorithm, which turns to be a cumbersome problem.

However, we can express the solution of the overdetermined system as a linear combination of the solution vector and a constant λ by the eigenvector with the lowest singular value, which is equivalent to assume that the matrix of the equations system is not full rank and has a one dimensional kernel, spawned by *e*, the last column of the V^T matrix of the SVD decomposition.

$$0 = \begin{vmatrix} x_0-\lambda e_0 & 0 & 0 & x_1-\lambda e_1 \\ 0 & x_0-\lambda e_0 & 0 & x_2-\lambda e_2 \\ 0 & 0 & 1 & x_3-\lambda e_3 \\ x_1-\lambda e_1 & x_2-\lambda e_2 & x_3-\lambda e_3 & x_4-\lambda e_4 \end{vmatrix} = \begin{vmatrix} f_1^2 & 0 & 0 & -f_1^2 p_x \\ 0 & f_1^2 & 0 & -f_1^2 p_y \\ 0 & 0 & 1 & -f_1^2 p_z \\ -f_1^2 p_x & -f_1^2 p_y & -f_1^2 p_z & xx \end{vmatrix} \qquad \text{(eq 2.14)}$$

a third degree polynomial in terms of λ can be obtained to enforce the supplementary condition. Then, using SVD to obtain *H* and back-substituting in Equation 2.7, a final metric reconstruction is computed under the assumption of known principal points and skew values.

Once all the metric projection matrices have been recovered we proceed to extract the focal length information for each of the reference view projection matrices P_i

$$P_i = K \cdot P = \begin{bmatrix} \mu f & 0 & 0 \\ 0 & \mu f & 0 \\ 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} i_x & i_y & i_z & t_x \\ j_x & j_y & j_z & t_y \\ k_x & k_y & k_z & t_z \end{bmatrix} = \begin{bmatrix} \mu f i_x & \mu f i_y & \mu f i_z & \mu f t_x \\ \mu f j_x & \mu f j_y & \mu f j_z & \mu f t_y \\ \mu k_x & \mu k_y & \mu k_z & \mu t_z \end{bmatrix}$$

Since the vectors i, j belong to an Euclidean rotation matrix their norm is unitary and, by taking the modulus of the respective components in P we obtain the desired focal length. However, since this has been obtained as a least square solution, the two values possible values might be different due to noise in the data. Therefore an average solution is computed as well as the scale factor μ, that even it does not play a very important role in the projection equation, since it cancels out when doing the perspective division, it is interesting to decouple it from the rotation matrix.

An interesting effect that may occur when extracting the focal length and scale factor from the matrices, is that the final camera axis (expressed as the rows of the rotation matrix) might not be orthonormal because the solution is least squares regarding the reprojection errors and does not enforce strictly orthonormality between the axis. The only way to obtain orthonormal cameras is to perform a final optimization where this condition can be enforced.

4.3 Refinement of the metric solution

The solution presented to the autocalibration process is a closed form least squared constrained approximation of the structure from motion problem. A final non-linear optimization process is required in order to reduce the reprojection error accounting for all the non-linearities not recovered in the metric solution. Moreover, if a more complete camera internal parameters description is required (e.g. adding the principal point), it can be incorporated into the optimization process as well.

Additionally, if during the preprocessing of the measurement matrix some of the measurements have been left out because they were not present in all the views, we can include them in this nonlinear analysis to improve the overall error. That is possible because all the camera parameters have been recovered and we can estimate by a least squared error approximation an initial 3D value of the reconstructed point.

A bundle adjustment process is applied to obtain a maximum likelihood estimation that minimizes the mean squared distances between the measurements $x_{i,j}$ and the reprojected image points from a new estimation of P_i^M and X_j. The minimization criterion can be expressed as

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} d(x_{i,j}, P_i^M X_j)^2$$

As shown in [TMHF00], there exist several approaches in the literature to solve this global minimization problem, being the most commonly used the *Levenberg-Marquardt* (LM) algorithm because, as we will see, the number of minimization equations grows extremely fast with the number of points and views but presents a particular structure that can be used to optimize the calculation. Moreover, in the best case, LM behaves like a Newton's algorithm with quadratic convergence.

We can write the iterative equations that drive the Levenberg-Marquardt algorithm as

$$\hat{U} = f(\hat{V}^{(i+1)}) + \varepsilon^{(i+1)} = f(\hat{V}^{(i)} + \Delta^{(i)}) + \varepsilon^{(i)} = f(\hat{V}^{(i)}) + J\Delta^{(i)} + \varepsilon^{(i)}$$

where f is the function to minimize and the Newton error minimization equations, the normal equations $N\Delta = J^T J\Delta = J^T\varepsilon$, are modified to the *augmented normal*

equations $N'\Delta = J^T\varepsilon$ where $N'_{ii} = (1+\lambda)N_{ii}$ and $N'_{ij} = N_{ij}, i \neq j$. The LM iteration proceeds by initializing λ to a small value, e.g 10^{-3}. If the solution obtained for $\Delta^{(i)}$ reduces the error, the solution is accepted and the value of λ is divided by 10 before the next iteration. If the case is that the error increases, λ is multiplied by 10 and the augmented normal equations are solved again until a proper value for λ is obtained that decreases the iteration error.

The LM iteration is a combination between a Newton and a steepest descent iteration. When λ is small, the effect on the augmented normal equations can be negligible and the iteration equation is just the one of Newton's method. On the other hand, if Newton fails to converge (i.e. the error increases) the value of λ is increased, and when it is large enough (substantially greater than 1) the off-diagonal terms of J^TJ become insignificant and can be ignored in front of the diagonal terms. Then the increment is $\Delta^{(i)}$ is driven by the gradient of the function which will decrease the error in a steepest descent manner (for a more formal proof we suggest [HaZi00]).

The non-linear functions no minimize in our case are the pinhole camera projection equations (Equation 2.2) for the different measurements and frames. They can be re-written as

$$x = \frac{f(i_xX + i_yY + i_zZ + t_x) + \beta(j_xX + j_yY + j_zZ + t_y) + u_0(k_xX + k_yY + k_zZ + t_z)}{k_xX + k_yY + k_zZ + t_z}$$

$$y = \frac{\alpha f(j_xX + j_yY + j_zZ + t_y) + v_0(k_xX + k_yY + k_zZ + t_z)}{k_xX + k_yY + k_zZ + t_z}$$

assuming our hypothesis of no skew ($\beta = 0$) and central point perfectly centered on the image plane ($u_0 = v_0 = 0$), and normalized coordinates ($\alpha=1$) we obtain the simplified version

$$x = \frac{f(i_x X + i_y Y + i_z Z + t_x)}{k_x X + k_y Y + k_z Z + t_z}$$
$$y = \frac{f(j_x X + j_y Y + j_z Z + t_y)}{k_x X + k_y Y + k_z Z + t_z}$$

now we can construct a vector of measurements by concatenating the values of the 2D features by frame and then by different frames

$$\hat{m} = (x_{1,1}, \ldots, x_{1,n}, \ldots, x_{m,1}, \ldots, x_{m,n})^T$$

Then we can define our vector of parameters as the concatenation of the different variables for the camera parameters. To reduce the overall number of parameters and to enforce orthonormality of the camera axis, instead of using the rotation matrix (9 parameters) we will estimate a rotational unitary quaternion, with 3 parameters q_1, q_2, q_3 (q0, the fourth element, is fixed by the unitary norm condition) that will be expanded to a full rotation during the evaluation of the Jacobians and errors. Then the parameter vector will be

$$\begin{aligned} cam_i &= (t_x, t_y, t_z, q_1, q_2, q_3, f)^T \\ point_j &= (X_x, X_y, X_z)^T \\ V &= (cam_1, \ldots, cam_m, point_1, \ldots, point_n)^T \end{aligned}$$

It is easy to see that given the structure of our problem, the Jacobian matrix presents a very large size, and also a very sparse structure, since the derivatives of the projection

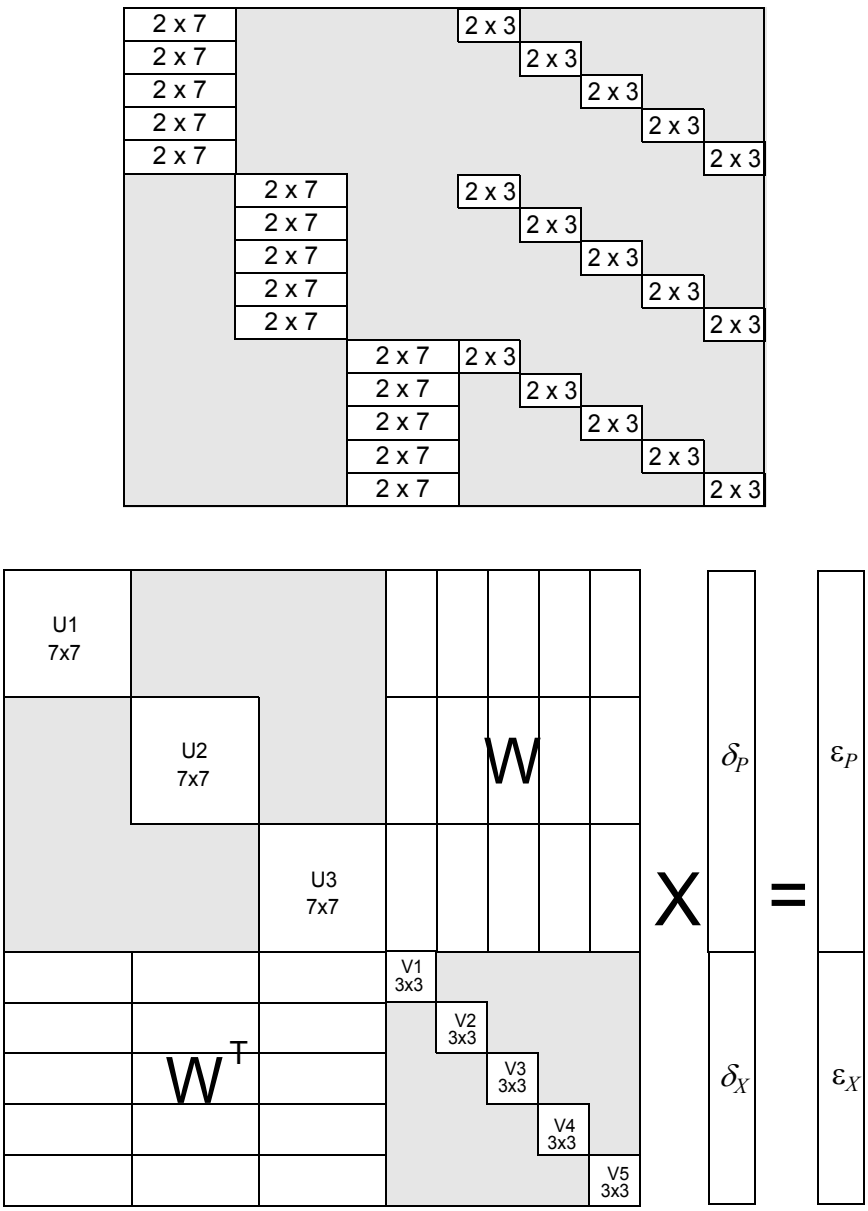


Figure 2.4: Sparse structure of the Jacobian and Normal Equations. This example shows the matrices for a bundle adjustment of 3 images and 5 points

equations of one frame are completely independent from the other frames. This directly affects the structured of the normal equations matrix $N = J^T J$ providing a sparse equation system (see Figure 2.4) that can be solved very efficiently. The blocks that shape this matrix can be easily written down as follows

$$U_k = \sum_i \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{P}_k}\right)^T \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{P}_k}\right)$$

$$V_i = \sum_k \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{X}_i}\right)^T \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{X}_i}\right)$$

$$W_{ki} = \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{P}_k}\right)^T \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{X}_i}\right) \qquad \text{(eq 2.15)}$$

$$\varepsilon(P_k) = \sum_i \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{P}_k}\right)^T \varepsilon_{ki}$$

$$\varepsilon(X_i) = \sum_i \left(\frac{\partial \hat{m}_{ki}}{\partial \hat{X}_i}\right)^T \varepsilon_{ki}$$

where $\frac{\partial \hat{m}_{ki}}{\partial \hat{P}_k}$ and $\frac{\partial \hat{m}_{ki}}{\partial \hat{X}_i}$ are the matrices of partial derivatives of the components of the state vector $\hat{m}$ at the *i-th* point in the *k-th* view to the camera parameters P_k and points X_i respectively. The normal expressions can then be expressed as a

$$\begin{bmatrix} U^* & W \\ W^T & V^* \end{bmatrix} \begin{bmatrix} \delta_P \\ \delta_X \end{bmatrix} = \begin{bmatrix} \varepsilon_P \\ \varepsilon_X \end{bmatrix} \qquad \text{(eq 2.16)}$$

A first step on decoupling these two equations is to left-multiply both sides of the equation by $\begin{bmatrix} I & -WV^{*-1} \\ 0 & I \end{bmatrix}$ assuming that V^* is invertible, obtaining

$$\begin{bmatrix} U^* - WV^{*-1}W^T & 0 \\ W^T & V^* \end{bmatrix} \begin{bmatrix} \delta_P \\ \delta_X \end{bmatrix} = \begin{bmatrix} \varepsilon_P - WV^{*-1}\varepsilon_X \\ \varepsilon_X \end{bmatrix} \qquad \text{(eq 2.17)}$$

Since the top right submatrix has been eliminated, we can solve for δ_P using the following equation

$$(U^* - WV^{*-1}W^T)\delta_P = \varepsilon_P - WV^{*-1}\varepsilon_X \qquad \text{(eq 2.18)}$$

And then back-substitute for δ_X obtaining

$$V^*\delta_X = \varepsilon_X - W^T\delta_P$$

In our specific problem of calibration, it turns out that the V^* is *3m x 3m* and made of diagonal blocks od *3 x 3*, making very efficient the calculation of V^{*-1}. The matrix W is a set of blocks of *7 x 3* giving a final matrix of *7m x 3n*. The most expensive operation to solve Equation 2.18 is the inversion of the term $(U^* - WV^{*-1}W^T)$ which turns to be a matrix of *7m x 7m* elements, where m is the number of reference images and n is the number of measured features. Keeping in mind that usually $n >> m$ in order to obtain statistically stable solutions, we have developed this sparse approach that provides a better solution than solving directly the normal equations which require inverting J^TJ inversion which is a matrix with a size of *3n x 3n*.

An additional difficulty on top of solving the normal equations is the calculation of the coefficients of the matrices, which correspond to the partial derivatives of the projection equations with respect to the elements of the state vector. Although they could be calculated analytically for a given projection model, we have chosen to compute them numerically for flexibility reasons.

5 CALIBRATION WITH OCCLUSIONS AND OUTLIERS

The autocalibration method presented in Section 4 starts from the assumption that all features used are present in all the views, all features are static points on the surface of the object (i.e. all features are valid for calibration) and for sake of computational time, it is recommended not to use too many frames. However, this scenario does not

happen very often since tracked measurements of camera paths around objects always have features that are not visible in all the frames. Moreover such video sequences can have more than thousands of frames and in the case of using an automatic tracker, due to highlights of spurious detected features, not all the tracked points might be real points of the scene.

In this section we present a divide-and-conquer strategy built on top of the presented autocalibration method that addresses the following fundamental problems:

- Division of the original video sequence into sub-sequences suitable for calibration without occlusions. And for each subsequence, selection of the most significant frames, understanding by significant those frames that introduce additional 3D information.
- For each subsequence, detection and classification of the input measures into inliers and outliers, guaranteeing that the measures used during autocalibration do not contain misleading information.
- Merging of subsequences into the complete sequence minimizing error drifting. A non-linear optimization is applied on the final solution.

The following subsections extend on these topics.

5.1 Sequence division

Given an image sequence, the typical output that an automatic tracking system provides is a set of 2D measures and the correspondence relations between the different frames. A common setup is to have the tracker to track a fixed number of features and when any of them is lost a new feature is found as a replacement. This originates that the

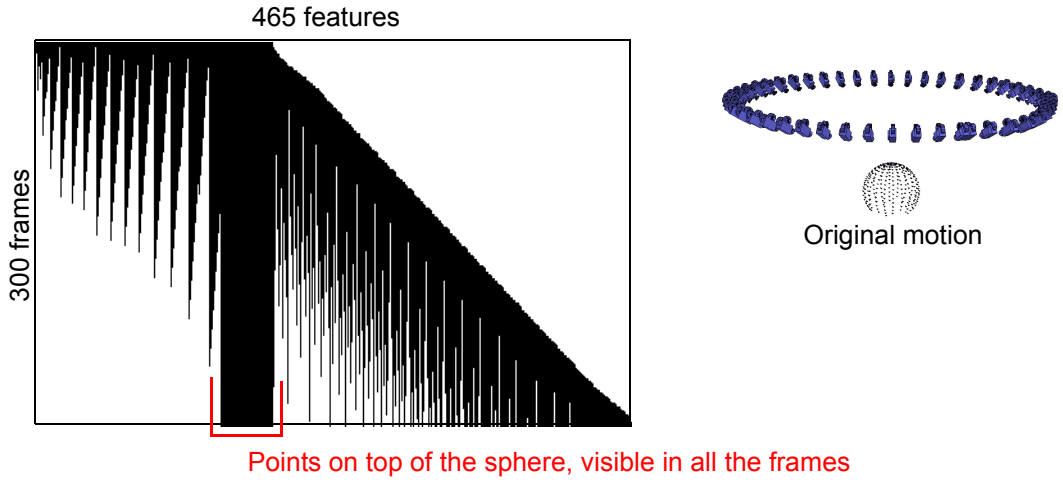


Figure 2.5: Measurement pattern of a sequence with occlusions.The image illustrates a circular motion pointing down 45^o at the center of a synthetic sphere that generates this characteristic measurement pattern.

measurement matrix is partially filled (as shown in Figure 2.5), making impossible to use the proposed calibration algorithm.

Moreover, when dealing with video sequences we obtain an oversize set of images, where consecutive frames reflect very little scene changes, specially when the relative motion of the camera slows down or even stops. In such cases it is interesting to detect which are the significant frames, the keyframes, that will introduce enough 3D information to calibrate the sequence.

In this work we propose a method that fragments the sequence into disjoint sets of keyframes, suitable to be calibrated. The fragmentation algorithm is divided in two steps, (1) to find the last frame in the sequence suitable to be calibrated together with the initial one and (2) find a set of keyframes between the selected beginning and end of the fragment.

The first part starts with the set of measurements from frame *i* and tracks their behavior in the subsequent frames until one of the following conditions is satisfied:

- The end of the sequence is reached.
- More than selected percentage of the original features do not appear in frame *j*.

When this occurs, a new subsequence is created between frame *i* and *j* and we proceed to the second part which consists of identifying the set of keyframes in order to reduce the dimensions of the measurement matrix. This sequence fragmentation process is repeated until all the frames have been assigned or discarded and all the subsequences suitable to be calibrated have been identified. In order to guarantee a minimal connectivity between the different fragments, an overlap of one frame is enforced during the fragmentation, so the last frame of a sub-sequence corresponds to the first frame of the next one.

5.1.1 2D homography determination

Given a set of 4 points $x_i \in R^2$ in image *i*, and the corresponding points $x'_i \in R^2$ in image *j*, it is possible to define an homography *H* that establishes a planar projective transformation from x_i to x_i'. Moreover this transformation has a closed form linear solution if we consider that every pair of points give the following equations

$$\begin{bmatrix} kx' \\ ky' \\ k \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \text{(eq 2.19)}$$

where usually it is assumed that $h_9 = 1$ giving a equation system of 8 unknowns that can be solved if 4 or more points are available.

Planar homographies generate a 2D mapping between points provide that there exist such transformation. When the positional variation between the two frames follow more complicated models, such as a 3D space transformation, the error of the mapping increases. This error is a good indicator on how much non-planar motion exists between two frames, and we can exploit this to determine wether a frame k such $i \leq k \leq j$ provides enough 3D information to be included as a keyframe for posterior calibration.

Another fact that has to be taken into account is that not all the measurements found and tracked might be real points on the surfaces of the scene, and although they have been determined and followed properly in the frames, this outliers are noisy measurements that can severely affect the calibration performance.

In order to solve this problem we use a robust estimator, the *Random Sampling Consensus* (RANSAC), approach to detect and eliminate the outliers. According to [HaZi00], the concept is very simple: from a set of measurement pairs we randomly select a minimal set of 4 pairs that can be used to obtain an homography solution, and we measure the *support* that the calculated solution gets from the rest of input data (i.e how many of the measurements project with less error than a threshold using that homography solution). This process is repeated until the best fit that contains the maximum number of inliers is obtained. The complete algorithm is shown in Algorithm 2.2

Obviously, when the number of measurements is large, it becomes unfeasible to evaluate all potential sets of 4 elements. Instead, the RANSAC approach offers a probability formula that bounds the maximum number of samples to test depending on

```
begin
   N = ∞, iterations = 0
   do
      select randomly a sample of 4 elements
      calculate homography H
      classify all measures in inlier/outlier
      set ε=1-(#inliers)/(#points)
      set N according to Equation 2.20
      iterations++
   while(N > iterations)
end
```

Algorithm 2.2: RANSAC algorithm. This approach is used for identifying 2D inlier sets between to images related with a 2D Homography.

the size of the minimal set used to obtain a solution and what is the probability that one of the evaluated sets does not contain outliers. The following equation gives the number of random tests required

$$N = \frac{log(1-p)}{log(1-(1-\varepsilon)^{s})} \qquad \text{(eq 2.20)}$$

where N is the number of selections of s measurements, p is the probability that at least one of the random sets is free from outliers (usually is set to 0.99), and ε is the probability that a given measurement is an outlier. However in the general case the value of ε is unknown, and it is necessary to calculate an estimation. This can be done adaptively, starting with a worst case scenario ($\varepsilon = 1$), and every time a sample provides a lower percentage of outliers, the upper limit of samples N is recalculated using Equation 2.20.

Once an initial set of inliers has been determined, an improved homography reconstruction is computed by using all the inliers simultaneously (i.e. by constructing and overdetermined equation system similar to Equation 2.19). Then the measurements are re-classified into inlier/outlier using the new homography, and this cycle is repeated

until the set of inliers remains constant. Usually this robust estimate converges in less that ten iterations, and can increase the number of inliers by a factor of 2.

Other approaches of RANSAC, [HaZi00], include a non linear minimization and guided matching to correct misclassified correspondences, however this is out the scope if this work.

We use the percentage of outliers as a measurement of how good the 2D fitting is in our corresponded measures between frames *i* and *k*. Whenever this value is lower than a given threshold, the pair of images is considered to have underlying motion not mappable with a 2D homography and frame *k* is marked as a keyframe. Then this process is repeated starting from *k* as the initial frame and iterating over *k+1,...,j* to detect the next keyframe.

The final set of keyframes for a specific fragment will contain *i* and *j* as the lower and upper bounds, as well as all the frames in between that have been selected during the RANSAC analysis.

5.2 3D Inlier determination

Once each of the individual fragments have been identified, the next step is to calibrate them using the approach presented in Section 4. A new measurement matrix *W'* can be built using the keyframes and those features that are visible in all those frames. However, the outliers are still mixed with the good measurements and need to be removed. It is important to notice that the set of inliers/outliers found during the homography calculations are of no use since they were determined for a 2D mapping, not a 3D reconstruction.

We have designed a similar approach as the RANSAC in the previous section, that performs a similar random analysis in 3D projective space. The minimal set of measurements that can construct a projective solution for all the frames of the sub-sequence is four, since *W'* has to be a rank-4 matrix.

At this point we could use the proposed iterative projective reconstruction algorithm of Section 4.1 to perform the RANSAC iterations. However, since this preliminary step is only to determine the inliers and usually starts with a large set of measures and has to be repeated several times, the iterative algorithm can be very slow.

Instead, an affine reconstruction algorithm [HaZi00] can be used because it presents a closed form least squares solution very fast to compute. The obtained solution will not be as accurate as a true projective reconstruction in terms of inlier/outlier classification, but it removes the most obvious outliers from the set of points.

5.2.1 Affine reconstruction

Tomasi and Kanade introduce in [ToKa92] a one step factorization algorithm to solve the SFM problem when a set of correspondences is available and the camera model is affine (i.e. orthographic projection). The projection equations for this model are

$$\begin{bmatrix} x \\ y \end{bmatrix} = M \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

where M is a 2×3 matrix that represents an affine transformation and t is a translation vector. A solution for m views that minimizes the geometric projection error

can be found with more than 4 points, by first computing a translation vector for each frame as the centroid of the measurements

$$t_i = \frac{1}{n}\sum_j x_{i,j}$$

then centering the measurement data by subtracting the centroid in each frame, and finally constructing a $2m \times n$ measurement matrix W using that centered data. After decomposing $W=UDV^T$ using SVD, UD will provide the camera matrices M_i and the structure $X_1,...,X_n$ is computed from the matrix V.

The affine model is not a very good one to fit the type of input data obtained using regular cameras since it does not account for perspective effects, but as shown in the experiments it allows to quickly obtain an initial set of inliers. In order to improve the inlier/outlier classification, a final iterative reevaluation of the solution and the inlier set, similar to the one used during the 2D homography estimation, is performed using this time the iterative projective reconstruction algorithm presented in Section 4.1. Although this method is much slower than the affine reconstruction is much more precise and the refinement takes usually less than 10 iterations. Moreover once the set of inliers have been found, the projective reconstruction $W''=PX$ of the fragment has also been retrieved as part of the process.

5.3 Subsequence Calibration

Once a subsequence has been determined, the set of inliers is known, we proceed to perform a complete metric calibration followed by a bundle adjustment as described in Section 4.

5.4 Sequence merging

Up to this point, following the first half of a divide and conquer strategy the presented method has taken a long sequence and has fragmented it into sub-sequences free from outliers, and each of them has been properly calibrated into a metric reconstruction. The next step is to merge the information of the individual fragments to recover a complete scene structure.

When performing the division into each of the subsequences we have enforced that two contiguous fragments share a common frame. Therefore the merging of two subsequences is performed in metric space by determining the set of common points, between the last frame of one subsequence and the first of the next one, that by construction corresponds to the same camera and measurements.

Suppose that a point j has a reconstructed version X_j in the first subsequence and X'_j in the second one. Since both represent the same point in space it follows that

$$X_j \cong HX'_j$$
$$P_i \cong P'_i H^{-1}$$

where P_i and P'_i are the metric camera projection matrices representing the same reference view expressed in different basis. H is a homography that maps the points from one basis to the other one. We want to determine H minimizing the following distance

$$\sum_i d^2(P_i H X'_j, P_i X_j)$$

for all common overlapping points j between the two subsequences. The distance function is assumed to be the standard Euclidean distance. Exploiting that we have two

reconstructed metric frames P_i, P'_i, we can perform a basis alignment so the common points are expressed in the same reference. Without any lack of generality we can multiply each of the camera matrices by its inverse (pseudo inverse if we use a 3 *x* 4 notation) as follows

$$P_i X_j = P_i P_i^{-1} P_i X_j = [I|0] P_i X_j = [I|0] \tilde{X}_j$$
$$P'_i X'_j = P'_i P'^{-1}_i P'_i X'_j = [I|0] P'_i X'_j = [I|0] \tilde{X}'_j$$

Since the two sets of points are representations of the same set of real points and they share identical 2D projections, we can restrict the homography H to be a uniform scale and a translation, given the following expression

$$H = \begin{bmatrix} s & 0 & 0 & t_x \\ 0 & s & 0 & t_y \\ 0 & 0 & s & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

since there are 4 unknowns, a solution to merge the subsequences can be achieved if 4 common points are available between the two fragments. If more points are shared, an overdetermined equation system can be built and a least squares solution can be obtained.

Using the method presented above we are able to merge two sequences that share a common frame and more than 4 points. However, when multiple fragments have to be merged, a strategy has to be defined so no error propagation or drifting can occur. In the trivial solution of performing a sequential merging, if any of the subsequences has been reconstructed with a larger error than the rest, it can compromise the merging of the

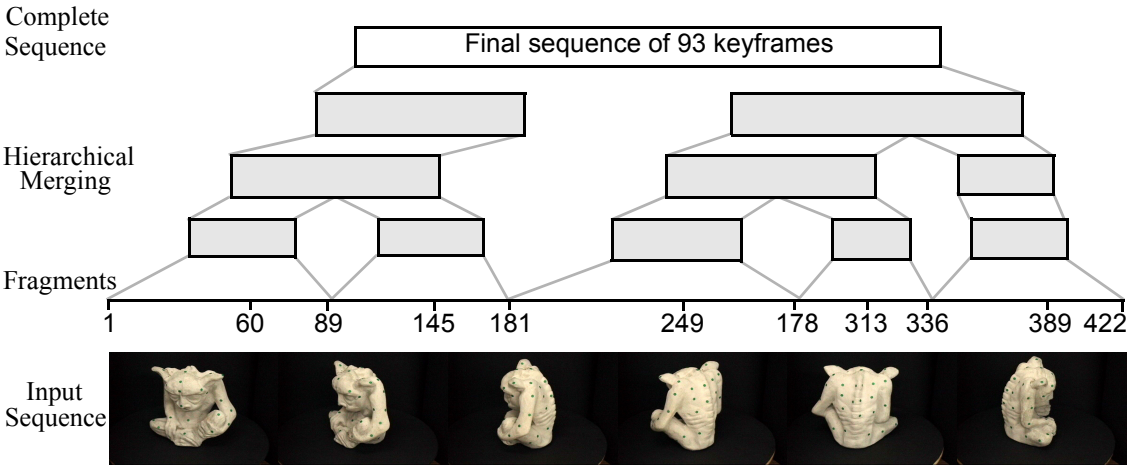


Figure 2.6: Example of a sequential merging. A video consisting of 424 initial frames is calibrated using our approach. It automatically breaks the sequence in 10 fragments, calibrates them independently and then merges them as shown in the top of the figure.

following ones. In order to avoid such errors we perform a hierarchical merging that will distribute the error more evenly over the entire sequence (see Figure 2.6)

5.5 Post-processing

Once the sequence has been completely merged, a bundle adjustment similar to the one introduced in Section 4.3 is performed to globally optimize a final solution. This is a very expensive process that is highly dependent on the size of matrices involved and ultimately in the number of different features and frames.

A common problem in tracked data is that due to noise or some spurious errors, points might be lost for a short number of frames and then found again and tagged as different measures, which will introduce extra columns in the measurement matrix. Another common problem occurs when the camera path revisits parts of the scene and several features are found again and misclassified as different points. When a 3D

reconstruction of the scene is obtained, most of this misclassified points generate very close 3D points than can be easily merged.

We perform a measurement data compression by locating those reconstructed 3D points pairs that have the following properties

- An Euclidean distance between them below a threshold, usually a low percentage of the maximal distance in the scene.
- They do not present overlapped measurements. If a pair of points are very close but are different, most likely they will be seen in the same frames so they have different entries in the measurement matrix in some common frames. This points have been simultaneously detected and they do not have to be merged since they are different features, regardless how close they are.

Once all the 3D common points have been located, a new smaller measurement matrix is formed and a final Levenberg-Marquardt adjustment is computed over the complete sequence to reduce the overall error.

6 COMPLETE CALIBRATION ALGORITHM

Below (Algorithm 2.3) we present the complete calibration algorithm in order to summarize the different techniques and the sequence of steps necessary to perform the calibration of a sequence. Also some results are presented showing how the proposed method behaves under noise

In Figure 2.7 an example of calibrated sequence shows in 3D space the location of the different keyframes as well as the points recovered during the process. It is

```
begin
   start = 1, end = start + 1
   while end < totalframes
      do
         ratio = (start/end common points)/(start total points)
         end++
      while ratio > threshold_points
      k = start
      for i= start+1 to i = end
         calculate robust homography between k and i
         ratio = inliers/outliers
         if ratio < threshold_2D
            assign i as keyframe
            k = i
         end
      end
      calculate robust projective reconstruction [start...end] -> P,X
      calculate metric reconstruction P^M, X^M
      bundle adjustment
      start = end;
      end = start+1
   end
   while nFragments > 1
      pick hierarchicaly two fragments
      set both references as the identity
      find common points
      calculate scale+translation transformation
      apply transformation to both subsequences
      merge matrices W, P, X
      nFragments --
   end
   global bundle adjustment
end
```

Algorithm 2.3: Complete solution for calibration of long sequences of images with presence of occluders.

important to notice that due to noise related errors during calibration or because the set of features used in each of the subsequences is different, small disparities in the focal length between the two overlapped frames of two consecutive sequences can appear.

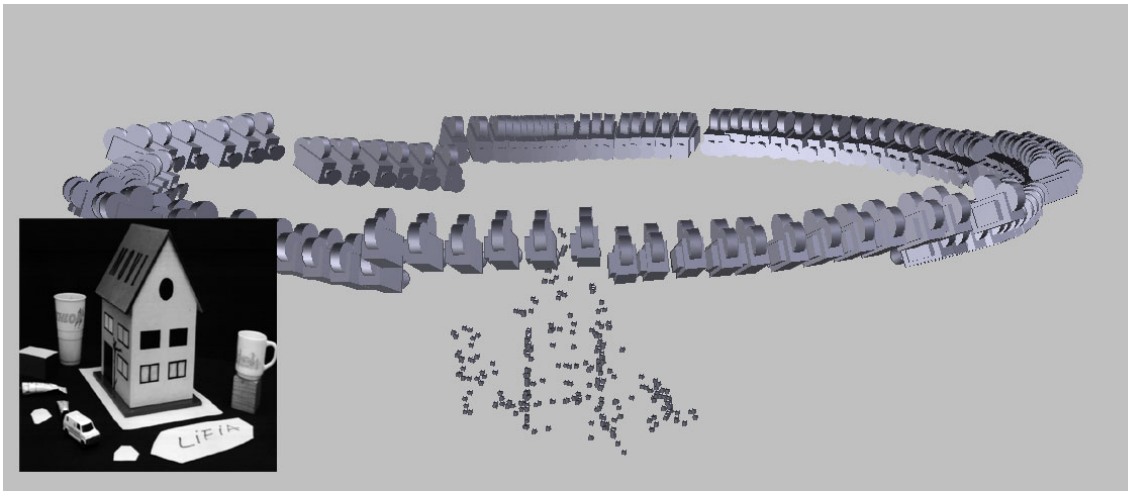


Figure 2.7: Example of sequence calibration. It is easy to identify the different frame fragments determined during the calibration process.

This does not affect the overall reprojection error since it is a reconstruction fully compatible with the input data.

In the case that a correction was necessary, a recalibration of one of the subsequences could be performed using the desired focal length as a constraint. However for the purpose of the work, reconstruction of 3D models form image base rendering, this is not required since the 3D reconstruction of the features is accurate enough and the reprojection error is small.

7 CONCLUSION

Camera calibration is a critical problem in applications such as augmented reality and image based model reconstruction. When constructing a 3D model of an object from an uncalibrated video sequence, large amounts of frames and self occlusions of parts of the object are common and difficult problems.

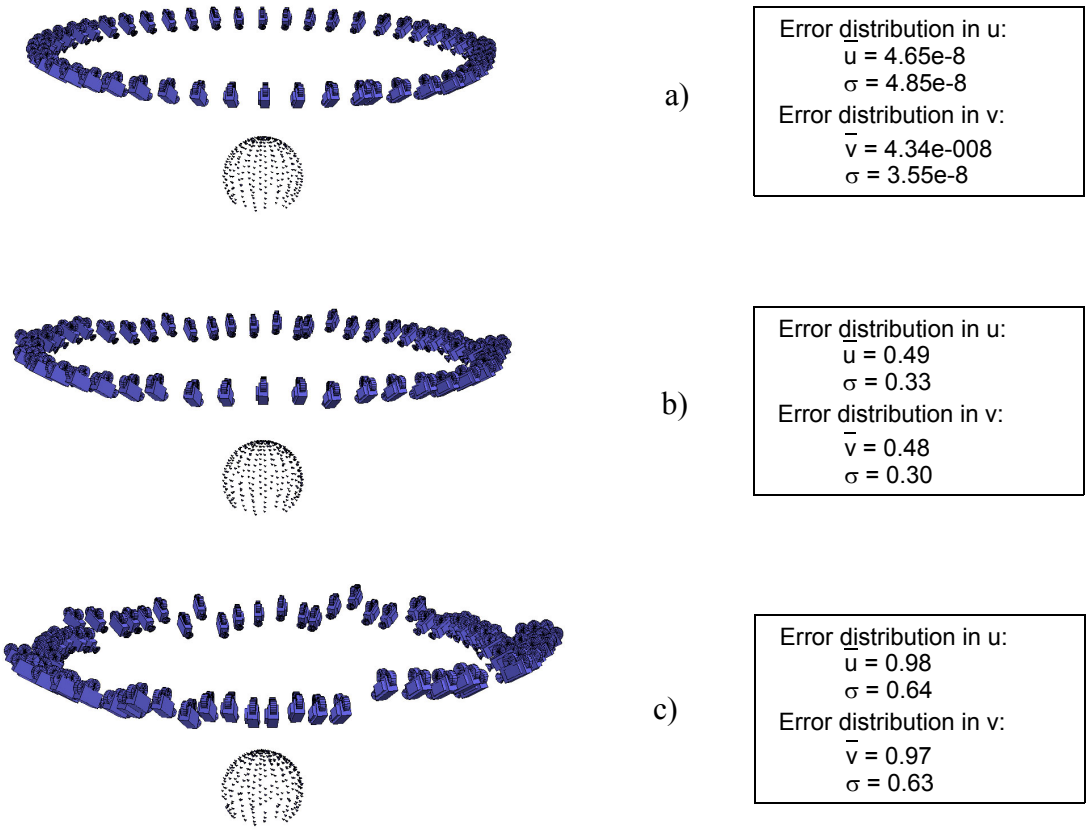


Figure 2.8: Example of sequence calibration with occlusions. In this case we use the hierarchical fragment decomposition of a synthetic sphere under different noise conditions. Case a) is the original data with no noise. Case b) shows the same set up model with random uniform noise of 2 pixels of amplitude. Case c) is the same as before with 4 pixels of noise amplitude. All the mean errors and standard deviations are calculated as 2D reprojection errors and are expressed in pixels.

In this chapter we have given an overview of the different techniques available as well as their limitations. Also we have presented in detail the proposed solution, a fast and robust algorithm that uses a divide and conquer strategy to split the video sequence into sub-sequences containing only the most relevant frames. Then a robust stratified linear based algorithm is able to calibrate each of the subsequences to a metric structure and finally the subsequences are merged together and a final non-linear optimization refines the solution.

CHAPTER 3 SCENE RECONSTRUCTION

1 INTRODUCTION

The determination of three-dimensional information form two-dimensional digital images is a fundamental task in many research areas. Traditional monocular and stereo vision methods have been successful in extracting 3D information in a variety of fields such as robot navigation and visual inspection. The original monocular or two view techniques have evolved in robust multi-view methods that make possible to recover more complete 3D information from a specific scene.

On the other hand, volumetric data representations, initially introduced in the 70's in the context of 3D medical imaging, have gained importance due to the exponential growth of storage and CPU power of computers. These representations are today a valid alternative for many applications in computer graphics, scientific visualization and computer vision. Specially medical imaging and IBMR researchers have developed

multiple approaches to obtain volumetric reconstructions from images, competing directly with traditional stereo techniques.

A brief description of different solutions for automatic multi-view 3D scene reconstruction available in the literature are presented together with the proposed solution.

2 SINGLE AND MULTI-VIEW STEREO VISION

Stereo vision reproduces the human visual system by using two cameras mounted on a calibrated rig and separated a distance known as the *baseline*. The two cameras take simultaneously a picture of the scene generating two *reference views*. Given these two photographs the stereo vision algorithm can be separated into two parts: correspondence and reconstruction.

The correspondence problem consists in identifying for each scene element in one image, the matching scene element in the other image. The correspondence algorithms can be classified in (1) *correlation-based* methods, which establish correspondences for each pixel in the stereo pair, and (2) *feature-based* which reduces the number of scene elements to a sparse set of image features such as points or lines.

Correlation-based methods define a small window around a pixel in the first image and attempt to find the corresponding match in the second image by sweeping a window over each pixel and calculating a similarity measure. The two most common measures are the cross-correlation similarity and the sum of square differences (SSD). The maximum value of these measures between a pixel of the first image and all possible

pixels in the second image defines where the correspondence is located. The spatial difference of two corresponding pixels can be stored in a disparity map.

Feature-based methods match image features such as lines, edges and corners being points the most commonly used. Since such features do not occur at every pixel of the images, these methods find correspondences over a sparse set of pixels on each image.

The goal of the reconstruction problem is to determine the location and structure of the surfaces seen by the stereo rig using the correspondence information. Once the correspondence has been established, and depending on the degree of calibration of the cameras, different reconstructions are possible:

- **Full calibration**. In this case the intrinsic and extrinsic parameters are known for both cameras. It is possible to triangulate the correspondence pairs and obtain its location in 3D space. Using such a method a disparity map can be converted into a depth map with unambiguous values. If a feature-based correspondence analysis has been performed, only the 3D values of the features can be recovered.
- **Strong calibration.** If only the intrinsic parameters of the cameras are known, it is possible to estimate the extrinsic parameters up to a scale factor using the correspondences. The reconstruction can be used for new view synthesis, where this scale factor is inconsequential.

In order to reduce the computational cost of the correspondence search, a common technique is to *rectify* [FuTV97] the images in order to align the epipolar lines for both views. This reduces the 2D search space to one dimension improving the overall algorithm efficiency.

The principle used in stereo reconstruction can be easily extended to a multi-view approach where more than two cameras are used to increase the robustness and depth resolution of the disparity map estimation. There exist different configuration of cameras, and they can be classified based on the baseline:

- **Small baseline stereo.** View points are relatively close compare to the average scene depth. This configuration usually occurs with camera array rigs or with video sequences where the spatial variation between two consecutive frames is small. The main advantages of these configurations is that it is very easy to obtain correspondences and occlusions are very small. However, depth estimation becomes very sensible to noise in the disparity maps.
- **Wide baseline stereo.** This configuration is used when a set of still photographs is taken from very different viewpoints. Since the views are not similar it becomes very difficult to estimate correspondences and the occlusion areas between views are large. On the other hand, triangulation from disparity becomes very robust and provides better depth resolution.

In [Poll99], Pollefeys presents the *multi-viewpoint linking* method that combines the virtues of small and wide baselines. Corresponding image points are concatenated by tracking the correspondences over adjacent image pairs. The result is a very dense depth map consistent with all reference views that is meshed and textured with one of the original images.

The major challenge of the disparity based depth reconstruction is determining accurate correspondences. Despite the variety of proposed enhancements, the

correspondences are typically noisy and inaccurate which has a direct impact on the quality of the final model. Moreover, since the correspondences have to be evaluated between all images, the computation time tends to be very large compared to other approaches.

3 VOLUMETRIC RECONSTRUCTION

All volumetric reconstruction algorithms assume a discrete and bounded 3D space containing the scene to be reconstructed. Typically the initial reconstruction volume is divided into voxels and the task is to correctly classify the set of voxels that represent the different objects contained in the scene.

All these algorithms require a set of calibrated input images, and some of the approaches additional classification of the pixels in background/foreground. A common assumption is that the objects contained in the scene are nearly Lambertian so they reflect light equally in all directions.

The following subsections present a review of some of the most significant methods based on volumetric reconstruction.

3.1 Volumetric intersection

Volumetric intersection algorithms reconstruct the surface and interior space of an object using its silhouettes from the different reference views. The process is performed by tracing rays from the center of projection of each camera through the contour of the object projection in the corresponding image plane. The resulting bounding volume is the reconstructed scene.

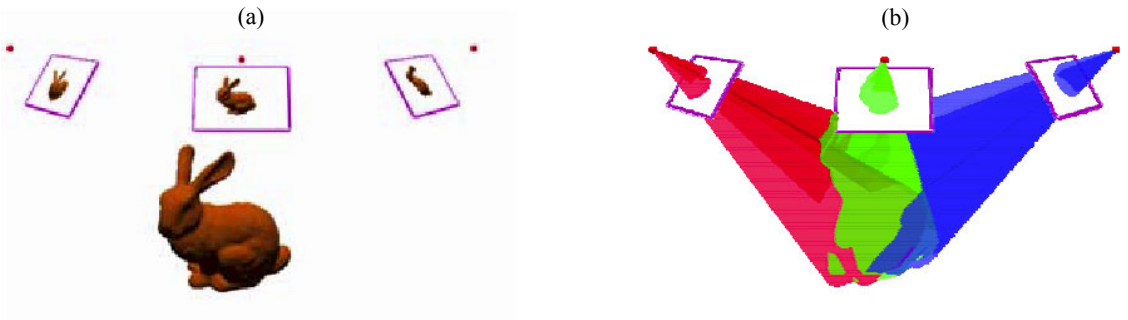


Figure 3.1: The inferred visual hull. In (a), photographs are taken of a scene. In (b), the foreground regions of these photographs are back-projected into 3D space. Their intersection forms the inferred visual hull.

The first image based volumetric model reconstruction algorithms were introduced in the mid 80's, and used a *visual hull* [Laur94] to approximate the objects (see Figure 3.1). The visual hull can be described as the maximal shape that generates the same silhouette for every reference view outside of the object's convex hull.

Beneficial properties of the visual hull are:

- It is guaranteed to enclose the real object.
- Depending on the number of views and the geometry of the object, it can be a tighter approximation than the convex hull.
- The size of the visual hull decreases monotonically with the number of views of the object, although there is no guarantee that it will converge to the physical object.

An application example is the one presented by Szeliski [Szel93] proposing a method that starts with a low resolution voxel space that is carved and refined iteratively. The reference views are segmented into background and foreground and each candidate voxel is projected onto each of the views. If the voxel footprint, that is, the set of pixels where the voxel projects, is in the background in any of the images, the

voxel does not belong to the object and is carved away. If the voxel footprint is within the foreground in all the reference views, the voxel belongs to the object and is kept. In the case that the voxel is partially in both foreground and background, the voxel is labeled as ambiguous and it is subdivided and recursively evaluated. The result of the volumetric reconstruction is an octree containing both interior and surface voxels. From the reference views a color can be assigned to each of the surface voxels and any new view can be synthesized from arbitrary viewpoints.

Although there are many great contributions in volumetric intersection methods, recent advances by Matusik *et al.* [MBR+00], describe an efficient real-time image-based approach to compute and shade visual hulls from silhouette image data. Taking advantage of epipolar geometry and incremental computation they achieve a constant rendering cost per rendered pixel. In a later work [MaBM01], the same authors present new algorithms for creating and rendering visual hulls in real-time where an exact polyhedral representation for the visual hull is computed directly from the silhouettes.

Volumetric intersection methods are fast and simple algorithms but effective at reconstructing multi-view scenes. However they have two main limitations. First, it is required to have segmented reference views, which usually involves a manual preprocessing of the set of the input images, unless the background is constant and an extra reference view without the object can be obtained. Second, volumetric intersection methods are unable to capture surface concavities unless they are visible in the silhouette of the reference images.

3.2 Voxel Coloring

Points on a Lambertian surface have the property of being *color consistent*, that is, the color property remains constant regarding the location of viewpoint and the light sources. Exploiting this property, Seitz and Dyer [SeDy99] presented a seminal method that can reconstruct scenes with sufficient color variation starting from a volume of voxels encompassing the scene. This method, *Voxel Coloring*, works by testing each voxel on the surface of the volume for color consistency among all the images in which it is visible. The color consistency test consists of projecting the voxel to all the reference images and performing a similarity test of all the projections. If the voxel color value is similar in all the images the voxel is said to be consistent and it is kept. Otherwise, the voxel is rejected and removed from the volume. The algorithm stops when all the remaining voxels are consistent with the images.

Inspired by Seitz and Dyer's method [SeDy99], a set of different voxel coloring algorithms has been proposed by Kutulakos and Seitz [KuSe98], [KuSe00], Culbertson, Malzbender and Slabaugh [CuMS99], [SCMS01], Eisert, Steinbach and Girod [EiSG99], [SeGE00].

A key common component of all the voxel coloring approaches is the determination of which pixels of each reference image corresponds to the projection of a given voxel, taking into account visibility occlusions from other voxels. The calculation of this voxel *footprint* is performed several times during the process and must be done efficiently. To overcome this, Seitz and Dyer impose constraints on the camera locations such that the scanning of voxels is done in increasing distance to the cameras. To avoid evaluating

voxels that are occluded by consistent voxels previously evaluated, a mask image is kept per reference view storing which pixels of a specific view have been validated.

Overcoming the camera placement limitation, Kutulakos and Seitz [KuSe98] presented *Space Carving*, an implementation of *Voxel Carving* with no camera constraints. This approach uses multiple scans, typically across the positive and negative axes of the oriented voxel volume. Each scan is forced to be near-to-far relative to the *active cameras* at each iteration. Occlusion maps are kept to avoid the redundant evaluation of occluded voxels. The algorithm presented in [KuSe98] is very effective, but the voxel consistency is only calculated with a subset of all the cameras that can see the voxel. A revised version of space carving [KuSe00] describes additional steps in the algorithm to compute the exact voxel visibility using all possible cameras.

Another approach by Culbertson *et al.*, *Generalized Voxel Carving*, described in [CuMS99] presents an efficient implementation of the unconstrained voxel carving algorithm. The authors implement two different approaches that use buffers to store surface voxel visibility for each pixel. In addition the Z-Buffer algorithm is used to speed up the carving process by rendering the voxels depth sorted onto the images. This approach obtains very good results, at the cost of performance and rapidly growing data structures.

Eisert, Steinbach and Girod [EiSG99], [SeGE00], proposed *Multi-hypothesis Voxel Coloring*, which is quite similar to *Voxel Coloring* and *Space Carving*. The advantage is that instead of evaluating a voxel in each image with the additional cost of visibility estimation, all the voxels are evaluated at once for every image, establishing a color

hypothesis for each of them, and then comparing all the hypotheses obtained per voxel at the end. The trade-off of avoiding the visibility computation is that it is necessary to maintain hypotheses for all voxels (interior ones included), increasing the number of hypothesis evaluations.

A more extensive survey of the different carving methods is presented in [SCMS01]. While the different approaches to the voxel coloring problem suggest the use of some sort of hardware acceleration to improve runtime performance, none of them have extensively explored the possibilities that graphic processors provide.

3.3 Surface Reconstruction

The basic problem of stereo, regardless how many cameras are used or how they are positioned, is to find the depth value of the 3-D scene point seen at each pixel of a base image, using other images as references. To accomplish this, for each pixel in the base image, its corresponding pixels in the reference images need to be identified and then by triangulation a corresponding 3D point can be obtained.

In some recent works ([FaKe98]), instead of computing the depth for individual pixel, a *depth surface* is adjusted based on constraints of the multi-view correspondences or color information. The problem is then formulated as a constrained variational optimization problem. The few existing results show good quality, however it is not clear the convergence of the optimization solvers nor the run-time and memory usage of the techniques.

Using this approach Zhang *et al.* [ZDSS01] are able to deform a 3D grid by manually setting curvature constrains and generate a reconstruction from a single image.

The optimization solver is enhanced achieving interactive evaluation of new constraints. However, it is not clear how additional images could be used to automate the process.

Later, Zhang *et al.* [ZaSe01] present a novel approach using a generic initial mesh surrounding the object and deforming it using color consistency as a source of deformation magnitude. The mesh is subdivided as it converges to the object to be modeled in order to maintain a certain triangle size. The final output is a texture-mapped triangular mesh that matches all the input images. There are however several open questions open such as the convergence time, generation of non overlapping triangles and the quality of the triangulation in the final meshes.

4 PROPOSED SOLUTION

The proposed approach for extracting a 3D volumetric representation of an object from a set of unstructured images is inspired in the *Space Carving Theory* by Kutulakos and Seitz [KuSe00]. The developed techniques combine a real-time visual-hull estimation with a carving approach as a refinement of the resulting volume, and also exploit extensively the OpenGL API, greatly supported in hardware by most video cards, to achieve shorter computation times. In the course of this work several implementations have been developed and tested with different types of iterative structure and different degrees of success in the results obtained.

In [SaBS02a] we presented a first attempt based in the space carving approach in [KuSe00]. In this case, the main iteration consists of six progressive carving steps by sweeping a plane, the *carving plan*e, along the positive and negative direction of each axis of the bounding box that contains the object to be reconstructed. At each iteration

step only the cameras visible to the carving plane are used to test which voxels in that plane will be kept. We use an octree data structure to keep track of the set of consistent voxels throughout the carving process which implies that instead of using a pre-allocated volume only the root node of the corresponding octree is initialized. Then the octree is then adaptively subdivided as consistent voxels are found. This has the advantage of a less expensive method in memory terms compare to the approaches [EiSG99] and [CuMS99], that start with an initial solid voxel volume, which during the carving process will shrink to the final number of voxels. For high resolutions *(n=256* or more) the amount of memory required to store the data becomes very large (Culbertson *et al.* report in [CuMS99] up to 462Mb for a 167x121x101 voxels dataset).

This proposed technique approach is further improved in [SaBS02b], where several hardware acceleration enhancements are added to the method, as well as an efficient image-based visual hull calculation pre-process.

The final adopted solution follows more strictly the generic space carving algorithm and reduces the plane sweeps to a single direction, and iteratively sweeps the plane until no voxel is removed. The following subsections present (1) the main underlying theory of the space carving, showing the different theorems and principles that guarantee convergence, and (2) the final adopted solution details of the different enhancements required to produce a reliable and computationally efficient method.

4.1 Space carving theory

The space carving theory, introduced formally in [KuSe00], proposes a new approach to the problem of reconstructing scenes from a set of N views for the case

when (1) there is no constraint on the shape of the objects contained in the scene, (2) there is no constraint on the position of the input views, (3) no information is available of features on the input photographs and (4) there is no prior correspondence information. The main questions to address are if it is possible to characterize the set of shapes that will reproduce the input photographs and what is the algorithm to perform the recovery.

It turns out that it is possible such a reconstruction when the scenes to be reconstructed can be classified as locally computable. This class of scenes includes those with parametrizable radiance models (e.g. Lambertian, Phong), for which the global illumination effects such as shadows, transparency and inner-reflections are negligible. Moreover when the viewpoint of each image is known (i.e. calibrated reference views) and the scene radiance follows a Lambertian model, there exists an algorithm that will recover photo-consistent shape of the scene.

The main advantages of this family of scene reconstruction methods are the following:

- The solution of the shape recovery from N views is analyzed independent from any algorithm, establishing the assumptions required to solve the problem as well as the intrinsic ambiguities.
- The obtained shape solution is the tightest possible bound of the scene that can be extracted from the set on N given views, regardless of the specific algorithm employed to obtain it.

- Since no constraints on the camera positions are imposed, the solution is a global reconstruction, eliminating the need of partial reconstructions and merges, enforcing a global consistency of the set of all images.
- The recovered shape is guaranteed to be photo-consistent with the input images, meaning that for viewpoints close to the reference ones, very visually accurate reprojections can be obtained.

Before introducing the proposed space carving solution, we present a summary of the main properties and underlying theorems of the theory of photo-consistent shapes. For additional theoretical background we suggest [KuSe00].

Lets define υ as an unknown shape defined by a closed set of points that occupy a volume in space. The points on the surface of the shape are contained in $Surf(\upsilon)$ and the radiance of a point $p \in Surf(\upsilon)$ in this surface is described by the plenoptic function $\Phi_p(\xi)$ that maps every oriented ray ξ passing through the point to the color of light reflected from p along that direction. The set of radiance functions $\Phi(\xi, p)$ for every point $p \in Surf(\nu)$ can be grouped together with υ to form the *shape-radiance scene description*, which can reproduce uniquely any image from any viewpoint.

Lets assume that there exists a set of N perspective projected views $I_1,...,I_N$, taken respectively from a set of locations $c_1,...,c_N$, that provide a sampling of the shape-radiance description of a given scene. Every reference view I_i partitions the set of all possible shape-radiance descriptions into two disjoint sets, the one that reproduces the photograph and the one that does not. This constraint is defined as *photo-consistency* and is formalized in the following set of definitions

Definition 1. *Let S be an arbitrary subset of R^3. A point $p \in Surf(\upsilon)$ visible from c_i is said to be photo-consistent with the image I_i if it does not project to a background pixel and the color of the projection of p is equal to $\Phi_p(\xi)$ being ξ the projection ray to the image plane. If p is not visible from c_i it will be conservatively considered as photo-consistent.*

Definition 2. *A shape-radiance scene description is photo-consistent with a reference image I_i obtained from c_i, if all visible points from c_i are photo-consistent and every non-background pixel is the projection of a point in υ.*

Definition 3. *A shape υ is photo-consistent with a set of N reference views if there exists a set of radiance functions $\Phi(\xi, p)$ of the points in υ that produce a photo-consistent shape-radiance description with all the reference images.*

The goal of the reconstruction is to obtain a valid characterization of the family of all the photo-consistent scenes with the initial set of *N* reference views. In order to do so, it is necessary to establish certain constraints on the shape of a scene

Background constraint. When an image *I* obtained from c has identifiable background pixels (pixels that do not belong to the shape to be recovered), this restricts υ to the cone defined by *c* and the non-background pixels of *I*. Given a set of *N* images, this constraint defines a useful volume in space where the shape υ is contained. As seen before, this is know as the *visual hull* [Laur94]. This constraint only exploits the information about the background pixels and it is very powerful when such information is available.

Radiance model. When certain restrictions are considered on the radiance of the scene it is possible to achieve photo-consistent scene reconstructions that are subsets of the visual hull, but unlike it, they can contain concavities (as the example shown in Figure 3.2). To model this restrictions lets define the following consistency criteria

- There exists a concrete method $consistency_k( )$ that takes as input $k \leq N$ colors $col_1,...,col_k$, k vectors $\xi_1,...,\xi_k$ and the known light source positions (if non-Lambertian models are used) and determines if it is possible for a surface point $p \in Surf(v)$ to reflect light of col_i in direction ξ_i for all $i=1,...,k$ at the same time.
- The function $consistency_k( )$ is monotonic, implying that if $consistency_k(col_1,...,col_j, \xi_1,...,\xi_j)$ is true, then $consistency_k(col_1,...,col_{j-1}, \xi_1,...,\xi_{j-1})$ for every permutation set of $1,...,j$ is also true.

These criteria define a class of radiance models, called *locally computable*, where the radiance at any point is independent of the radiance of all other points in the scene. This restricts the type of scenes to reconstruct to those that do not present global

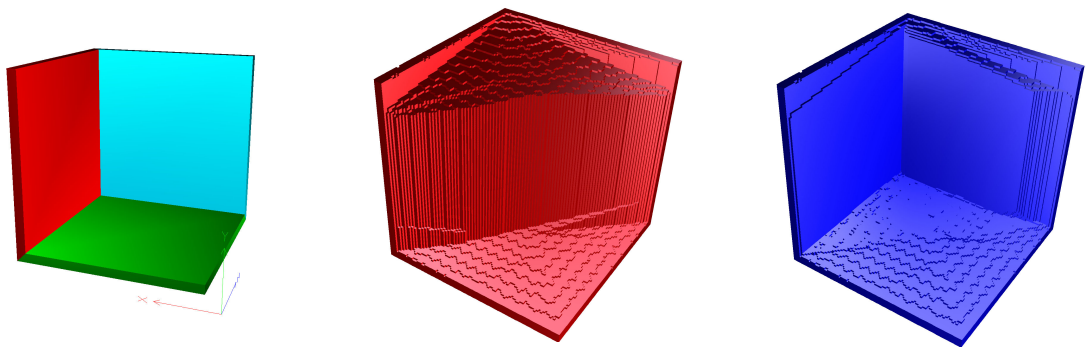


Figure 3.2: Difference between visual hull and voxel carving. A simple synthetic model like the one on the left cannot not be recovered properly since the large concavity is not detected by the visual hull, which is based in silhouettes. However with the space carving approach the empty space can be easily recovered, as shown on the right.

illumination effects such as transparency, inter-reflections and shadows. Moreover, if the light source positions are not know or cannot be modeled, the class of locally computable radiance functions will be further restricted to Lambertian scenes, with diffuse illumination only.

Given a locally computable radiance model and a $consistency_k()$ function, the photo-consistency of every point $p \in Surf(\upsilon)$ of the scene from a set on N images is fully determined, and which is more important the non-photo consistency, which tells important information about the underlying shape of the scene, is known too. Based on this, the following lemmas can be presented

Lemma 1. *(Visibility Lemma) Let* $p \in Surf(\upsilon)$*, and let* $Vis_\upsilon(p)$ *be the set of reference views in which p is not occluded by* υ*. If* $\upsilon' \subset \upsilon$ *is a shape that contains p, then* $Vis_\upsilon(p) \subset Vis_{\upsilon'}(p)$

Lemma 2. *(Non-photo consistency Lemma) If* $p \in Surf(\upsilon)$ *is not photo-consistent with a subset of* $Vis_\upsilon(p)$*, it is not photo-consistent with* $Vis_\upsilon(p)$*.*

These two lemmas (illustrated in Figure 3.3) show an underlying monotonic tendency very fundamental to this family of reconstruction algorithms: the set of

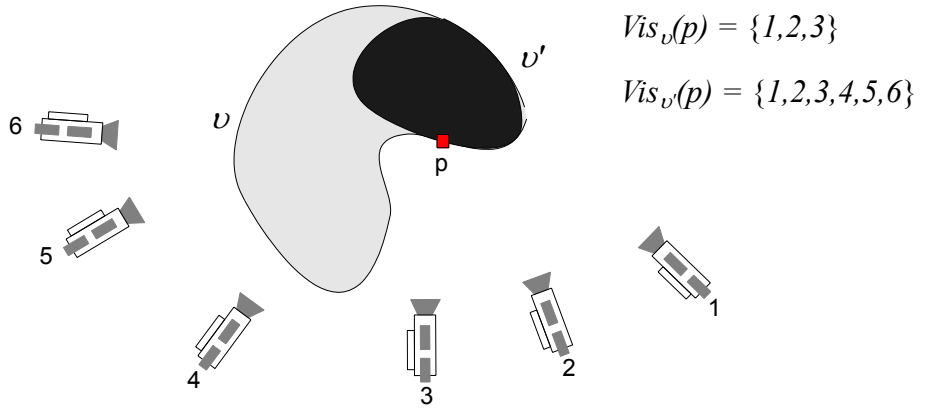


Figure 3.3: Illustration of the carving non-photo subset consistency Lemma.

reference views from which a given point $p \in Surf(\upsilon)$ is visible strictly expands as υ gets smaller, and if the initial set of reference images increases, new constraints are added on the photo-consistency, which also means that once a point fails to be photo-consistent, there is no additional reference view that can re-establish photo-consistency. This can be stated in the following theorem

Theorem 1. *(Subset theorem) If $p \in Surf(\upsilon)$ is not photo-consistent, no photo-consistent subset of υ contains p.*

These concepts can be further developed to lead to the following theorem, which basically states that for any shape υ there is a unique photo-consistent shape that contains any other photo-consistent within its volume, giving a least commitment reconstruction.

Theorem 2. *(Photo hull theorem) Let υ be an arbitrary subset of R^3. If υ^* is the union of all photo-consistent shapes in υ, every point on the surface of υ^* is photo-consistent. The set υ^* is called the photo hull.*

When we have a finite scene that can be contained in a discretized volume, these set of properties can be used to define a generic algorithm that will compute the photo-hull by iteratively removing portions of the initial volume υ until it converges to the photo-hull. The different implementations that follow this procedure belong to the family of space carving algorithms. In order to specify a useful algorithm we need determine an answer to the following questions:

- How can be determined an initial volume υ.
- What representation of that volume can be used to facilitate the carving.

```
begin
   Initialize υ to a volume that contains the original scene
   for each voxel v ∈ Surf(υ) do
      Project v to all reference views in Vis_υ(p).
         Let col_1,...,col_j be the pixel colors where it projects.
         Let ξ_1,...,ξ_j be the projection rays of v and the optical
         centers of the reference views.

      Determine the photo-consistency using
         consistency_k( col_1,...,col_j, ξ_1,...,ξ_j)
   until a v non photo-consistent is found
   if a non-photoconsistent voxel if found, set υ = υ - {v} and
      return to beginning of the for loop.
   else set υ* = υ and terminate.
end
```

Algorithm 3.1: Space carving algorithm

- How the carving process is carried in each iteration to guarantee the convergence to the photo-hull.
- What conditions define the end of the carving process.

The generic solution proposed by Kutulakos *et al.* consists of defining an initial finite volume υ that contains the scene to be reconstructed. Then, assuming that the radiance of the scene follows a proper model and it exists a well defined consistency algorithm, the volume can be discretized in a collection of voxels $v_1,...,v_m$ in such a way that the carving iteration eliminates one voxel at a time from υ. The algorithm can be outlined as shown in Algorithm 3.1.

The space carving algorithm requires a number of photo-consistency tests that is upper bounded by *M x N*, where *M* is the number of reference views and *N* is the initial number of voxels in the uncarved volume.

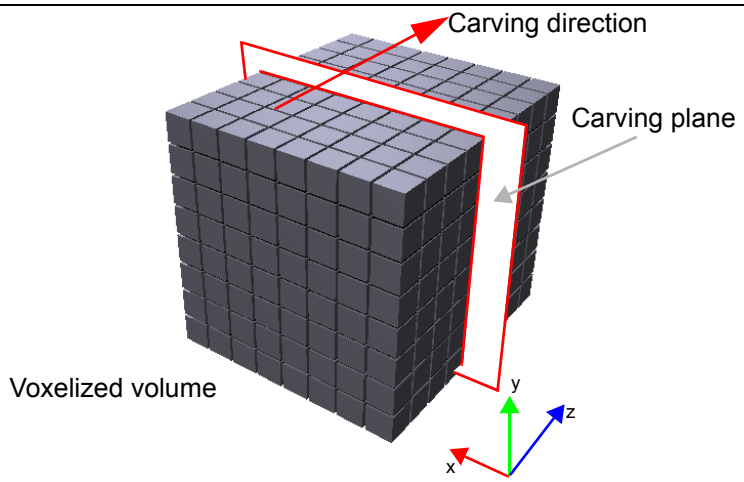


Figure 3.4: Standard configuration for a plane sweep carving algorithm. In this case, Z is the sweep direction of the carving plane

4.2 Our approach

The proposed implementation of the space carving methodology consists of an iterative algorithm that has an outer loop that performs a progressive plane sweep along one axis of the bounding box that contains the object to be reconstructed (see Figure 3.4) until the set of non-carved voxels remains constant.

On each iteration of the sweep, only the voxels that belong to the surface *Surf(υ)* of the volume slice are processed. Then, during the color consistency test, each voxel *v* is tested against each of the views to determine the set of contributing reference images $Vis_{\upsilon}(v)$. This is performed by precalculating a set of occlusion maps for each of the views for every outer loop iteration. Once $Vis_{\upsilon}(v)$ has been determined, the voxel footprint for every reference view $I_i \in Vis_{\upsilon}(v)$ is calculated and the color consistency function decides whether the voxel is consistent and then kept, or is inconsistent and removed. Since the slices are calculated each time, but the occlusion maps are only updated after a complete sweep run, to avoid modifying the visibility by deleting non-

```
begin
   Background segmentation using the visual hull
   do
      Generate occlusion maps
      totalCarved = 0
      for i = 0 to i = maxSlice do
         find surface voxels on slice i
         for each surface voxel do
            for each reference view do
               project voxel center to image plane
               if voxel is visible and not occluded then
                  store footprint information
            end
            evaluate voxel consistency
            if voxel is not consistent then
               mark voxel for deletion
               totalCarved ++
            end
         end
      end
      delete voxels
   while(totalCarved > 0)
end
```

Algorithm 3.2: Proposed implementation of the carving method

consistent voxels, the deletions are delayed until the end of the outer loop iteration, before the update of the occlusion maps. The pseudocode for the algorithm is presented in Algorithm 3.2.

The main data structure used to store the voxels of the volume to be carved is a three dimensional bitmask, encoding only if a voxel is present or not. The main advantage of such structure is the memory footprint at run time: for a maximum subdivision of 512 voxels per side of the box containing the object, the bitmask has 512 *x* 512 *x* 512 bits which adds up to 16Mbytes of storage. The data structure can be constructed from a contiguous set of standard types such as integers that sum up to the required bit count, or

in a C++ environment, we can use the bitset template and avoid the implementation of the logic for accessing and testing individual bits, which is already supported by the standard template.

It is easy to see that the algorithm presented in Figure 3.2 is compliant with the generic space carving method. Given a set of voxels in the surface, and using the remaining voxels interior ones as occluders, each voxel is examined and tested for consistency. If it is consistent it will be left and re-examined in further iterations until it is deleted or it remains as part of the reconstructed object. Following the conservative approach of this family of methods, in case of doubt, the voxel is left, expecting that later on its situation will become better defined.

In order to achieve a reasonable computational and memory costs, we have design and implemented a set of enhancements that are presented in the following subsections.

4.2.1 Occlusion map generation

In the proposed algorithm, we want to evaluate the voxels $v \in Surf(\upsilon)$ in the surface of the current shape υ taking into account the occlusions that the inner solid volume can produce on the voxels for some of the views. Kutulakos and Seitz use a bitmask for each reference view and mark the pixels already used in consistent voxels during the previous sweeps so they act as occluders. In [SaBS02a] after rendering all the voxels of the carving plane of a reference view, we render all the consistent voxels of previous iterations in black color occluding those areas of the image already assigned (see Figure 3.5b). In [SaBS02b], since the approach works by projecting the reference images onto the carving plane, the previous consistent voxels should partially occlude

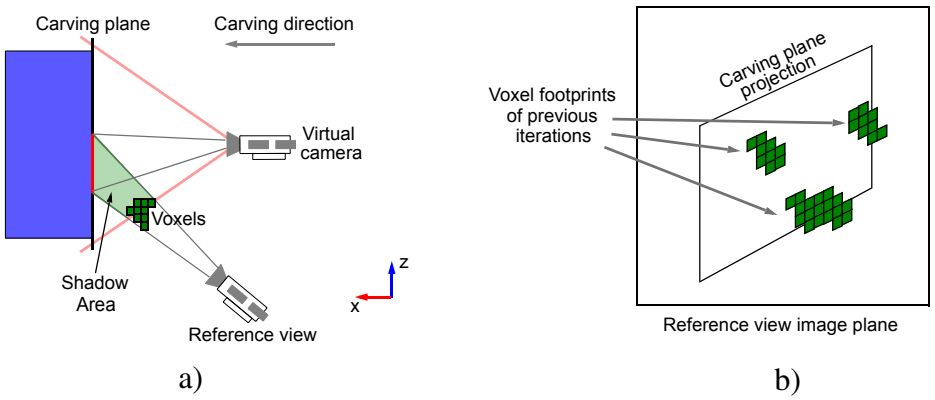


Figure 3.5: Shadow projection of previous consistent voxels.

these projections, generating a shadow on the carving plane (see Figure 3.5a). To achieve such effect the set of accumulated voxels are rendered in black using the planar projected shadow method described in [Blin88].

In the final proposed solution, we consider another way of determining if the current voxel *v* object of study is visible from a given reference view *i* inspired on the standard computer graphics z-Buffer algorithm.

When the surface voxels are known, they can be removed from the volume generating an one-voxel eroded version of the scene. Then, this can be used to generate a depthmap from each of the reference views. To determine if voxel *v* is visible from reference view *i*, one can project the center of the voxel to the camera coordinate system and do a simple depth test for visibility determination: if the voxel depth from the reference view is larger than the depthmap value at that projection, the voxel is occluded by the active volume of the scene. Otherwise the voxel is visible and we can proceed to determine its footprint for that image.

Since we are using a 3D bitmask to store which voxels are object and which ones are empty space, it is very simple to determine the set of voxels on the surface by using a 6-neighbors test and selecting those active bits that have at least one face to face contact with an empty space bit. This algorithm is $O(n^3)$, being n the size in voxels of the carving space. However n rarely exceeds 512 elements, being 256 a realistic upper bound of it, which keeps the real complexity of the algorithm low.

This has the advantage of the simplicity, since the depthmaps have to be determined only before a complete iteration of the sweeping plane and then the visibility determination involves a product of a vector by a matrix in order to perform the voxel depth calculation and a comparison with the current depthmap.

However the one-level eroded approach has a main drawback, specially in cases such as when a set of voxels are lying in a plane almost perpendicular to the viewpoint. In such case, the voxels closer to the center of projection should occlude the ones that are further back, which in the case of the eroded surface is not true.

To overcome this problem, we propose to use the *ε-z-buffer* test concept similar to the one presented in [PaSG03], which consists of generating a depthmap with an offset applied while rendering the geometry. This offset is a displacement ε along the perspective projection ray of the vertices of the geometry, that can be set to the semi diagonal of a voxel to guarantee that a visible surface voxel center is in front (closer to the viewpoint) of the depthmap. Figure 3.6 illustrates the results that can be achieved with these different approaches.

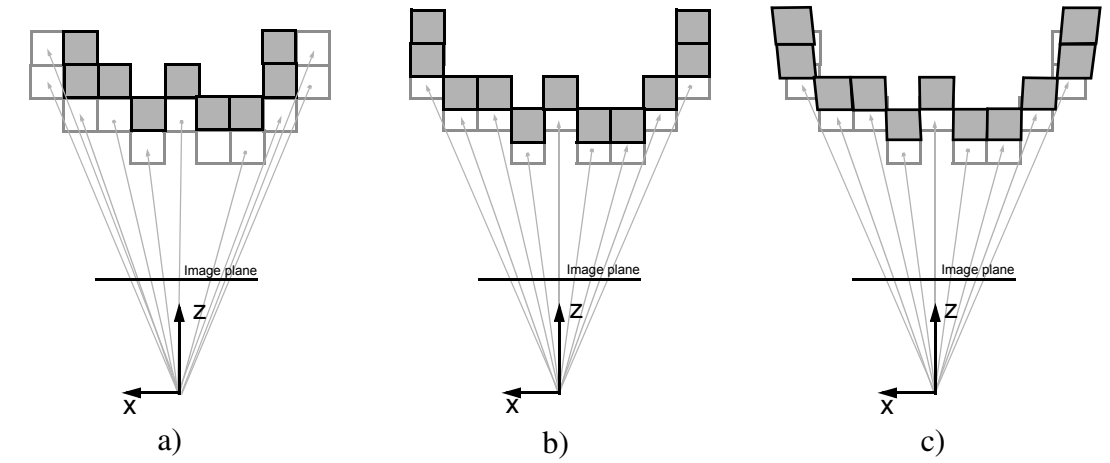


Figure 3.6: Occlusion map generation. The different approaches use a) one level erosion, b) zbuffer offset and c) ε-z-buffer

In order to exploit hardware acceleration the generation of the depthmaps can be performed using the OpenGL API by determining which faces of voxels have to be drawn and then capturing the depth buffer from the video card memory. Furthermore it is important to notice that the *ε-z-buffer* can not be achieved using OpenGL by merely translating the surface using the depth offset functionality. The only way to perform a correct perspective translation along the projection rays is by using the latest video hardware with support for vertex programs and actually calculate the displacement using the projector of each vertex.

4.2.2 Determination of surface voxels in slice

Once the inner loop starts, the first step in each iteration is to determine which voxels lie on the surface of the actual slice. This can be achieved following the same approach as for the determination of the complete set of $v \in Surf(\upsilon)$. Obviously the computational cost is $O(n^2)$ with n always lower that 512.

The set of voxels determined in this step will be considered for footprint determination and color consistency evaluation. However, there might be some of these

voxels that have already been evaluate in previous iterations, and even if their visibility set has not changed, they will be reevaluated with the consequently computational cost.

As we have seen in the presented theory, if a voxel has been determined consistent for a set of k images, it will remain in that state unless more reference views are added where the voxel shows to be inconsistent (i.e. the principle of monotonic growth of $Vis_{\upsilon}(v)$).

Therefore, if we have available a bookkeeping of which is the set of images where a voxel is visible, then there is an extremely fast test that defines if a voxel needs to be reevaluated or not. Moreover, as the algorithm converges to the final shape υ^*, more voxels have a constant set $Vis_{\upsilon}(\upsilon)$ and the convergence speed increases, since the consistency is performed less times per iteration.

4.2.3 Footprint calculation

During the carving process, the consistency test evaluates the color values of the footprints of each voxel of the carving plane for each of the active reference views. The common approach to obtain the footprint consists in rasterizing the voxel on the reference view. As proposed in [SaBS02a], this can be improved by projecting simultaneously all the voxels of the carving plane by rendering a set of quadrilaterals, each of them with a unique color ID that identifies to which voxel it belongs. The result will be a colored mask that indicates to which voxel belongs each pixel of the reference image. However, when the voxel size is small and for large angles between the reference view and the normal of the carving plane, some voxels might be projected to the same pixel locations, overlapping each other and distorting the footprints.

In [SaBS02b] we propose to use a different approach consisting on projecting the reference image on the carving plane using projective texture mapping (see Figure 3.7b). This approach has two advantages, (1) each voxel presents the same constant footprint for each view, facilitating the registration of the footprints and statistics calculations and (2) since OpenGL interpolates the textures, subpixel accuracy is reached at no extra cost.

However, since a single face of the voxel is viewed at each sweep iteration, the overall method requires 6 passes (on in each direction of the axis) before a consistency decision can be made, and that introduces additional problems of bookkeeping the information to be used in a delayed consistency check.

In the final proposed approach, we consider the voxels to be spherical so the footprint of a voxel in any reference image is identical. Moreover we will assume an average radius per voxel that will be calculated at the beginning of the iterative process.

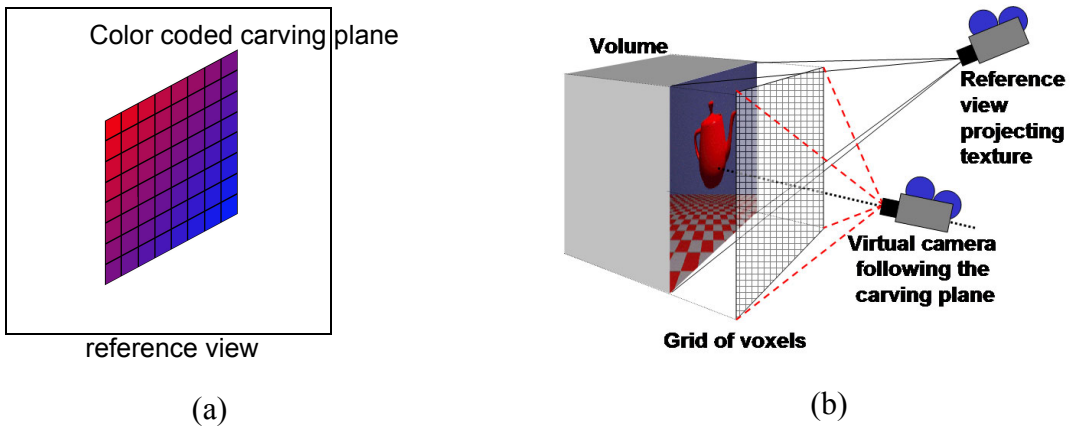


Figure 3.7: Two ways of calculating the footprint of the voxels. a) shows a color coded rendered carving plane projected to a reference view image plane. b) shows the scheme of a virtual camera perpendicular to the carving plane and the reference view is projected onto the plane.

In this way, once a voxel has been determined to be visible, a disc of a fixed radius and centered on the projected center, will be used as a footprint.

Although this introduces some more samples than the real footprint, the consistency test has been designed to take this into account. Heuristically, this solution has proven to be statistically correct even for highly textured surfaces.

4.2.4 Color consistency criterion

The original voxel coloring algorithm by Seitz and Dyer [SeDy99] considered only the color of the projection of the voxel center for the consistency test. Unfortunately, using a single pixel color per voxel, important information about the color distribution of a footprint is lost, making the method sensible to the sensor measurement variability. In [SaBS02a] we propose to use a statistical criterion based on the ANOVA technique to determine if the different footprints of a voxel can be assimilated to the same color distribution. However when voxel resolution is large, the footprints present a small area in pixels and the low number of samples causes the statistical analysis to be inaccurate. An alternative approach is used in [SaBS02b] where the consistency of a voxel is determined by performing a distance measurement in normalized color space of the pixels of the footprints. That is, between each image pair *i, j* we apply the following mapping

$$dR_{i,j} = \left| \frac{\bar{R}_i(X_i, Y_i)}{\overline{RGB}_i(X_i, Y_i)} - \frac{\bar{R}_j(X_j, Y_j)}{\overline{RGB}_j(X_j, Y_j)} \right|$$

$$dG_{i,j} = \left| \frac{\bar{G}_i(X_i, Y_i)}{\overline{RGB}_i(X_i, Y_i)} - \frac{\bar{G}_j(X_j, Y_j)}{\overline{RGB}_j(X_j, Y_j)} \right|$$

$$dB_{i,j} = \left| \frac{\bar{B}_i(X_i, Y_i)}{\overline{RGB}_i(X_i, Y_i)} - \frac{\bar{B}_j(X_j, Y_j)}{\overline{RGB}_j(X_j, Y_j)} \right|$$

$$dR_{i,j} + dG_{i,j} + dB_{i,j} \leq threshold$$

with $\overline{RGB}_i(X, Y) = \bar{R}_i(X, Y) + \bar{G}_i(X, Y) + \bar{B}_i(X, Y)$

The normalization of the colors by the sum of components increases the robustness in respect to varying illumination conditions between the different images.

Recently, Stevens *et al.* [StCM02] proposed a new approach to the color consistency criterion that avoids the use of a fixed/variable threshold and its associated initialization problems. Instead of grouping all pixel colors of all the views of a given voxel, a set of image pair tests are performed and a voxel is said to be consistent if for all the pairs of pixel sets, there is color histogram intersection. Lets define the set of pixels of voxel *i* viewed from image *j* as π_j^i, then the consistency test compares all non empty pairs such

$$\pi_k^i \approx \pi_l^i, \forall k, l \; k \neq l$$

In order to count for texture variability and color distributions the comparison between each pair of set is performed by intersecting their color histograms [Swai90], defining a voxel as color consistent if all the histograms of all views intersect

$$Hist(\pi_k^i) \cap Hist(\pi_l^i) \neq \varnothing, \forall k, l \; k \neq l$$

Since we are working in a RGB color space and each channel has a value between 0 and 255, the histogram is a three dimensional structure with more 16 million elements, making its storage and processing unnecessary expensive. Instead we quantize the histograms into a set of bins for each color variable reducing considerably the storage. A typical subdivision that produces good reconstructions is 8 bins per channel, reducing from 16 million to 512 elements with a size of two bytes each allowing for a worst case maximum color uniform footprint size of 256 *x* 256 pixels.

Two histograms are considered to intersect if at least one pair of corresponding bins has a non-zero count, regardless of the actual count of elements in each of them. This binary test has proven to give good results that do not justify the use of a more sophisticated and computationally expensive test. Then, one could argue that each bin could be encoded as a single bit indicating present or not present color component. From our experiments we found that it is better to perform the counting on the histograms and before performing the tests eliminate those bins that have a very small amount of contribution (less than a 5% of the total count), assuming that it is due to noise.

We extend the proposed criterion on [StCM02] by storing the 3D color histogram structure, $Hist_j^i(r, g, b)$, for each footprint *j* of a given voxel *i*, defined as

$$Hist_j^i(r, g, b) = \frac{\#(r,g,b)_j^i}{\#p_j} \qquad \text{(eq 3.1)}$$

where $\#(r,g,b)_j^i$ stands for the number of times the color value *(r, g, b)* appears in the footprint *j* of voxel *i*, and $\#p_j$ is the total number of pixels included in the footprint.

Then, instead of looking only for histograms intersections, we calculate the normalized correlation,

$$N_{j,k}^{i} = \frac{\sum Hist_j^{\,i}(r,g,b) \cdot Hist_k^{\,i}(r,g,b)}{\sqrt{\sum (Hist_j^{\,i}(r,g,b))^2 \cdot \sum (Hist_k^{\,i}(r,g,b))^2}} \qquad \text{(eq 3.2)}$$

between each pair *j,k* of histograms and count how many are above a minimum correlation value. When the ratio of high versus low correlated pairs is above a threshold, the voxel is considered as consistent. This relaxes the consistency criterion to allow for some empty intersections between histograms.

With no further modifications, this histogram consistency test fails when the incoming voxel is close to the boundaries of a bin. Assuming that our image sensor has a standard deviation for the colors of σ_0, most of the measurements of a given pixel will be in the RGB sphere centered around the color with a radius of $3\sigma_0$. Moreover σ_0 is usually unknown and might not be constant and equal in the three color channels. To account for the sensor variability and the measurement variations between two successive exposures, we redefine the histogram bins to be overlapping by a certain amount, that heuristically we set to 15% of the size of the non-overlapping bins. This will lead to have some pixels double counted in different bins enforcing the robustness of the method.

The main advantage of this histogram based color consistency is that it does not require sophisticated parameter tuning and it works for a wide range of cases from uniform colored to highly textured scenes using identical settings.

4.2.5 Background segmentation

When the set of views are taken around a target object, most of the voxels in the initial volume will be carved away as they are not contained in the object (e.g in the typical setup of a rotation platform with an object only around a 20~25% of the initial volume is occupied by the object).

This is not taken into account in most of the voxel coloring implementations, and the implication of this is that all those voxels will be analyzed for consistency and subsequently carved, increasing unnecessarily the runtime of the algorithm. On the other hand, when the reference images can be segmented into background and foreground, a volumetric intersection method could easily remove the exterior voxels. The first solution presented in [SaBS02a] consists in using an octree data structure that is initialized to the intersection volume of all the reference view frustums (see Figure 3.7a) and then shrank to only octree cells that fall partially or entirely in the object silhouettes for all the images. This solution reduces the computation time of the algorithm.

However in [SaBS02b] a more refined technique is presented, that calculates for each carving plane the corresponding section of the visual hull in real-time. Following the approach of Lok [Lok01] and using OpenGL features of projective texture mapping and stencil test, an inside/outside mask can be overimposed over the voxels of the carving plane, eliminating from the consistency analysis those voxels that surely do not belong to the object.

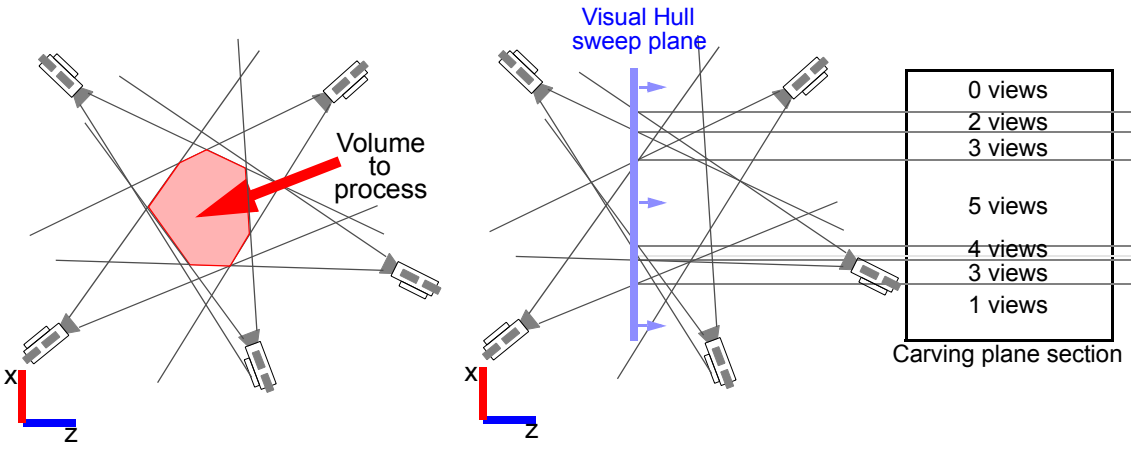


Figure 3.8: Two different ways of calculating the visual hull. a) performs a volumetric intersection of the view frusta, while b) show a plane sweeping process using stencil buffers to determine how many views see each voxel of the plane.

Similar to the proposed space carving technique, a virtual camera is located at a constant perpendicular distance to the carving plane (Figure 5). During each of the sweeps, the reference images are projected onto the carving plane using projective texture mapping and analyzed from the virtual camera. At the initialization stage, OpenGL buffers are cleared, stencil and alpha tests enabled to paint only the foreground of the images and background pixels are marked with an alpha value of 0 (fully transparent). Next, all views are rendered while the stencil buffer accumulates the number of times a given pixel has been painted. If all n views project foreground pixels to a voxel on the plane, the corresponding stencil values will be equal to n, the number of cameras, and therefore the voxel belongs to the visual hull. Then the stencil is read from the video card memory and the corresponding slice of bitmask holding the initial volume υ is updated with those voxels that belong to the visual hull.

5 CONCLUSION

In this chapter a review of the different methods for scene reconstruction from N views has been presented. Then we have focused on the family of methods based on space carving and color consistency. We have developed the underlying theory that supports and guarantees a valid space carving algorithm.

The proposed solution has been presented and compared with the seminal method as well as different approaches such as the works presented in [SaBS02a] and [SaBS02b]. The first one enhances the original space carving algorithm by adding a background segmentation, a statistical consistency criterion and an automatic calculation of the consistency threshold based on the recovered 3D measurements during the self-calibration of the reference views. The second work is a major revision that substantially reduce the computational cost of the space carving algorithm, allowing a multiple sweep merging.

CHAPTER 4 MODELING

1 INTRODUCTION

In this chapter we present the modeling stage of the proposed object reconstruction pipeline. The modeling process can be defined as the set of operations performed on the volumetric reconstruction of the scene and the reference views to obtain a representation suitable to be rendered in a computer using a specific rendering algorithm.

An immediate voxel based representation can be obtained with a space carving method consisting of a set of colored voxels. However this representation is not necessarily the most indicated for a particular rendering system. In fact it turns to be very inefficient, since each voxel requires the use of six quadrilaterals or twelve triangles, and the texture information from the different views is not taken into account, loosing one of the main components of photorealistic quality during rendering. Therefore, it is important to define a set of object representations that can be obtained

using the available information from the previous stages. Moreover, these representations must be prepared and optimized having in mind a feasible rendering paradigm that will be able to fully exploit the information to achieve great realism at interactive frame rates.

In this chapter we present a review of the modeling problem discussing different model representations.

2 MODEL REPRESENTATIONS

There are many different model representations available that use as underlying primitives very different elements ranging from images, geometric meshes to distance/ potential fields. Since the input data to this stage of the pipeline is a binary voxelized space and a set of calibrated images we will restrict our survey to those structures that can be obtained from such inputs. Moreover, the model representation is designed with rendering purposes in mind which will restrict to renderable structures using actual 3D graphics hardware.

The following subsections present different approaches to the modeling problem based on possible rendering schemes and their required primitive representation.

2.1 Image based models

The most direct representation that one can think of is to combine the reference views to generate novel renderings using the recovered camera calibration information and the position and settings of the new view. Little extra information can be added by the volumetric reconstruction except if the image based rendering algorithm requires depth information together with color information for each pixel of the reference views.

Depth information can be easily calculated by rendering the reconstructed model from each of the reference viewpoints.

The notion of depth per pixel has been introduced as disparity between images in [ChWi93] and used for image synthesis by interpolation between pairs of input images. The depth information, calculated as the distance from the center of projection along the view direction to the corresponding surface point, allows to re-project pixels from a depth-image to arbitrary new views. In [Mill95], a unique evaluation order is presented to guarantee back-to-front drawing order when (forward) warping pixels from the input depth-image to the frame buffer of a new view.

An extension to depth-images is presented in [SGHS98] called a layered depth-image (LDI) which can store multiple depth and color values per pixel. The use of a LDI allows improved depth-image warping with fewer exposure artifacts – exposure of regions not visible in the reference image. The use of precalculated, multi-layer depth-images has previously been discussed in [Max96] for rendering of complex trees. LDIs are also used in [AlLa99] together with an automatic preprocess image placement method to support interactive rendering at guaranteed frame rates, and in [OvBi99] to represent objects using a bounding box of LDIs. The idea of LDIs has further been extended in [ChBL99] to LDI-trees for improved control over the sampling rate.

Another approach to cope with exposure artifacts is to represent depth-images as triangle meshes [MaMB97]. The triangulated depth-buffer provides a connected surface approximating the 3D scene and supports automatic pixel interpolation for exposed or stretched regions as well as hardware accelerated warping by rendering the textured

triangle mesh. The approach presented in [DaCV97], [DaCV98] creates simplified irregular triangulations of depth-images used as cubical environment maps. Both approaches use multiple reference depth-images to limit exposure artifacts for new views. In [DSSD99] multiple layers of triangulated depth-images are proposed to solve exposure problems.

Images with per-pixel depth values have been used to represent individual objects [SGHS98], [OvBi99] or to approximate small parts of a large scene in interactive walk-through applications such as in [AlLa99], [ACW+99] and [AlBW01].

2.2 Point based models

The most simple geometric rendering primitive that can be used in a graphics pipeline is a colored point. One can consider the set of voxels from the reconstruction algorithm as a set of points using the center of the voxels as their coordinates. The use of points with no connectivity information gives a very low per-primitive cost models that is very useful when displaying interactively large models.

A normal vector can be assign to each point based on the nearest neighbors or the voxel connectivity when available. A color for each point can be extracted from the images by performing a weighted average of the voxel color in all the images.

Point based approaches such as [PZBG00], [RuLe00], [ZPBG01], eliminate exposure artifacts due to under-sampling and zooming by rendering points as disks, surface elements with non-zero extent. When rendered, the tightly packed surface elements appear to represent a smooth continuous surface.

2.3 Geometry based models

Polygonal based models represent the traditional approach in computer graphics: transformations, shading and texturing operations are fully supported and optimized in actual graphics hardware. These primitives are also supported in all existing 3D modeling tools from entry level to expensive professional modelers (that also support other geometrical primitives such as NURBS patches). The purpose of obtaining geometric models out of a set of views is because (1) hardware fully supports them and countless algorithms are available to operate with them, (2) to be able to import the model into any modeling package to generate complex animations and (3) there are certain image based rendering algorithms that require a geometrical proxy, usually a rough approximation of the object, to properly combine and render the reference views.

Being the start point of this stage the voxel-based reconstructed model, the first operation is to generate a polygonal mesh that adjusts to the surface of the object. This is a very well know problem in the computer graphics community and it has been studied in many applications with the purpose of extracting meshes different sources such as range data obtained from one or multiple scans of a physical object, medical imaging like CT scans, etc. Although there has been intense research in this field since the 90's a brief description of the most relevant contributions is presented.

2.3.1 Surface reconstruction from 3D spatial point data

Several methods have been proposed since the 90's to extract an interpolated or approximated surface from a set of unorganized points. The following list of approaches is not extensive but wants to illustrate the variety and richness of this research field.

Hoppe *et al.* [HDD+92] present a general method for automatic reconstruction of accurate, concise, piecewise smooth surface models from scattered range data. Their method is based on minimizing an energy function that trades off conciseness and accuracy of fit to the data, and is capable of representing surfaces containing sharp features such as creases and corners.

Turk and Levoy [TuLe94] present a method for combining a collection of range images into a single polygonal mesh ("zippering them together") that completely describes an object to the extent that it is visible from the outside. This is an incremental approach: scans are acquired and combined one at a time. This allows to acquire and combine large number of scans with minimal storage overhead.

Liao and Medioni [LiMe95] use deformable models to approximate a three-dimensional surface given by a cloud of 3D points. For the approximation B-splines are used. The user (or the system itself) provides an initial simple surface, such as a closed cylinder, which is subject to internal forces (describing implicit continuity properties such as smoothness) and external forces which attract it toward the data points.

Crossno and Angel in [CrAn97] and [CrAn99] present a method that first adjust a set of particles to the surface of an scanned object. The partial organization and information of the particles is used to extract an initial star-shaped triangular mesh of a point and its neighbors and then propagate the triangulation to the neighbors. An edge that separates the triangulated surface from the non assigned points spirals out over the surface until the edge reaches the border of the object. The authors claim a three order of magnitude reduction in the computation time compared to [HDD+92].

2.3.2 Surface reconstruction from 3D volume data

The oldest methods of volumetric surface reconstruction are contour oriented methods, which divide the volume in slices and try to find the corresponding contour region in each slice and connect the consistently oriented contour lines between adjacent slices with triangular meshes. Sometimes Delaunay triangulation is used for decision support. Sometimes ambiguities arise and user interaction is required.

A second category are the lookup methods, which use a lookup table of possible triangulations for connecting data points between adjacent slices, producing a closed polygonal surface. The simplest method is the *cuberille*-model, which define a cuberille as a cube between two data slices consisting of eight neighboring points of the dataset. The vertices can be classified as internal or external to the object's surface (see Figure 4.1). Whenever there are two different colors along an edge of the cube the surface must intersect the edge. A lookup table is define based on the 2^8 possible configurations of vertices.

A similar approach is the *Marching Cubes* algorithm (MC) [LoCl87], which consists of two primary steps to reconstruct a surface. First, the surface is located corresponding to a user-specified value and triangles are created. Second, to ensure a quality image of the surface, normal vectors to the surface at each vertex of each triangle are calculated. Marching Cubes uses a divide-and-conquer approach to locate the surface in a logical *cube* created from eight points, four from each of the two adjacent slices. The algorithm determines how the surface intersects this cube, then moves (or *marches*) to the next cube. To find the surface intersection in a cube, the value *one* is assigned to a cube's

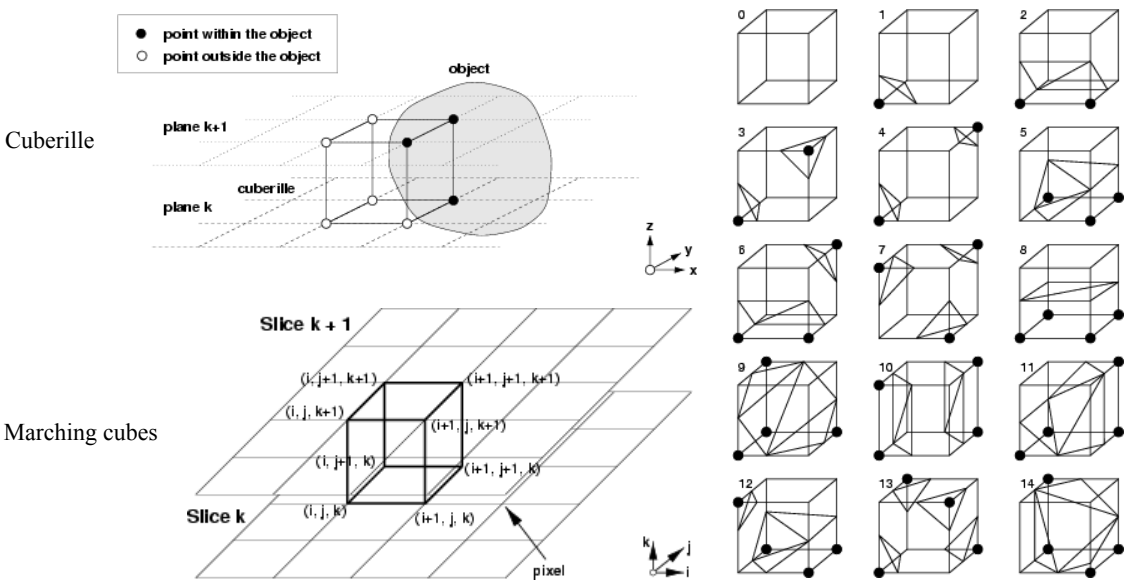


Figure 4.1: Cuberille and Marching Cubes patterns. The figure shows the Cuberille vertex classification and Marching Cubes cell determination with the 15 different basic vertex configurations. (figure extracted from [Perz97])

vertex if the data value at that vertex exceeds (or equals) the value of the surface that is constructed. These vertices are inside (or on) the surface. Cube vertices with values below the surface receive the value *zero* and are outside the surface. The surface intersects those cube edges where one vertex is outside the surface (one) and the other is inside the surface (zero). With this assumption, the topology of the surface within a cube is determined. The location of the intersection is calculated later (with linear interpolation). The 2^8 possible state of the vertices can be stored in a lookup table, which is reduced to determine 15 basic possible patterns (see Figure) and the permutation of these using complementary and rotational symmetry produces the 256 cases.

3 MODEL PREPROCESSING

Before discussing the different studied modeling representations, in the following subsections we introduce a set of preprocessing techniques used to transform the data

into more friendlier to the modeling processes. The first one is the extraction of a set of oriented and colored points from the volumetric scene representation. The second preprocessing technique is the extraction of a polygonal based mesh of the scene from the set of surface voxels recovered during the reconstruction stage.

3.1 Point data extraction

A model representation based on the use of colored 3D points can be easily obtained from the resulting voxel representation from the scene reconstruction pipeline stage. Given the list of surface voxels, we can consider the center of each voxel and its color as a basic representation. Moreover a normal calculation is extremely simple to compute using the connectivity information by adjusting a plane to the neighborhood and assigning the normal to the point.

In Section 4.2.1, Occlusion map generation (page 81), we proposed an algorithm to extract the surface of a reconstructed scene that can be slightly modified to simultaneously calculate the normal of any given surface voxel by fitting a plane on its *k*-neighbors, where *k* is the radius of influence of the normal calculation. This plane calculation can be efficiently performed by solving the following least-squares minimization [Eber01]

$$d^2(p, n) = n^T M(p) n$$

where n is the normal vector that defines the plane, p is the point where the plane is contained and $M(p)$ is the following matrix

$$M(p) = \begin{bmatrix} \sum_{i=1}^{m} (x_i - p_x)^2 & \sum_{i=1}^{m} (x_i - p_x)(y_i - p_y) & \sum_{i=1}^{m} (x_i - p_x)(z_i - p_z) \\ \sum_{i=1}^{m} (x_i - p_x)(y_i - p_y) & \sum_{i=1}^{m} (y_i - p_y)^2 & \sum_{i=1}^{m} (y_i - p_y)(z_i - p_z) \\ \sum_{i=1}^{m} (x_i - p_x)(z_i - p_z) & \sum_{i=1}^{m} (y_i - p_y)(z_i - p_z) & \sum_{i=1}^{m} (z_i - p_z)^2 \end{bmatrix}$$

This equation fits a plane to the input points in a least-squares way where the errors are the distances measured orthogonally to the plane being fitted. The solution that minimizes the square of the distance can be obtain by calculating the smallest eigenvalue of *M(p)* and the corresponding normalized eigenvector gives us the normal *n*.

3.2 Surface data extraction

Some of the proposed model representation presented later require a polygonal surface as the main data structure or as an intermediate one for further processing. We present an algorithm to extract such a surface as a triangular mesh assuming a voxelized model of the scene is available as a binary 3D mask obtained from the proposed scene reconstruction method.

The presented method is inspired in the *Surface Net* [Gibs98] algorithm and uses a similar running approach to the Marching Cubes [LoCl87] of a single progressive sweep across the volume of two planes to reconstruct a closed mesh surface. Furthermore the Surface Net approach has many advantages in front of a marching-cubes reconstruction.

The method has an outer loop that traverses the volume following one of the axis aligned directions (usually +Z). Then a double inner loop iterates over the voxels of two planes perpendicular to the sweep direction, giving an overall complexity of $O(n^3)$ for a cubic volume, being n the number of voxels per size (being 512 the largest upper bound used throughout this work). The access to the different individual voxels is done in a ordered manner from lower to higher coordinate values.

The algorithm performs the construction of a mesh in two passes over the sweeping plane. The first step is to determine the set of *cubes* that contain surface voxels. As shown in Figure 4.3a, each cube is defined as a set of 8 neighboring voxels, 4 on each of the two consecutive adjacent planes on the sweep direction. A mesh-cube belongs to the surface when at least one of the eight voxels it contains is different from the others.

Simultaneously, a mesh can be initialized by placing a vertex at the center of each surface cube. The vertices are progressively linked with their neighbors using a 6-neighbor connectivity scheme (Figure 4.3b). However if each vertex is tested for the presence of the six neighbors, several links would be repeated forcing to perform a

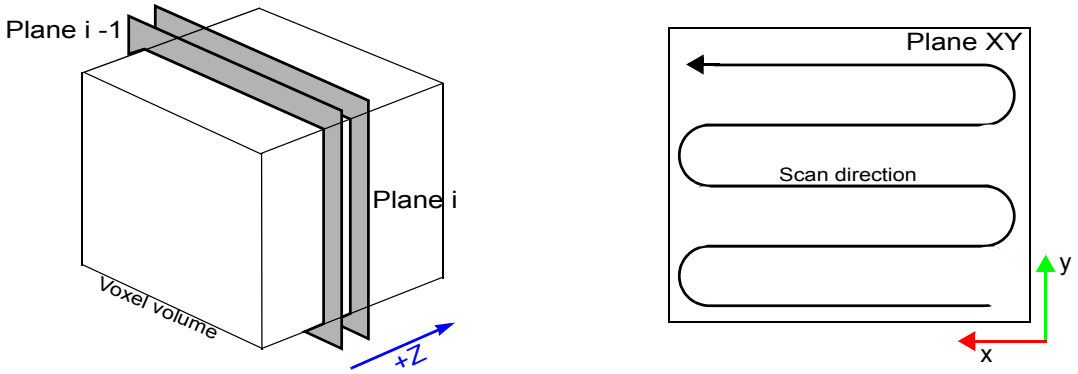


Figure 4.2: Ordered iterations over the voxel data structure.

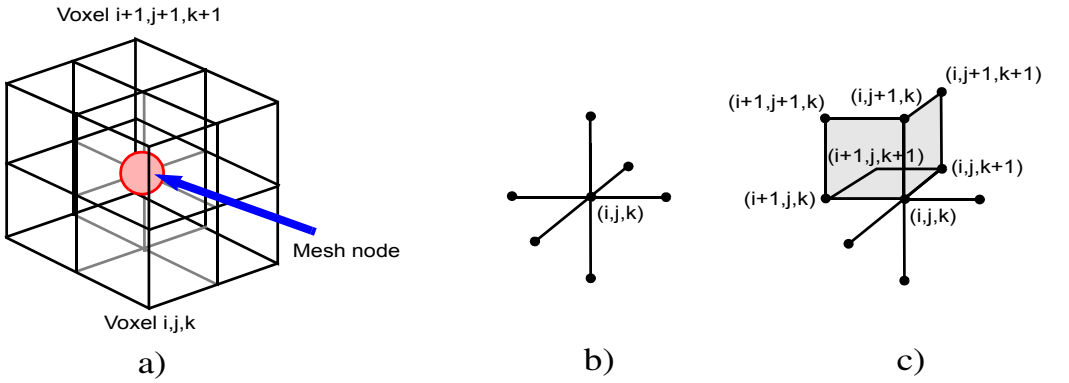


Figure 4.3: Mesh cube and surface mesh generation. a) illustrates the concept of a mesh cube. b) shows the 6 neighbor connectivity. c) shows the mesh face creation lookup.

bookkeeping to avoid repetition. Since the plane sweep and the voxel scan is performed in an ordered manner, it is only necessary to test for neighbor presence in three of the six possible directions, the same ones as the ones used for iteration purposes (+x, +y and +z).

Once the links have been established it is very simple to obtain a quadrilateral mesh by searching if for any given vertex $v = (i,j,k)$, there exists link connecting the following vertices that form the quadrilaterals Q1,Q2 and Q3 (shown in Figure 4.3c)

$$
\begin{aligned}
Q1 &= \{(i,j,k),(i,j+1,k),(i+1,j+1,k),(i+1,j,k)\} \\
Q2 &= \{(i,j,k),(i,j,k+1),(i,j+1,k+1),(i,j+1,k)\} \\
Q3 &= \{(i,j,k),(i,j,k+1),(i+1,j,k+1),(i+1,j,k)\}
\end{aligned}
$$

The generated mesh of connected quadrilaterals follows the faces of the voxels of the surface of the scene. To define the proper orientation of the quadrilaterals it is necessary to determine which of the two voxels that share the quadrilateral as a common face belongs to the surface voxel set. If both voxels do, then the quadrilateral is not added, since it corresponds to a pinch on the object surface.

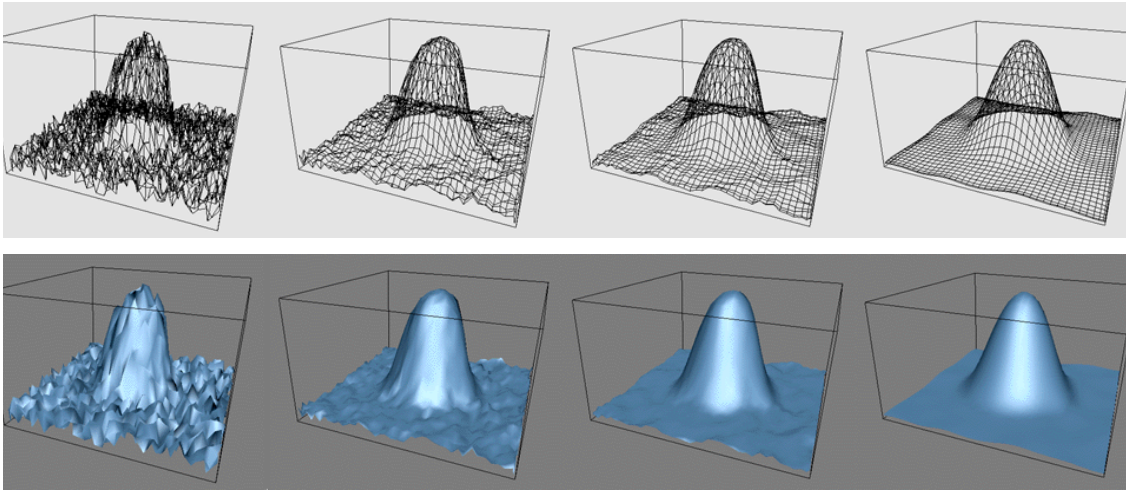


Figure 4.4: Laplacian mesh relaxation. The left image is a 2D Gaussian with noise added to each x,y,z component of each vertex. The subsequent smoothed versions are the result of applying the smoothing technique 2, 3 and 6 times respectively.

Once the outer main loop ends, a mesh smoothing process based on surface relaxation is applied in order to remove the sharp corners and edges of the obtained mesh. The relaxed surface has the same number of quadrilateral faces but the vertex positions are moved to generate a smoother representation retaining the features of the surface geometry. The technique used in this implementation considers each vertex v_i and all the neighboring vertices v_j that share a face with it and updates the position of the vertex based on the following equation, which provides a gaussian relaxation

$$v_i^* = v_i + \frac{1}{n} \cdot \sum_{j=0}^{n-1} (v_j - v_i) \qquad \text{(eq 4.1)}$$

This method tends to preserve gross features, generates equal size facets and can be applied iteratively as is shown in Figure 4.4.

However one of the main problems of the gaussian relaxation is that every time it is applied, the volume of the object is reduced, and eventually the shape converges to a

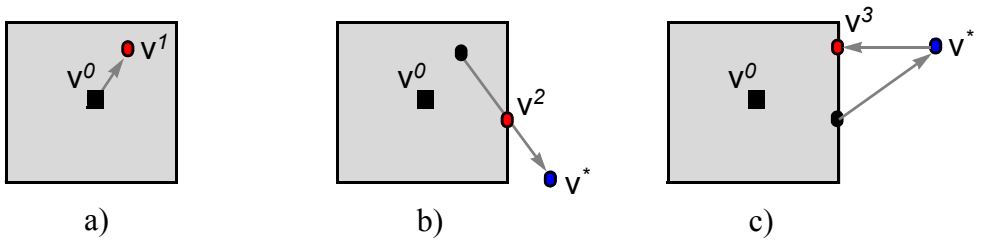


Figure 4.5: Displacement calculation during relaxation. In case a) the new point stays within the neighborhood of the voxel. In case b) the displacement is scaled so the final point stays at the boundary. In case c) since the point is in the boundary, it moves to a new location minimizing the distance to the calculated position.

single point in space. In order to avoid this side effect, we have added a constraint to the vertices that limits the displacement from its original position before the relaxation. The displacement vector is calculated at each iteration *k* as

$$d_i = v_i^* - v_i$$

where v_i is the processed vertex and v_i^* is the approximation calculated by Equation 4.1. Then, if the displacement moves the vertex outside the voxel neighborhood, the displacement is corrected by scaling it with a factor *c* so the new position of the vertex is place at the boundary of the allowed neighborhood.

$$v_i^{(0)} - (v_i^{(k)} + c \cdot d_i^{(k)}) \leq (r - 0.5)(1, 1, 1)^T$$

where *r - 0.5* is a constant that sets the appropriate neighborhood radius (e.g. if the neighborhood is the voxel itself, the constant *r = 1* and the factor is 0.5). In the case that the position *k* is already in the boundary of the neighborhood, the *x*,*y* or *z* component/s of the displacement vector that would pull the vertex outside are cancelled so the vertex can slide on the surface of the region boundary. Figure 4.5 illustrates the possible cases of displacement calculation.

The relaxation iterates until a maximum number of iterations have been performed or the energy value has reached a threshold. We calculate this energy as the sum of the square of the displacement applied (including the constant c) to each of the vertices

$$E(v) = \sum_i d_i'^2$$

Once a stable vertex configuration has been reached, the final mesh is created and the quadrilaterals are divided into triangles using a shortest distance criterion. The dividing edge will be placed between the pair of not connected vertices that have the shortest distance.

The presented meshing method produces smooth meshes avoiding the terracing effects of other algorithms in the literature such as Marching-Cubes. Such effects are difficult to avoid without out distorting the overall shape of the object due to an oversized smoothing post process. Moreover, the presented method generates a triangular mesh that smoothly wraps the object but is able to keep sharp corners as well as preserve thin structures.

4 PROPOSED MODEL REPRESENTATIONS

The main goal of the presented work is to propose a new paradigm for creating 3D models from images of real objects and be able to render those models in a computer virtual environment while trying to maintain as much as photorealism as possible. The way the presented pipeline has been design, is to enforce the use of the original imagery as part of the automatically reconstructed models. Furthermore, traditional computer

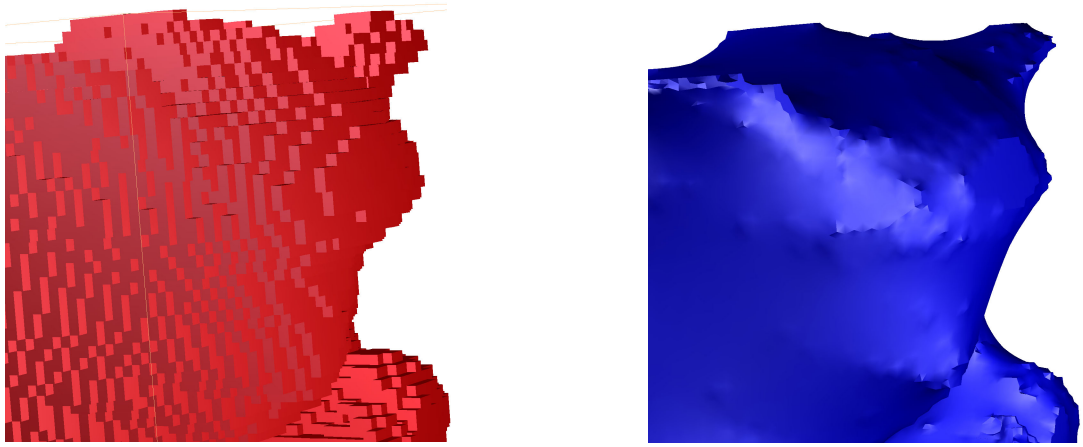


Figure 4.6: Reconstructed mesh example. The left side shows a close-up of a reconstructed mesh prior to the smoothing process. The right side is the same data after smoothing applying the spatial constraint to avoid excessive shrinking of the object.

graphics models, which consists of a set of polygonal meshes and textures, are not the unique representation and they are not necessarily the best one to achieve the desired quality on the renderings.

Several methods appear in the literature that follow the paradigm of *Image Based Rendering* (IBR), that is, they reuse the acquired image data rather than geometry to synthesize arbitrary views. The spectrum of methods goes from those that purely use images to generate novel views to those that rely in a more or less complex geometry proxy in combination with the acquired images.

In this work we want to push the envelope a little bit further by exploring and comparing different methods to combine the calibrated images and the scene reconstructions obtained in the previous stages of the proposed pipeline

In the following sections, four different approaches are presented, ranging from a point based approach without connectivity information, to a multiple overlapped meshes

combination, a new single mesh with overlapping textures and finally a more classical approach based on colored vertex meshes.

4.1 Object Splat

Point-based surface representations have recently been established as viable graphics rendering primitives [Gros01]. In particular they lead to compact multiresolution representations that can provide efficient level-of-detail (LOD) rendering of large point-sampled surfaces such as the digitally scanned statues of Michelangelo [LPC+00]. The advantage of point-based representations over triangle meshes is that explicit surface reconstruction and storage of mesh connectivity is avoided. Recent efforts in point-based rendering have focused on compact and very efficient multiresolution models, as well as on high-quality texture sampling and rendering [ZPBG01].

The major challenge for point based methods is to achieve smooth and continuous surface interpolation from point samples that are irregularly distributed over a surface. Furthermore, it must support correct visibility as well as provide an efficient rendering algorithm.

In [PaSG03] we propose a novel point blending and rendering technique that is based on the direct interpolation between point samples in 3D. In contrast to previous methods, the blending of surface points is defined as a weighted interpolation in object-space. We also provide an efficient technique to calculate splat sizes in a multiresolution point hierarchy that will be used by the rendering pipeline to optimally interpolate the color values between points to generate a final smooth surface.

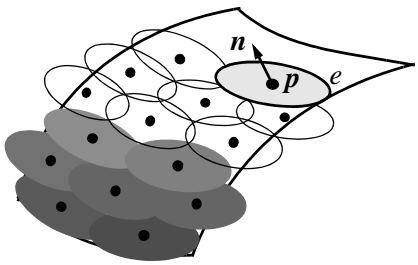


Figure 4.7: Elliptical surface elements covering a smooth curved 3D surface.

The input data set consists of point samples, surface elements (*surfels*) *s* with attributes for spatial coordinates $\boldsymbol{p}$, normal orientation $\boldsymbol{n}$ and surface color $\boldsymbol{c}$. This information is directly obtained from the scene reconstruction and the reference views using the procedure described in Section 3.1, Point data extraction (page 102).

Furthermore, the model representation assumes that each surfel also contains the information about its spatial extent in object-space. This *size* information specifies an elliptical disk *e* centered at $\boldsymbol{p}$ and perpendicular to $\boldsymbol{n}$. For correct visibility these elliptical surfel disks must cover the sampled object nicely without holes and thus overlap each other in object-space as shown in Figure 4.7. As noted also in [PZBG00], tangential planar disks may not completely cover a surface if it is strongly bent or under extreme perspective projections. However, this is not often noticeable in practical situations.

An elliptical surfel disk *e* consists of major and minor axis directions $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ and their lengths. Together with the surfel normal $\boldsymbol{n}$, the axis directions $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ define the local tangential coordinate system of that surfel.

4.1.1 Multiresolution hierarchy

The surfel representations discussed in [PaSG03] are hierarchical space-partitioning data structures based on *point-octrees* ([Same84], [Same89]) hierarchy which partitions

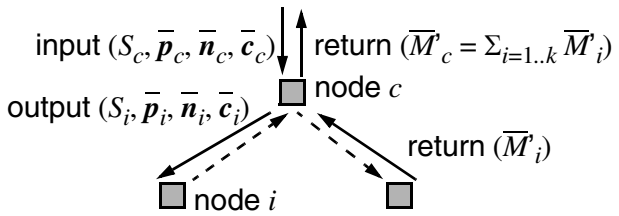


Figure 4.8: Point-octree generation data flow.

the space adaptively to the sample distribution (data-driven) rather than regularly in space (space-driven).

Each cell or node c of such a hierarchy H, containing a set of k surfels $S_c = \{s_1 \ldots s_k\}$ has a representative sample s_c with average coordinates $\bar{\boldsymbol{p}}_c = k^{-1} \cdot \sum_{i=1}^{\kappa} \boldsymbol{p}_i$, as well as average normal $\bar{\boldsymbol{n}}_c$ and color $\bar{\boldsymbol{c}}_c$ information.

Given n input surfels $s_1 \ldots s_n$, a point-octree data structure can efficiently be generated by a single depth-first traversal in O(n log n) time as illustrated in Figure 4.8. Given the k surfels $S_c = \{s_1 \ldots s_k\}$ of a node $c \in H$ and the average position $\bar{\boldsymbol{p}}_c$ of the surfels S_c, the set S_c is partitioned into sets $S_1,\ldots,S_8$ according to the eight octants with respect to the split coordinate $\bar{\boldsymbol{p}}_c$. While dividing S_c into the subsets S_i the averages $\bar{\boldsymbol{p}}_i$, $\bar{\boldsymbol{n}}_i$ and $\bar{\boldsymbol{c}}_i$ are computed, and the process is repeated recursively for the eight child nodes. To compute the elliptical disk e_c the child nodes of c return their *generic homogeneous* (see Section 4.1.2) covariance matrices $\overline{M}_i$ to calculate $\overline{M}_c$ and then derive the ellipse e_c and its size.

4.1.2 Splat size calculation

According to [PaSG03], a set of n points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n \in \mathbf{R}^3$ and the associated average point $\bar{\boldsymbol{p}}$ a covariance matrix can be defined as

$$M = n^{-1}\sum_{i=1}^{n}(\boldsymbol{p}_i - \bar{\boldsymbol{p}}) \cdot (\boldsymbol{p}_i - \bar{\boldsymbol{p}})^{\mathrm{T}} \qquad \text{(eq 4.2)}$$

In homogeneous space with $\boldsymbol{p}'^{\mathrm{T}}_i = (\boldsymbol{p}^{\mathrm{T}}_i, 1)$ we can rewrite the expression $(\boldsymbol{p}_i - \bar{\boldsymbol{p}}) \cdot (\boldsymbol{p}_i - \bar{\boldsymbol{p}})^{\mathrm{T}}$ to $(T \cdot \boldsymbol{p}'_i) \cdot (T \cdot \boldsymbol{p}'_i)^{\mathrm{T}}$ with the transformation matrix T denoting the translation by $-\bar{\boldsymbol{p}}$. Thus we can rewrite Equation 4.2 to

$$M' = n^{-1}T \cdot (\sum_{i=1}^{n}\boldsymbol{p}'_i \cdot \boldsymbol{p}'^{\mathrm{T}}_i) \cdot T^{\mathrm{T}} = n^{-1}T \cdot \overline{M}' \cdot T^{\mathrm{T}}$$

with $\overline{M}'$ denoting the new *generic homogeneous* covariance matrix of points $\boldsymbol{p}_1 \ldots \boldsymbol{p}_n$. In fact, the homogeneous covariance matrix of a set of points can be expressed for any local coordinate system from the generic homogeneous covariance matrix as

$$M' = n^{-1}R \cdot T \cdot \overline{M}' \cdot T^{\mathrm{T}} \cdot R^{\mathrm{T}} \qquad \text{(eq 4.3)}$$

with T denoting the translation of the point set with respect to the new origin and R expressing the rotation into the new coordinate axis directions.

Moreover, given two different sets of points $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ and $Q = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m\}$ as well as their covariance matrices $\overline{M}'_P = \sum \boldsymbol{p}'_i \cdot \boldsymbol{p}'^{\mathrm{T}}_i$ and $\overline{M}'_Q = \sum \boldsymbol{q}'_i \cdot \boldsymbol{q}'^{\mathrm{T}}_i$ as sum of tensor products, the combined generic covariance matrix $\overline{M}'$ of the union $P \cup Q$ is simply given by

$$\overline{M}' = \overline{M}'_P + \overline{M}'_Q \qquad \text{(eq 4.4)}$$

4.1.3 Splat-size determination

The elliptical disks e_c of nodes $c \in H$ must cover the surface at all levels in the multiresolution representation. It can be proved that a 2D covariance matrix of a set of points represents an ellipse with semi-axis length proportional to the eigenvalues of the matrix.

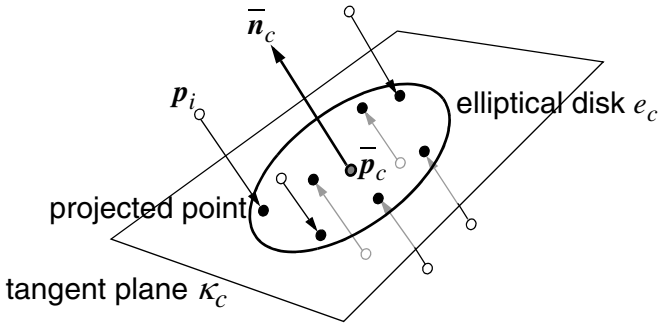


Figure 4.9: Projection of points onto tangent plane. The plane k_c is located at position p_c and with normal n_c.

Given a set of points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n \in \mathbf{R}^3$ and their center $\bar{\boldsymbol{p}}_c$ and its normal $\bar{\boldsymbol{n}}_c$, the set of points $\boldsymbol{p}_1 \ldots \boldsymbol{p}_n$ can be projected onto the tangent plane κ_c: $\bar{n}_c \bullet (\boldsymbol{x} - \bar{p}_c) = 0$ defined by $\bar{\boldsymbol{p}}_c$ and normal orientation $\bar{\boldsymbol{n}}_c$, as illustrated in Figure 4.9.

Then, we calculate the generic homogeneous covariance matrix $\overline{M}'_c$ expressed in the local tangent plane κ_c by first calculating it in the object reference system and second transforming it to the tangent plane using Equation 4.3. It turns out [PaSG03] that the 2 *x* 2 submatrix corresponding to the axis in the plane describes an ellipse with semi-axis length proportional to the eigenvalues of the submatrix, and principal planar orientation defined by the corresponding eigenvectors. By calculating these, we obtain the 2 axis on the tangent plane defining, together with the normal, a true 3D local coordinate system. By calculating the scale factors that applied to the eigenvalues make the ellipse contain all the points, an ellipse size can be determined.

Moreover, when calculating the hierarchy we can use the addition property of the covariance matrix to accumulate the matrices and then calculate the axis by a simple

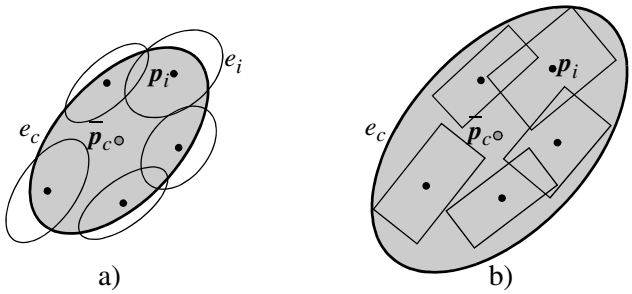


Figure 4.10: Ellipse coverage determination. a) Ellipse e_c bounding only the points $\boldsymbol{p}_i$ and b) conservatively bounding the disks e_i.

realignment and a 2D eigenvector problem. Therefore it is not necessary to accumulate the points to calculate an aggregated ellipse.

In order to calculate the appropriate ratio for the ellipse semi-axis of level *k+1* without considering all the aggregated points of level *k*, we test the corners of the projected bounding boxes of all the ellipses e_i of level *k* which conservatively determined which is the size of the ellipse at level *k+1* (see Figure 4.10). Without this conservative measure a coarse LOD would not cover the surface without holes. For non-leaf nodes the bounding ellipse covers its child node ellipses.

The initial elliptical disk of the input point set is computed using the *k*-nearest neighbors of each point, calculate the average normal if necessary, compute the covariance matrix of this neighborhood and get a bounding ellipse of the *k*-nearest neighbors as outlined above.

The complete model representation for the Object Splat method presented in [PaSG03] contains a hierarchical point-octree data structure which nodes store information about their optimal local coordinate system, normal information and the splat size and color for that node. The leafs of the octree are the initial points with

Figure 4.11: Splat based rendering example. The head of Michelangelo's David statue rendered using the Object Splat method by Pajarola *et al*.

normal and color orientation. Figure 4.11 shows an example of an extremely large model (over 2 million points) rendered using the object splat system.

4.2 DMesh Objects

DMesh, by Pajarola *et al.* [PaSaY03], is a model representation which consists of a piece-wise linear approximation of depth images as a textured and simplified triangle mesh. The method obtains a hierarchical multiresolution triangulation from each of the depth buffers of the reference views and performs a segmentation to avoid occlusion artifacts. The final model representation consists of a set of overlapping multiresolution triangular meshes, the *depth-meshes*, that use the original respective color images as a texture maps in order to render the original object as a set of textured and warped depth-meshes.

The DMesh representation offers several improvements and alternatives compared to previous depth-image warping techniques. The main contributions in respect to the modeling process include:

- A fast depth-buffer triangulation and simplification technique based on a hierarchical quadtree triangulation algorithm.
- Adaptive and view-dependent depth-meshes that can be efficiently stored and rendered at interactive frame-rates for high-resolution depth-images.
- An efficient and extremely simple multiresolution mesh segmentation method to remove non-significant triangles.

The process of creation of a DMesh set starts by obtaining a smooth mesh of the scene to be modeled, using the method proposed in Section 3.2. Then the algorithm renders the scene from each of the reference view points using the information of the camera calibration pipeline step. For each render, a depthmap from the camera's point of view is obtained and triangulated using the depth-mesh method based on a image space quadtree triangulation.

4.2.1 Quadtree Triangulation

DMesh uses the *restricted quadtree triangulation* (RQT) method ([SiSa92],[Paja98]), commonly used in terrain visualization applications, to generate a triangulated surface using an incoming depth-buffer. Figure 4.12 shows the basic recursive quadtree subdivision and triangulation that introduces vertices from the grid in two steps.

To avoid cracks in the triangulated surface from unrestricted adaptive subdivision and triangulation of the quadtree as shown in Figure 4.13, RQT subdivision is constraint such that the levels of adjacent quadtree nodes differ by at most one.

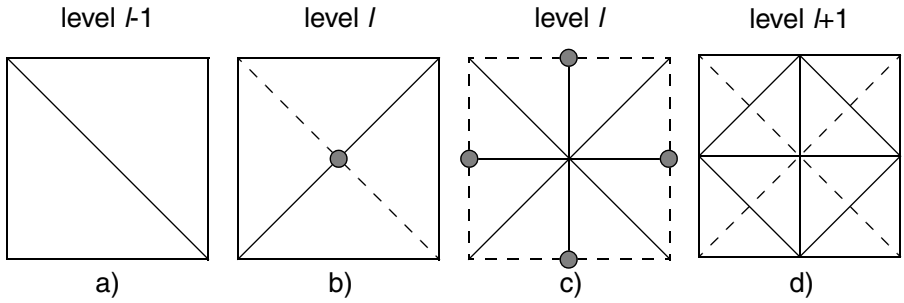


Figure 4.12: Recursive quadtree subdivision and triangulation. Refinement points are shown as grey circles in b) for a diagonal edge bisection and c) for vertical and horizontal edge bisections. The bisected edges are denoted by dashed lines

An efficient variation of this constraint to avoid cracks is the dependency relation shown in Figure 4.13. This relation specifies for each vertex v on level l two other vertices v_a, v_b on level l as in Figure 4.13 b) and d), or on level l-1 as in Figure 4.13 a) and c), that must be included in the triangulation such that v itself can be selected without introducing a crack.

Note that this hierarchical triangulation allows the entire triangle mesh to be represented by one single generalized triangle strip [Paja98], representation that will be exploited for rendering efficiency.

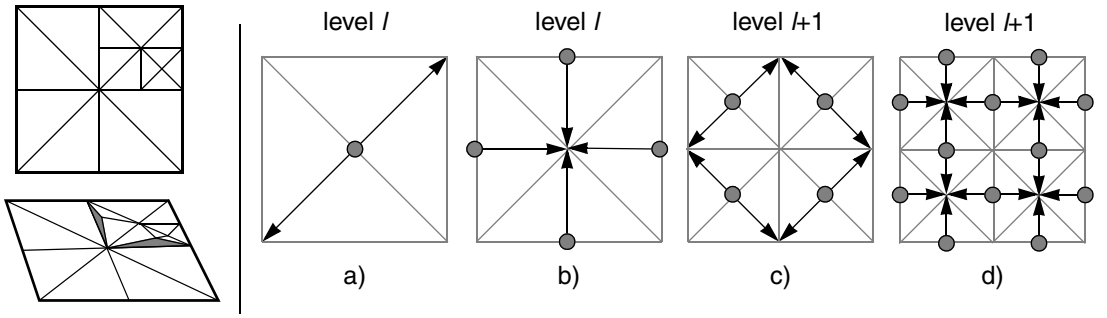


Figure 4.13: Cracks in unrestricted quadtree subdivision. Shaded in grey appear the resulting from an unrestricted quadtree subdivision. The dependency relations used to avoid these cracks are shown on the right. The center vertex a) depends on the inclusion of two corners of its quad region. The boundary edge midpoints b) depend on the center vertex. Dependencies within and between the next higher resolution levels are shown in c) and d).

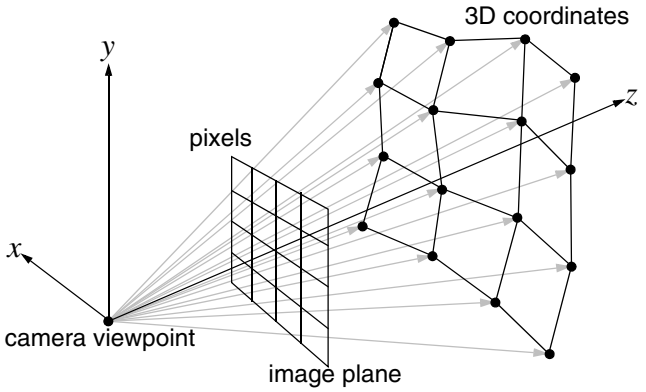


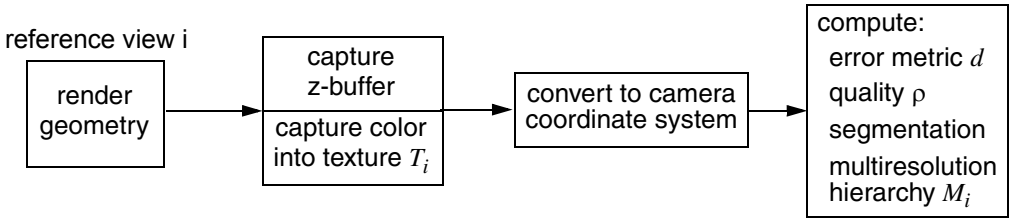
Figure 4.14: Grid of depth-image pixels. They define a projective height-field grid of 3D coordinates in the camera coordinate system.

4.2.2 Construction of a depthmesh

We can consider the depth map associated to a reference view to be a 2.5-dimensional (projective) height-field with a regular grid structure, the image pixel grid, similar to terrain elevation models.

Given the depth values in the z-buffer for a particular reference image, we can calculate for each pixel i, j its corresponding 3D coordinate $p_{i,j} \in R^3$ in the camera coordinate system by projecting a ray through the center of the pixel and calculating the intersection with the depth map (see Figure 4.14). Using the coordinates of points $p_{i,j}$, a quadtree based multiresolution triangulation hierarchy can be constructed on the grid of pixels of the reference depth-image. This triangulation of a reference image with its associated depth map is called a *depth-mesh*, and the representation of an object by multiple depth-image triangulations a *depth-mesh object*.

The main steps to generate a multiresolution depth-mesh data structure from a given mesh model of an object can be summarized in the following block diagram



After rendering the scene from the reference view, the acquired depth-buffer has to be converted into 3D points expressed in the camera coordinate system. On the resulting grid of 3D points the quadtree hierarchy is initialized by computing an error-metric and quality measure for each point. Furthermore, in a post process a per-triangle segmentation is performed to remove the potential triangulation errors the triangulation might introduce.

During initialization of a depth-mesh an error-metric is required to determine the approximation error of a simplified triangle mesh. Point-to-surface distance is used as object-space geometric error metric. The error of a refinement point is its distance along the elevation axis, the *z*-axis, to the refined edge, thus a point-to-line distance function. In general, for refinement points p bisecting an edge between points a and b the approximation error is calculated as the 3D point-to-line distance:

$$d = \frac{|(b-a)\times(b-p)|}{|b-a|} \qquad \text{(eq 4.5)}$$

To achieve a conservative monotonic error metric in the quadtree hierarchy, this distance error metric is maximized in such a way that center vertices (Figure 4.12 b)) store the maximal error of all points within that sub-tree of the quadtree hierarchy.

Additionally for adaptive meshing and rendering purposes we compute and store the following information in the hierarchy. For each depth-mesh vertex p we store the surface normal n_p, and we calculate a per-vertex quality measure $\rho_p = \left| n_p \bullet (0, 0, 1)^T \right|$ with the vector (0,0,1) being the view direction in the local camera coordinate system. A per-vertex quality measure allows smooth interpolation and blending over depth-mesh triangles. Moreover, for each center vertex of a quadtree block (Figure 4.12 b)) we store a bounding sphere radius r_p that includes all depth-mesh vertices within that block. The bounding sphere information is used for the view-dependent depth-mesh simplification

The multiresolution quadtree triangulation hierarchy imposed on the depth-buffer as explained above can be used in different ways to generate adaptively triangulated depth-meshes. The basic algorithm is to perform a recursive top-down traversal of the restricted quadtree triangulation hierarchy [SiSa92],[Paja98] and select points adaptively according to some error tolerance criterion. To guarantee a crack-free triangulation, for each selected point the dependent points are selected as well (see [Paja98]). The selected set of points define a conforming restricted quadtree triangulation which can be represented by a single triangle strip. This triangle strip is also generated in a single top-down traversal as shown in [Paja98].

A view-independent depth-mesh simplification does not take the novel viewpoint into account for which depth-image warping is performed. In this case a reference depth-mesh is simplified according to an object-space geometric approximation error threshold ε. A scale-independent geometric error can be achieved by dividing the error-metric distance d_p of a depth-mesh point p by its distance z_p from the reference

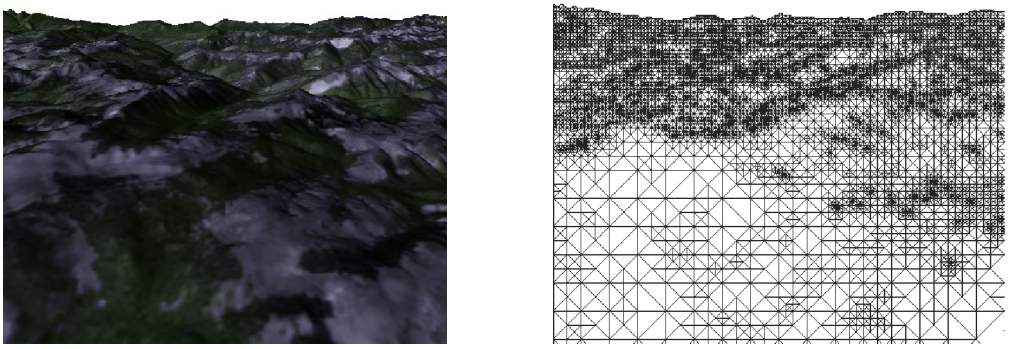


Figure 4.15: Adaptively triangulated terrain using the DMEsh approach. Instead of warping all pixels in the image, DMesh renders a triangulated height map with texture mapping.

viewpoint. Hence given an error tolerance ε and since we initialized the error-metric as squared distances, a static simplified depth-mesh is extracted by selecting all points with

$$d_P^2 / z_P^2 > \varepsilon^2 \qquad \text{(eq 4.6)}$$

Figure 4.15 shows an example of a triangulated reference depth-image of size 513 × 513 pixels viewed from the actual reference viewpoint. A traditional depth-image warping has to warp 263,169 individual pixels, while DMesh, using a small error tolerance of ε = 0.00001, warps this reference depth-image to any new viewpoint by rendering 32,961 textured triangles

In addition to view independent depth-mesh simplification, the quadtree triangulation hierarchy also allows to simplify a reference depth-mesh specifically for any given novel viewpoint which the reference view has to be warped to. This can be achieved by adding the view point location in Equation 4.6. Thus using an image-space approximation error threshold τ and given the viewpoint *e*, a depth-mesh point is selected if $f(d_p, p, e) > \tau$. We choose *f* to be the geometric error d_p projected to the new

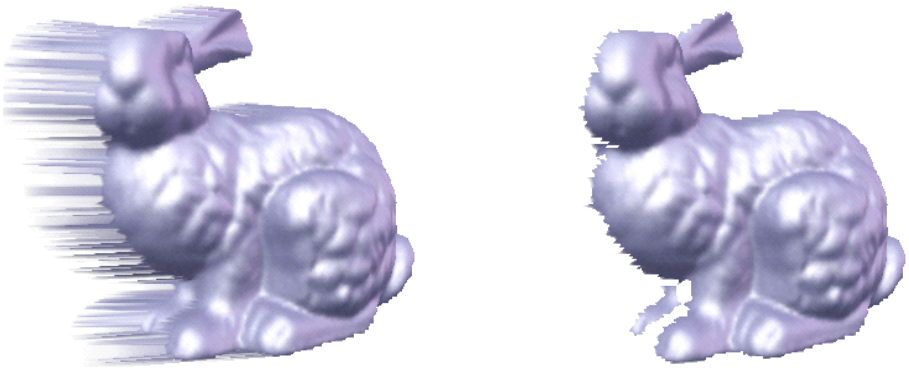


Figure 4.16: Rubber sheet artifacts.In a) the artifacts are clearly visible, and are removed in b) by appropriate segmentation

viewpoint. This criterion can be applied during rendering time, obtaining an adaptively triangulated depth-mesh by selecting all points with

$$(d_p / |p - e|)^2 > \tau^2 \qquad \text{(eq 4.7)}$$

As outlined before, both vertex selections are performed recursively top-down in the quadtree hierarchy. In addition, for view-dependent simplification we can also take view-frustum culling into account by view-frustum testing the bounding sphere information mentioned before. The vertex dependency rules are also adhered to and a triangle strip is generated using the selected vertices as presented in [Paja98].

4.2.3 Depth-mesh segmentation

The triangulation of the depth buffer may introduce surface interpolations between different object surfaces and the background as shown in Figure 4.16 that causes artificial occlusion or *rubber sheet* [MaMB97] artifacts when warped to new viewpoints.

Therefore, it is important to determine whether triangles represent rubber sheets or not, and furthermore, if performed at run-time, this segmentation of the triangulated

depth-mesh has to be done very quickly. In DMesh [PaSaY03] an efficient per-triangle segmentation on the full resolution depth-mesh is proposed, that is performed once during the depth-mesh initialization phase and store the results in the multiresolution hierarchy.

We can observe that the rubber sheet triangles introduced during the triangulation of the depth-image have the following property: the triangle normal is almost perpendicular to the vector from the viewpoint to the center of the triangle as shown in Figure 4.17. Let v be the vector from viewpoint to the center of triangle t and n_t be the normal of t. The following inequality using an angular threshold ω can be used to determine if a triangle is a rubber sheet triangle:

$$\left| \frac{v}{|v|} \bullet n_t \right| < cos(90° - \omega)$$

In all segmentation methods, additional care should be taken not to remove very small triangles which represent rough surface features but do not constitute a discontinuity. Therefore, in addition we also consider the depth-range Δz of triangles at distance z in the camera coordinate system and we only remove triangles which span a depth-range larger than some threshold λ:

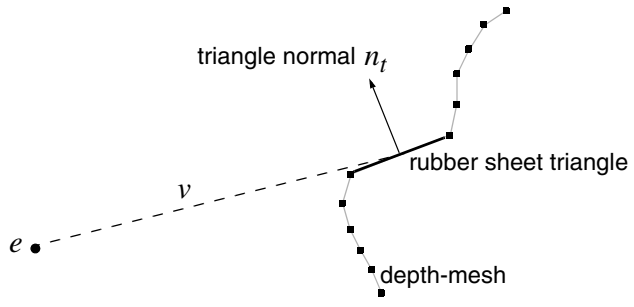


Figure 4.17: Rubber sheet triangles in the depth-mesh.

$$\frac{\Delta z}{z} > \lambda$$

The rubber-sheet segmentation is computed at initialization-time of a reference depth-mesh at full resolution. The segmentation result are stored as an 8-bit boolean flag S for the triangles a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2 of each quadtree node as shown in Figure 4.18 a). Then the segmentation results at the leaf level of the quadtree triangulation hierarchy are propagated upwards to the parent nodes. The same quadtree node, however, can also be triangulated with fewer triangles as in Figure 4.18 b). For these aggregate triangles a, b, c or d the segmentation is derived from the eight sub-triangles. For an aggregate triangle $t = a$ (as well as b, c or d) the segmentation is given by $S(t) = S(t_1) \vee S(t_2)$.

Each parent node on level l-1 determines the segmentation flags from the aggregate triangles of its child node on level l. For example, the segmentation flag of triangle a_1 on level l-1 in Figure 4.18 c) is set to $S(a_1^{l-1}) = S(a^l) \vee S(b^l)$ according to the aggregate triangles a and b in the child node on level l. Thus the segmentation of one triangle causes all parent triangles in the quadtree hierarchy to be segmented as well. Figure 4.19

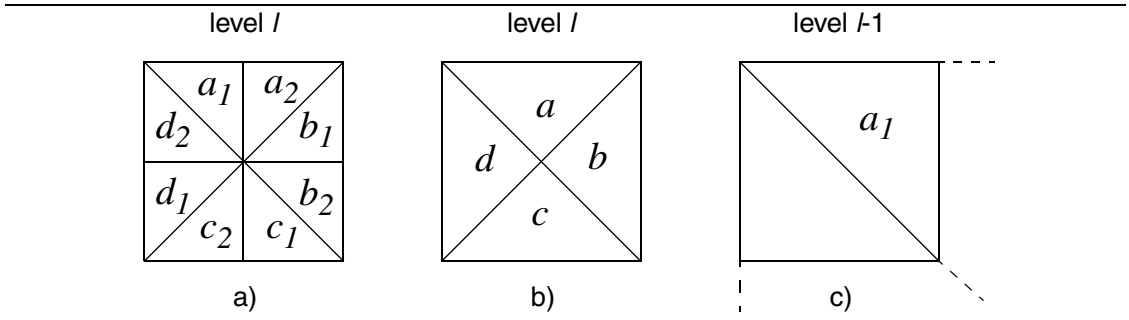


Figure 4.18: Segmentation flags. a) Segmentation flags for the full-resolution triangles of a quadtree node are stored as an 8-bit boolean field, and b) for the simplified triangles are expressed as boolean 'or' combinations. c) Flags on level l-1 are recursively calculated from level l.

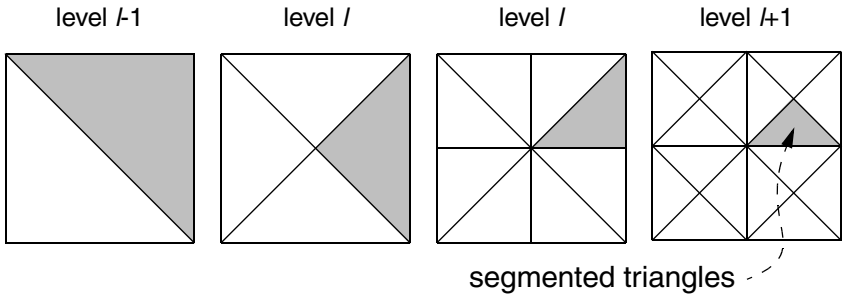


Figure 4.19: Propagation of segmentation. Segmented triangle (on level *l*+1) causes all parent triangles (on levels *l* and *l*+1) to be segmented too

illustrates this behavior of a segmented triangle that causes all parent triangles to be segmented as well.

After the initialization of the segmentation at full resolution and the propagation to coarser levels in the multiresolution hierarchy, the segmentation of an adaptively simplified depth-mesh can then be determined extremely efficiently for each quadtree node by a simple boolean expression. This works effectively for both static as well as dynamic view-dependent depth-mesh simplification.

In DMesh, the removal of segmented triangles is directly integrated with the triangle strip generation. While incrementally creating the triangle strip sequence the segmentation flags of each visited quadtree node are checked, and if necessary the strip is broken up into multiple smaller triangle strips.

4.2.4 Complete model description

The dataset corresponding to this model representation includes the reference view information as well as the images as texture maps. If a non-view dependant approach is used, a triangle strip of the selected vertices can be calculated during the creation of the model and stored along with the quality factor per vertex. In the proposed project we

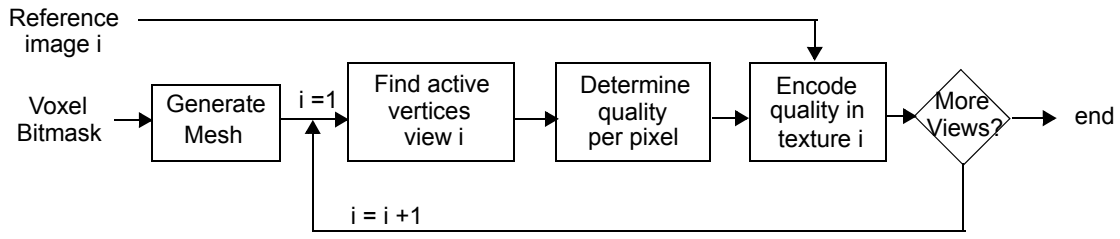
follow a view dependant approach, then the values of the error metric for each of the pixels is stored together with the segmentation flags and the bounding spheres for view-frustum culling.

4.3 MTMesh

The underlying basic structure of the reconstructed scene can be seen as a closed triangular mesh and a set of overlapping textures that can be projected onto the triangles to provide a realistic appearance under novel points of view. This basic structure provides a novel model paradigm, called *multi-textured mesh* (*MTMesh*), suitable to be rendered as we present later in Chapter 5

The MTMesh generation process of a given reconstructed scene starts by determining a smooth mesh that reproduces the scene using the procedure presented in Section 2.3.2. Then, for each reference view *i*, we need to determine which are the visible vertices and faces. Because of the configuration of the reference views and the main requirements of an image based reconstruction approach, there is a always a common set of faces that is visible for any given pair of views, indicating that a high degree of overlap is present between the views. For a given face of the mesh, not all the reference views provide the same information due to the relative spatial orientation, so a quality factor can be determined to weight the contribution of each reference view for

any given vertex and mesh face. The overall modeling process for constructing a MTMesh can be summarized in the following block diagram



The following sections discuss the different details of the MTMesh modeling process

4.3.1 Active vertices

We proceed to determine which vertices are visible in a given reference view *i* in a two step filtering process. First, a filter based on vertex normal orientation relative to the viewing direction can be applied, rejecting as not visible those vertices which are not oriented towards the view *i*

$$n \cdot (cop - c) \geq t$$

where *cop* is the center of projection of the reference view, *c* is the center of the face being tested, *n* is the normal of the face and *t* is a threshold to control the test. Ideally, and depending on the number of reference views, a large threshold is desired in order to filter out those faces which projection footprint is relatively small to the real size, assuming that some other reference view will have a better perspective of that face.

The viewing direction is computed as the vector from the center of face *f* to the center of projection of reference *i*, cop_i. As shown in Figure 4.20a, this method is better than considering the direction from cop_i to the center of the image plane, since it

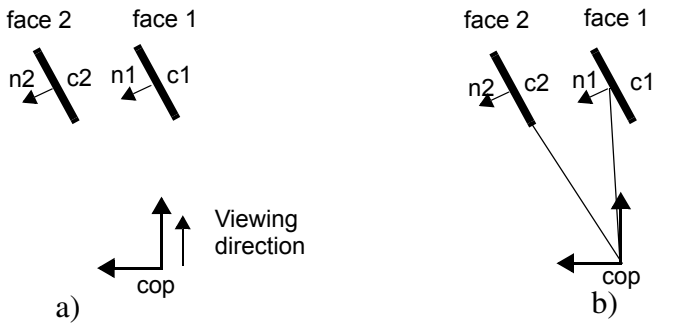


Figure 4.20: Normal orientation test. In case a) the central viewing direction is used to calculate the angle of the faces. In both cases will provide the same numerical result, while face 2 should be rejected. In case b) the compensated perspective viewing direction is used and in this case face 2 will be rejected.

compensates for the perspective camera model effects. Otherwise, faces that project close to the border of the image plane and which orientation is close to the one of the projecting ray will pass the test, generating a very small footprint, as shown in Figure 4.20b.

It is possible that some faces pass the orientation test but are occluded to the reference view because they belong to parts of the object that are not visible from that perspective. This requires a second filtering step based on a depth analysis. We use the same ε-zBuffer approach as the one presented in Section 4.2.1, Occlusion map generation (page 81), consisting of rendering the complete mesh from the reference view, using the calibrated settings of the camera and applying a perspective depth offset.

Any given face that passes the orientation test will be examined by testing the face center depth against the value stored in the depth map. If the face is visible, then both the vertices involved and their connectivity are stored as part of that reference view mesh, incrementally growing the geometric information.

4.3.2 Quality factors

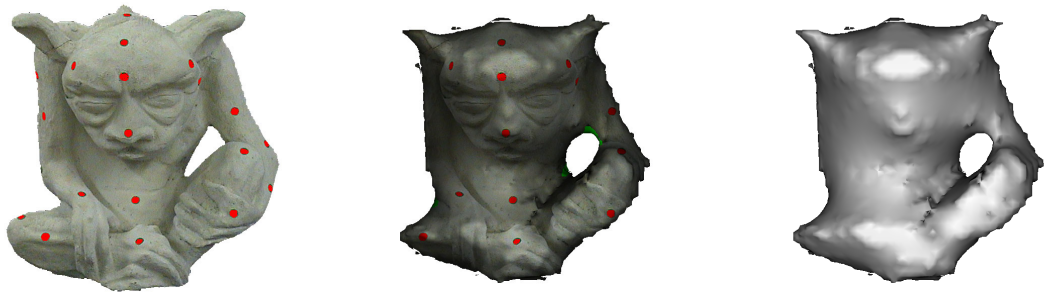
Given the set of faces visible in a reference view, it is easy to see that not all of them present the same footprint to the image captured from the view perspective. Faces with a more perpendicular orientation will have a lower pixel sampling than the ones that are parallel to the image plane. Moreover, exploiting the high degree of overlap, most likely there will be other reference views that will presented a better exposure of some of the visible faces from the actual view. Since during rendering all this information will be combined to generate the final color information, it is a very good measure to limit the contribution of a view to a face where the sampling is limited.

We propose to calculate a coefficient of projection quality based on the orientation of the faces that provides a weight per reference view that will be used to combine the multiple texture information. Moreover, the proposed coefficient will be calculated in a pixel basis for the reference view image.

Using a similar concept as the orientation filtering, it is very easy to calculate a perspective projection corrected relative angle between a given vertex and the reference view center of projection. The angle range in absolute value is between 0 and 90 degrees, and an angle factor between 0 and 1 can be calculated as

$$\alpha_{factor} = 1 - \frac{angle}{90°} \qquad \text{(eq 4.8)}$$

This factor can be calculated per each of the visible vertices, and then we use the OpenGL API to generate an interpolated pixel map of these quality factors. The faces are rendered using the different α_{factor} as color per vertex, and, at the end, the color buffer contains an interpolated colormap that can be captured and stored as a quality



a) b) c)

Figure 4.21: Quality map of for a given reference view. a) is the original image, b) is the image with a per-pixel quality factor measure multiplying the R, G and B channels, and c) is the quality factors encoded as a greyscale map.

map. In Figure 4.21 we can see an example of a reference view, the set of visible triangles and the quality map obtained using the proposed approach.

4.3.3 Data encoding

The MTMesh generation process finishes when all the reference views have been examined. The final model description is built and stored in a binary file containing the following data

- An array of vertices including the ones not marked as visible from any of the reference views. This is in order to obtain a water-tight model.
- An array of all the faces of the model, and similarly as the vertices, we include the non visible faces to keep the model as a closed mesh.
- A list of the information of the reference views, such as the transformation matrices and internal parameters. This will be used in the runtime rendering.
- For each of the reference views a list of indexed faces determined as visible from that perspective.

- For each reference view a texture that encodes the original image in the RGB channels, and the alpha channel is used to store the quality map determined during the processing of the reference view.

The presented MTMesh model representation contains all the information from the reconstructed scene and provides an efficient way of storing and rendering it to achieve very compelling visually models. However, when the number of reference images and/ or the number of faces in the mesh is large, the rendering step can be penalized. Moreover, this model, as it has been defined, does not include LOD capabilities by itself, and some mesh simplification preprocess should be included and performed in parallel to obtain more simplified versions of the geometry of the model.

4.4 Standard mesh model

Additionally we propose a more classical model representation that is more useful for standard rendering pipelines and commercially available tools such as 3D modelers. This representation consists of a triangular mesh with color information per vertex, integrating the information from the different reference views.

One could argue that since a closed mesh is available it could be possible to generate a single texture map to obtain a regular textured mesh, which is a very friendly representation commonly used. However, the problem of obtaining a 2D parametrization of a mesh, suitable to be filled with color information from the images, is not a simple one, and several approaches in the literature have been proposed. We limit our traditional representation to a set of colored vertices because is out of the scope of the presented work to present novel results in the surface parametrization problem.

The procedure to obtain a colored mesh can be seen as a simplification of the MTMesh process, since the only data that has to be retrieved are the color of the vertices. This is done by combining the color information of the projection of a vertex in all the reference views where it is visible.

Of course, as shown in the MTMesh generation description, not all the views do contribute equally, based on the orientation of a given vertex. We follow exactly the same procedure to calculate a vertex quality factor that is used as a weight in the color equation. The final color of a given vertex, visible in a subset of k images can be calculated as

$$col = \frac{\sum_{i=1}^{k} \alpha_{factor}^{i} \cdot col^{i}}{\sum_{i=1}^{k} \alpha_{factor}^{i}}$$

where α_{factor} is calculated the same way as shown in Equation 4.8.

The colored mesh representation is finally built as a list of vertices coordinates, together with the list of normals and colors per vertex. Then a set of indexed face completes the model description. This can be stored in any 3D model format that allows to specify color per vertex such as the PLY format [Ply].

This model representation does not account for level of detail but since it is a very standard type of model, commonly available solutions to obtain simplified meshes can be applied such as [Hopp96].

5 CONCLUSION

The modeling stage of the pipeline has been presented as the process to transform the information recovered from the earlier stages, acquisition, calibration and reconstruction, into a useful representation for rendering.

A brief survey of different approaches available in the literature has been presented and four different solutions and the different processes used to obtain them have been described in detail

- A point based representation, which provides very compact model representation by removing the connectivity information. Moreover, this representation can be easily extended to provide built-in level of detail support
- A depth mesh warping representation that can generate optimized textured meshes based on the depth buffers of a more complex mesh taken from the different reference views. We have also seen that these meshes can be generated on the fly at rendering using a view-dependant criterion, providing a level of detail solution.
- A single-mesh/multiple-textures solution has also been developed that splits the original mesh in set of overlapping ones that can be combine using a set of quality weights to obtain a rendered novel view.
- A more traditional model representation using colored vertices has also been presented. A simple scheme to integrate the color from multiple views is applied to combine all the available information into a single value per vertex.

CHAPTER 5 RENDERING

1 INTRODUCTION

The ultimate goal of the proposed model reconstruction pipeline is to extract models of physical objects that can be rendered with the best possible quality. Several approaches for rendering can be taken based on the available representation of the object, the interactivity degree of the renderer and the desired quality.

When render speed is a requirement, simpler models have to be used and more preprocessing and more clever algorithms are needed to obtain good final quality. When the model is used in an off-line rendering tool for computer animation, more detailed models can be used since speed is not a requirement. The desired algorithm must maximize the use of the information available (geometry, textures,...) while minimizing the cost of using it maintaining the best possible result when rendering.

In this section a revision of the main rendering algorithms for different types of representations are presented with some of the desirable main characteristics for the proposed rendering algorithms.

2 RENDERING APPROACHES

Different approaches to the modeling and rendering problem classified by the primitive representation.

2.1 Image based rendering

The most direct representation is to combine the reference views to generate novel renderings using the recovered camera calibration information and the position and settings of the new view.

In [SeDy96], Seitz and Dyer proposed an algorithm to perform view morphing between two views taken into account 3D projective cameras and transformations. The technique works by pre-warping the images to two parallel views, then a morphing is calculated by interpolating the position of each pixel from the corresponding pair of pixels of the two initial images. A final post-warping from the interpolated camera orientation to the desired view is calculated and the new image is generated. The most important factor to perform a correct 3D morphing is the set of correspondences between the two reference views. This rendering algorithm could be applied to the obtained datasets by calculating the correspondence maps relating where each voxel projects into which image. However a more complex interpolation function should be obtained to exploit the redundant information in a multiple image setup.

Another image based approach can be constructed by calculating a depth image for each reference view. Each depth image gives for each pixel of the reference view the distance between the surface of the scene and the viewpoint. Any arbitrary view can be calculated by performing a *depth-image warping* [ChWi93] and using a z-buffer ordering, overlapping artifacts can be avoided. In [Mill95] a unique evaluation order is presented that guarantees a back-to-front drawing order of the warped pixels. Depending on the relative position of the new view with respect to the original view. A common problem in these methods is that uncolored pixels appear in the new view when there is a lack of coverage in the reference views. The use of multiple sources reduces exposure artifacts. Another improvement presented in the literature is the use of multiple depths for each pixels, constructing the *layered depth images* (LDI) [SGHS98] which improves the image depth warping and reduces the exposure artifacts.

Another set of techniques that rely exclusively in the reference views to generate novel views, calculate an approximation of the plenoptic function, which parametrizes the light observed from all directions at any point in space using 6 parameters (7 if time is taken into account). The two most important contributions are [GGSC96] and [LeHa96] that parametrize the plenoptic function using four variables by establishing two parallel planes and assigning to each ray of light two coordinates on each plane. The main assumption is that the light on the space to be modeled is free of occluders and the radiance along a ray is constant. A sampled set of radiances are obtained from the reference images using a calibrated gantry and during rendering, for each pixel of the novel view, the four parameters of the ray are calculated and the radiance value is obtained from the images. The main limitations of these methods are the reduced

number of new viewpoints, the size of the modeled scene, the memory footprint of the data structures and the large number of images needed to obtain quality renderings.

2.2 Point based rendering

Several problems arise when using colored points to render a solid object, mainly because the lack of information anywhere between the points, which becomes very noticeable in a close view of the surface of the object. Other problems are how to perform efficient renders using occlusion culling and how to detect collisions. Several efforts have been presented that solve the problem by rendering each point as a 2D area, *splat*, on the screen, usually an ellipsoid, and by using a weighting function, overlapped pixels from different splats are combined to generate a smooth transition of colors.

The *Qsplat* algorithm was presented in [RuLe00] as a complete solution for displaying large meshes with levels of detail (LODs), visibility culling and data compression. Based on a hierarchical set of bounding spheres and a quantized set of points they achieve interactive display rates in a 366MHz PII laptop. They test the algorithm with data obtained from range scanners, which presents high spatial density, with no texture information. A solution to render splats in OpenGL using multipass is presented, and several kernel splats are used (square, circle, ellipsoid and gaussian). Overall is a very impressive solution that renders large, very detailed objects at interactive speeds with a progressive refinement of the object, allowing an easy integration with a 3D environment since collision detection is easily performed.

The surfels approach by Pfiester *et al.* [PZBG00], and their later refinements on point splatting [ZPBG01], is another excellent non-connected point based rendering

algorithm that incorporates texture and bump mapping during the preprocessing stages. Based on large complex synthetic models or datasets from 3D scanners together with images, they precalculate the splats with texture mapping and normal perturbations (if bump mapping is desired). Using a method called visibility splatting they determine the holes and active surfels on a z-buffer and by applying gaussian filters they are able to cover most of the holes. The geometric model is sampled at different resolutions, and stored in three orthogonal LDI's, forming a layered depth cube (LDC), constructing a final LDC tree that allows to visibility culling during rendering.

Point based rendering with splatting is a very good alternative to a voxel based representation. However there are a few considerations to make:

- Hole filling. Since the obtained sampling is not as dense as a 3D scanner, a hole filling algorithm is needed in order to reduce artifacts.
- Voxel noise. The space carving reconstruction generate voxels that are consistent with the reference views, but from novel viewpoints appear as isolated voxels. It is necessary to carefully evaluate whether ignore them (generating new holes) or not.
- Multiple Textures. Since several reference views sample a voxel (point) from different angles it is necessary to determine the final texture to be applied to the splat. A view dependant texture mapping [DeTM96] or a more sophisticated approach based on Lightfields could be used.

2.3 Geometry based rendering

When using a geometric model reconstructed from several views a problem arises when texture mapping has to be assigned, since several views can be available for a

specific triangle. Several artifacts can be generated like (1) a distortion of the texture at the boundaries of neighboring triangles textured from different images (mainly due to shape errors, calibration errors and/or illumination changes between images) and (2) hidden surface parts of the real object that have no texture information (i.e. parts which can not be observed by the camera in any of the images).

In [NiBr95], Niem and Broszio describe an algorithm for texture mapping a 3D model from multiple camera views aiming a reduction of texture distortion associated with the boundaries between triangles mapped from different camera views. The proposed algorithm assigns each triangle of the mesh to a particular view which maximizes the texture resolution (usually the smallest angle between the viewing direction and the normal). A region merging is applied so larger neighboring areas assigned to the same reference view are obtained. When two triangles sharing and edge belong to different reference views, a blending function is calculated to minimize the distortion. Finally for those triangles not visible in any reference view, a texture is synthesized by combining the neighboring textures.

Another approach to solve the multi-image texturing problem is to perform a view dependent texture mapping render. Based on the initial method in [DeTM96], Debevec *et al.* propose in [DeBY98] an algorithm based on projective texture mapping to render in real time textured geometry combining up to three reference images. A preprocess is performed to calculate for each polygon the set of reference views in which it is visible and create a *view map* that encodes the best texture map to use for a regularly sampled set of possible viewing directions. The algorithm is based in the OpenGL API to

calculate the visibility of the polygons from each reference view. Then a hole filling process is applied to propagate the color of neighboring areas to those areas not viewed from any reference view. During rendering, the view map is queried for each polygon to determine a set, of no more than three images, to blend together to render the polygon using a real-time optimized algorithm that performs three passes over the geometry to generate an excellent multi-view blended texture mapped object.

2.4 Hybrid approaches

Another set of rendering approaches combine the images with a geometric approximation of the object to render. The use of a shape proxy accelerates the rendering speed by relying in hardware acceleration, as well as helping in pixel evaluation, visibility tests, etc.

Wood *et al.* present in [WAA+00] the *surface light field for 3D Photography* which is an hybrid combination between the Lightfield principle with a geometric proxy. During the preprocessing of the model a large number (order of hundreds) of photographs of the object are taken as well as a set of range scans (over 30) of the object. A triangular mesh is reconstructed and manually registered to the images. A based simplified mesh and a parametrization that maps to the scanned geometry is calculated. Then for each vertex in the base mesh a *Lumisphere* is calculated from the photographs. A Lumisphere can be considered a sphere centered on a point that encodes the color of every possible direction. Since the set of images is limited, a sampled Lumisphere is constructed for each point in the base mesh. Three different methods are evaluated and two compression schemes are used to reduce the final size of the model.

A real-time rendering algorithm is proposed that in a first pass calculates for each pixel of an arbitrary view which point on the surface of the base mesh projects to. In a second pass, the direction of the ray is calculated and the corresponding Lumisphere is evaluated, returning the color value for the pixel. The algorithm performs extremely well producing realistic renderings and because of the use of a geometric proxy, the object can be edited and deformed without loosing the rendering quality.

Another approach based on the Lumigraph ideas is presented in [BuBM01] that generalizes many of the different approaches of image based rendering. Depending on the combination of the reference views and the availability of a geometric proxy, the algorithm behavior goes from a typical Lumigraph all the way to a view dependent texture mapping approach. The method uses a set of calibrated views of the scene to construct a *camera blending field* that determines which reference views contribute and with what weight for each of the pixels of a novel arbitrary viewpoint. The weights are calculated based on several criteria such as distance of reference view to the new view, closeness of viewing angle and image resolution. A very clever real-time rendering algorithm is presented that sets a grid of sampling points in the image plane, calculates the blending values only for those points for the *m* closest reference views. Then using a constrained Delaunay triangulation, a 2D mesh is generated and the hardware automatically performs an interpolation of the blending values for all the pixels of the rendered image. Finally each pixel is calculated weighting the corresponding colors of the reference images with the blending factors by performing *m* projective texture mapping passes.

3 RENDERING APPROACHES

In this section we present the rendering paradigms for the model representations proposed in Chapter 4. They have being carefully designed to obtain the maximum performance and visual quality, using well established rendering API's such as OpenGL.

The rendering algorithm for the model representation based on a colored mesh will not be discussed since is a very straight forward solution, and we concentrate on the other three more sophisticated models.

3.1 Point based rendering

The complete model representation for the Object Splat method presented in [PaSG03] and detailed in Chapter 4, Section 4.1, Object Splat (page 110), is composed of a hierarchical point-octree data structure whose nodes store information about their optimal local coordinate system, normal information and the splat size and color for that node. The leafs of the octree are the initial points with normal and color orientation.

We propose ([PaSG03]) a point blending and splatting algorithm that uses hardware acceleration to efficiently render surfels as α-textured polygons, exploiting vertex and pixel shaders programmability. The main steps required to render a single frame are the following:

- View-dependent LOD selection of surfels s_i with projected screen size less than a threshold of τ pixels.

- Selected surfels s_i are represented as triangles oriented according to the surfel normal n_i and the ellipse axis e_{i1} and e_{i2}. The blending function is specified by an α-texture on the triangle.
- The surfels are splatted using an ε-*z*-buffer concept such that surfels within ε distance of each other are blended together.
- The intermediate blending result is normalized in an imaging post-process by dividing each pixel color by its accumulated blending weight.

The rendering algorithm is implemented in OpenGL and takes advantage of vertex-program [OpGL02] and pixel-shader [OpGL02b] or texture-shader [DoSp01] extensions. Figure 5.1 illustrates the different stages of our point rendering process.

3.1.1 LOD selection

The first step is a view-dependent LOD selection of the set of surfels of the hierarchy that are going to be rendered. This is a standard depth-first traversal of a multiresolution hierarchy defined by the point-octree data structure. Given the hierarchy H a top-down traversal decides for each node $c \in H$ if the corresponding surfel is visible, within the view-frustum and front-facing. Furthermore, if its elliptical disk projection on screen is larger than a threshold of τ pixels the child nodes are recursively processed. At the end, as set of surfels $s_1,...,s_k$ is obtained.

3.1.2 Visibility splatting

The surfel (*surface element*) concept has been introduced in Chapter 4as a point on the surface of the object with a set of attributes such as the object reference spatial

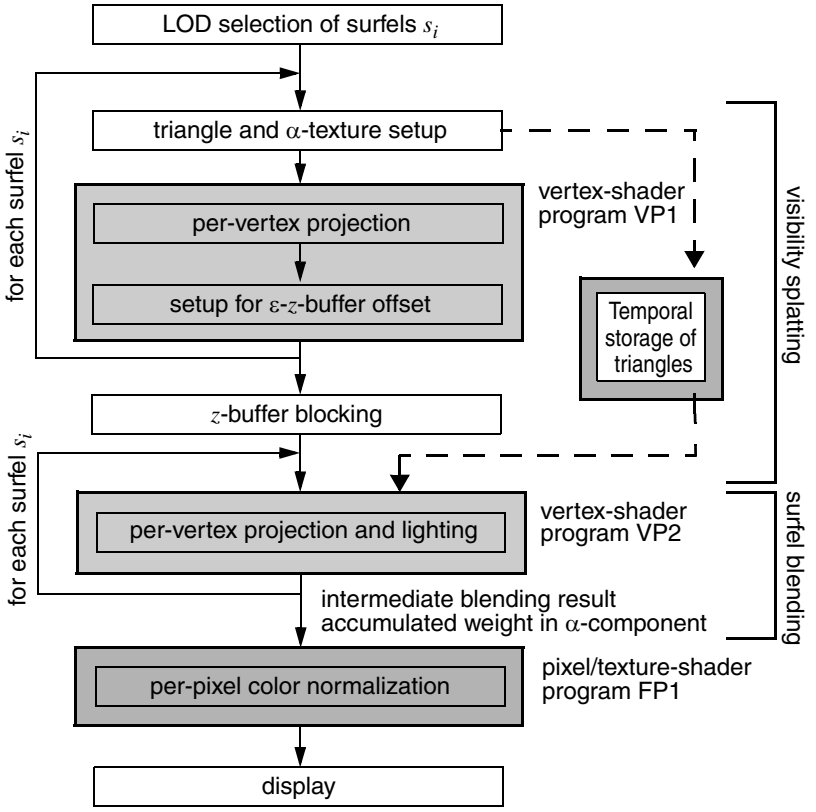


Figure 5.1: Point rendering algorithm block diagram.

coordinate p, a normal orientation n, a surface color c and a space extend size specified as an elliptical disk e centered at p and perpendicular to n, with a local orientation and the corresponding lengths e_1 and e_2 in the ellipse axis. Furthermore, the surfel elliptical disks have been calculated in such a way that there exists overlap between neighboring surfels, to guarantee that during rendering no holes in the surface are generated.

During rendering, a generic textured triangle t_{unit} is used as the primitive for a elliptical disk. This triangle has a α-texture map of a centered opaque circular disk as shown in Figure 5.2. Therefore a surfel is rendered by scaling the generic triangle t_{unit} in x- and in y-directions according to the major and minor axis lengths of ellipse e_i, rotating the triangle to align it with the surface normal n_i and the two ellipse axis, and then

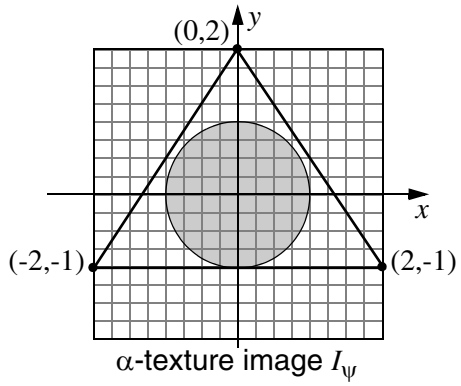


Figure 5.2: Generic triangle modeling a surfel.

translates it to the object-space coordinates p_i, resulting in triangle t_i. This can be performed very efficiently using a 3D API such as OpenGL.

In order to achieve smooth interpolation of the properties between partially overlapping surfels in object space, we define a set of functions, the *blending kernels*, which are analogous to the weight functions of the Bezier or B-Spline surfaces. Then the parameters of the surfels such as color are interpolated as a weighted sum of overlapping surfels.

Contrarily to the regular sampling of control points in a B-Spline patch, the surfels are not distributed in a regular grid (see Figure 5.3), Therefore, it is not possible to guarantee perfect overlapping and that the total sum of the weights at any given point is

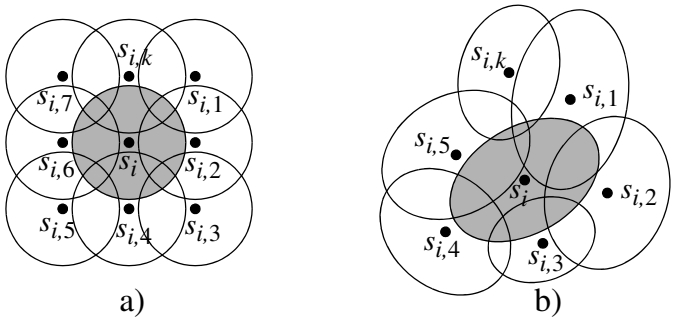


Figure 5.3: Distribution of surfels. a) shows a regular grid of control points and b) shows the actual irregular set of surface elements.

the unity. This constraints the surfel set and the rendering process requiring that (1) the overlap is not too dense so, in a worst case scenario, the sum of weights never exceeds one and (2) a final normalization will be needed to normalize the interpolated values.

The blending kernel is created as a two dimensional rotationally symmetric function $\psi: r \rightarrow [0, 1]$, that is applied to each surfel s_i by scaling and orienting it according to the ellipse e_i in object-space. There are many obvious choices for blending kernels $\psi(r)$ such as hat-functions or Gaussians

$$\psi_G(r) = e^{-r^2/2\sigma^2} \qquad \text{(eq 5.1)}$$

Pajarola *et al.* propose in [PaSG03] a new set of blending kernels $\psi_E(r)$ that support *plateau*-like blending kernels with variable sharp drop-off regions:

$$\psi_E(r) = e^{\left(-a \cdot \left(\frac{r}{b}\right)^n\right) / \left(1 - \left(\frac{r}{b}\right)^n\right)} \qquad for\ 0 \le r \le b \qquad \text{(eq 5.2)}$$

The novel blending kernel defined in Equation 5.2 is a positive smooth function with limited support ($\psi_E(r) = 0$ for $r > c_{max}$) and has control over the width and the slope of the function. In particular it is well defined over a limited support area specified by the parameter b with $\lim_{r \rightarrow b} \psi_E(r) = 0$. Furthermore one can adjust the width of the kernel with parameter a and separately control the slope of the drop-off region by the parameter n. Figure 5.4 illustrates the differences in control over the shape of the kernel between the novel functions and the Gaussian family. Obviously the Gaussian kernel can be made more or less localized with varying σ, however, this single parameter concurrently influences the slope of the interpolation around $\psi_G(c_{mid}) = 0.5$ and the support $\psi_G(c_{max}) \le \varepsilon$.

As can be seen from Figure 5.4b, when using the new proposed blending kernel, higher settings of a make more narrow blending kernels (for a fixed n), while high values n generate plateau like kernels. $\psi_E(r)$ can also simulate Gaussian shaped kernels if desired, for example $\psi_E(r)$ with $n = 2$ and $a = 10$ is very similar to $\psi_G(r)$ with $\sigma = 0.3$.

Blending between surfels is performed by mapping the blending kernel $\psi_E(r)$ onto the elliptical disk e_i of each visible surfel s_i. In fact, the blending kernel $\psi_E(r)$ is precomputed and discretized in a pre-process, and the result is stored in a high-resolution raster image I_{ψ}.

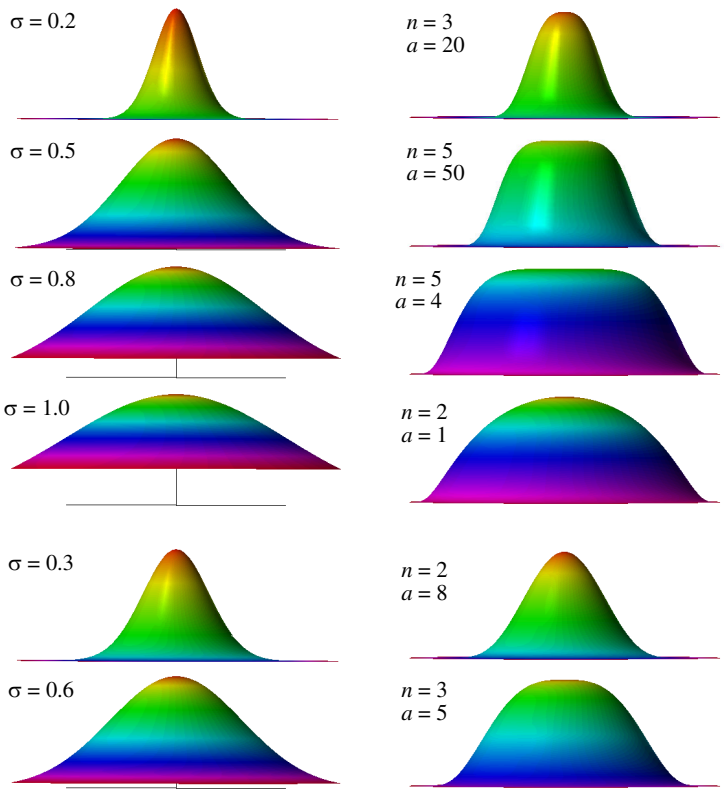


Figure 5.4: Blending kernels. Left: Gaussians $\psi_G(r)$ for r < 1.5. Right: Kernels $\psi_E(r)$ for b = 1.5 .

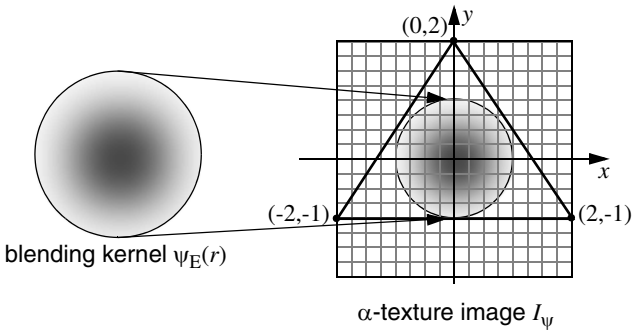


Figure 5.5: Mapping of blending kernel as α-texture onto a generic triangle.

As shown in Figure 5.5, this α-texture $I_{\psi.}$ is mapped onto a generic triangle t_{unit} in the *x,y*-plane such that the blending kernel covers the unit-size disk. As said before, for each surfel s_i, the triangle setup stage in Figure 5.1 performs all the necessary transformations resulting in triangle t_i. Thus each surfel is rendered as one triangle and blending is achieved by α-texturing.

Since the surfels are rendered using α-textures and blending functions, it is very important to guarantee non-neighboring surfels behind the actual rendered surface do not contribute to the final image. Moreover, the neighboring surfels are not necessarily coplanar to the one being rendered, but need to be taken into account for proper blending, invalidating a straight zBuffer test.

Instead we perform a two pass rendering, where in first pass all selected surfels s_i are rendered with lighting and α-blending disabled to create a ε-zBuffer (see Chapter 3, Section 4.2.3, Footprint calculation (page 85)). Note however, that the α-test function is enabled such that the α-texture map $I_{\psi.}$ indeed generates an elliptical splat during scan-conversion of the rendered triangle. After this first rendering pass the depth-buffer

represents the rendered surface perspectively translated by ε along the view-projection. This pass is optimally performed using a vertex program, VP1 in Figure 5.1.

Additionally, since the generic triangle transformation will be repeated for the second rendering pass and since this involves expensive CPU calculations, a temporal data structure is set to store the transformed values of the vertices of each surfel triangle t_i.

In the second rendering pass the depth-buffer is set to read-only and all visible surfels are rendered with lighting and α-blending enabled. Since the depth-buffer contains the visible surface offset by ε, the desired ε-*z*-buffering effect is achieved.

For the second rendering pass, the object-space coordinates of each surfel triangle t_i are read from the temporal data structure and sent to the standard OpenGL pipeline or a vertex program – VP2 in Figure 5.1 in our case – with per-vertex normal, α-texture and material properties in order to incorporate shading and lighting.

3.1.3 Normalization

The resulting image *I* contains pixel color values that correspond to an intermediate blending result $c' = (\alpha \cdot R, \alpha \cdot G, \alpha \cdot B, \alpha)$ where the α component contains the accumulated blending weight *w* for the different contributing kernels. In order to generate the correct colored image a normalization process needs to be performed. Since this is a common process of the next rendering paradigms, the details of the methodology are described in Section 3.4, Image Normalization (page 162).

Summarizing the Object Splat rendering method, Figure 5.6 shows an example of a reconstructed model rendered using our approach. The model is a cup reconstructed

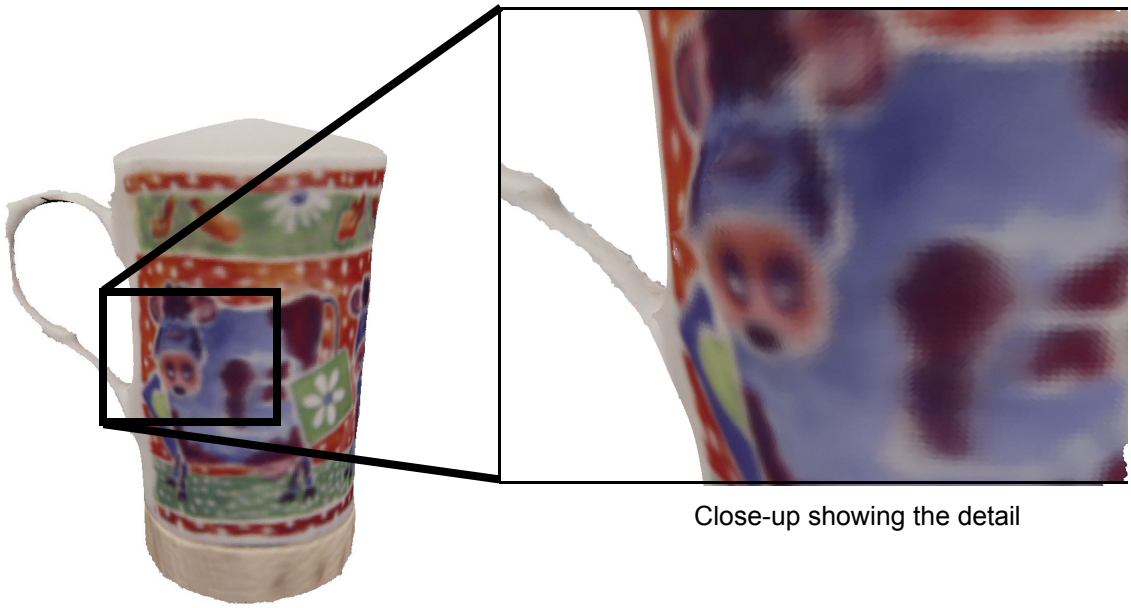
from 13 views. The geometry has 132,000 vertices and 265,000 faces and renders at about 4 frames per second in PC P4 2.0Ghz with and NVIDIA GeForce4 Ti 4600. The splatting algorithms needs 2 seconds to process the hierarchy and during rendering time it achieves 18 frames per second with lighting enabled, rendering up to 1,200,000 splats per second. The figure shows also a snapshot of the per-pixel splat contribution: in blue when only one splat contributes towards green when eight splats contribute.

3.2 DMesh rendering

The DMesh model representation is a hierarchical multiresolution triangulation that generates adaptively and view-dependently triangulated depth meshes from a set of reference views. The dataset corresponding to this model representation includes the reference view information as well as the images as texture maps. The values of the error metric for each pixel of the reference depth buffer, generated during the modeling stage, are stored together with the segmentation flags and the bounding spheres to support efficient view-frustum culling.

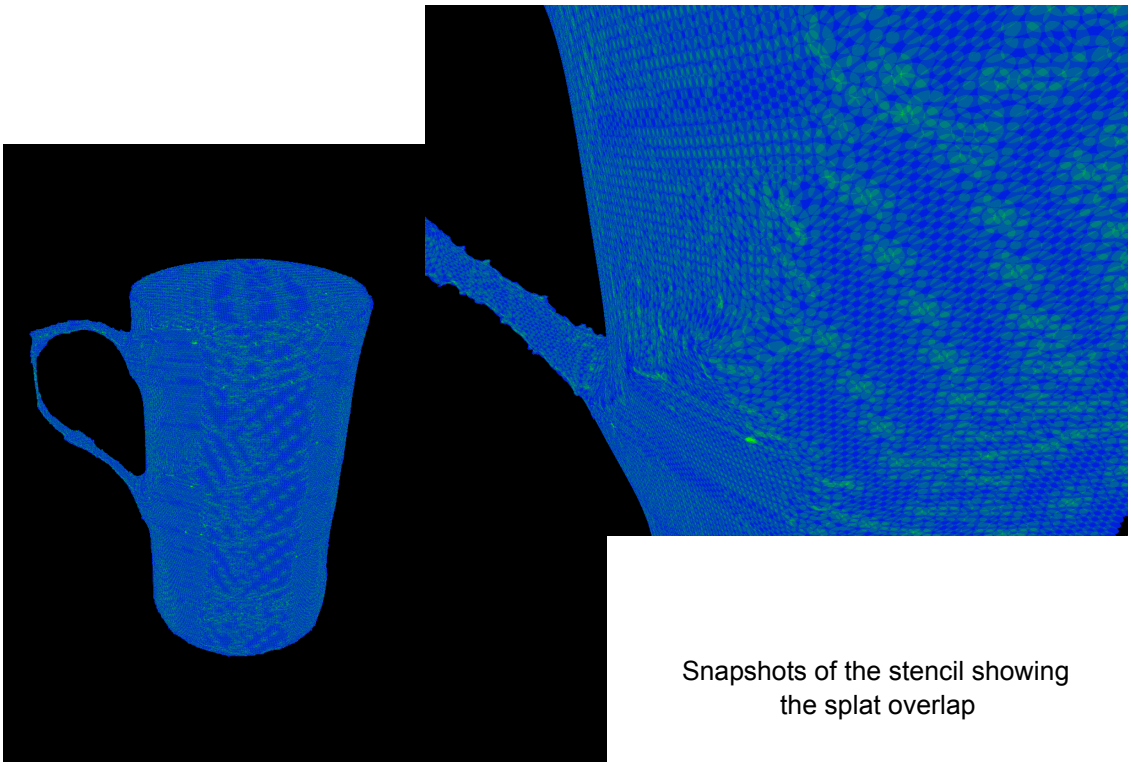
DMesh benefits from the hardware support to warp depth-images warping is by rendering textured polygons instead of projecting individual pixels from a reference depth-image to new views. The approximate depth-image consisting of a segmented triangulation of the depth-buffer, as presented in Chapter 4, Section 4.2, DMesh Objects (page 116), is rendered using the color values of the reference frame-buffer as texture.

Rendering a depth-image with a resolution of $2^k \times 2^k$ pixels involves warping of 2^{2k} pixels with traditional depth-image warping techniques (or rendering about $2 \cdot 2^{2k}$ triangles in [MaMB97]). With the proposed technique, instead of warping the



Close-up showing the detail

Final render of the cup model



Snapshots of the stencil showing the splat overlap

Figure 5.6: Splatted model of a reconstructed cup. The top of the figure shows two renders from novel viewpoints. The bottom of the figure shows the per-pixel contribution of the splats.

$2^{18} = 262144$ pixels of a 512×512 reference image, a textured depth-mesh with only a few thousand textured triangles can be rendered using hardware accelerated 3D graphics at a fraction of the cost of per-pixel depth-image warping.

The depth-mesh generation and segmentation process performs all the operations in the reference view coordinate system. Therefore, whenever a depth-mesh has to be rendered, the coordinate system transformation of that reference view is used as local model-view transformation to place the depth-mesh correctly in the world coordinate system.

To reduce exposure artifacts, most image warping techniques render multiple reference images that have to be merged to synthesize a new view. In [PaSaY03] we present a novel and highly efficient blending algorithm that exploits graphics hardware acceleration and that supports per-pixel weighted blending of reference depth-images. Blending of n reference depth-meshes to synthesize a new view consists of the following basic steps:

- Select n reference depth-meshes M_i, $i = 1 \ldots n$, and their corresponding textures T_i to be used for the current view, and calculate their positional blending weights w_i with respect to the current viewpoint e.
- Adaptively triangulate the depth-meshes M_i for the current viewpoint e, and generate the segmented triangle strip representations S_i.
- Render the triangle strips S_i without illumination and texturing to initialize the z-buffer Z_e for the current viewpoint e.

- Render the triangle strips S_i again with their textures T_i and per-vertex quality ρ as alpha values enabled. The per-vertex quality measure r is Gouraud interpolated across triangles and yields a per-pixel quality measure in the alpha-channel. The result is rendered into separate color-with-alpha frame-buffers C_i. Depth-buffer evaluation using Z_e is set to read-only at this stage.
- Synthesize the new image I from buffers C_i using positional weights and alpha-blending according to the equation $I = \sum_{i=1\ldots n} w_i \cdot C_i$.
- The image I contains the per-pixel weighted result. Note that the final alpha blending factor per pixel may be less than 1.0 at this stage and a normalization of the corresponding color yields the final image.

Figure 5.7 illustrates the data flow and rendering stages of our algorithm. The following sections explain the different stages in more detail and show how our algorithm exploits hardware acceleration.

3.2.1 Depth-mesh selection and triangulation

During rendering, for each frame DMesh selects the depth-meshes M_i to be used for the current view. For a depth-mesh object, we select up to k depth-meshes M_i out of the m reference views that may be visible from the novel rendering viewpoint e. As illustrated in Figure 5.8 a reference view i is selected if the angle $\beta_{v_i, n}$ between the reference view direction v_i and the new view direction n is less than 135° which is a conservative value that will include enough reference views to render an object without holes. Based on this angle, the *angular weight*

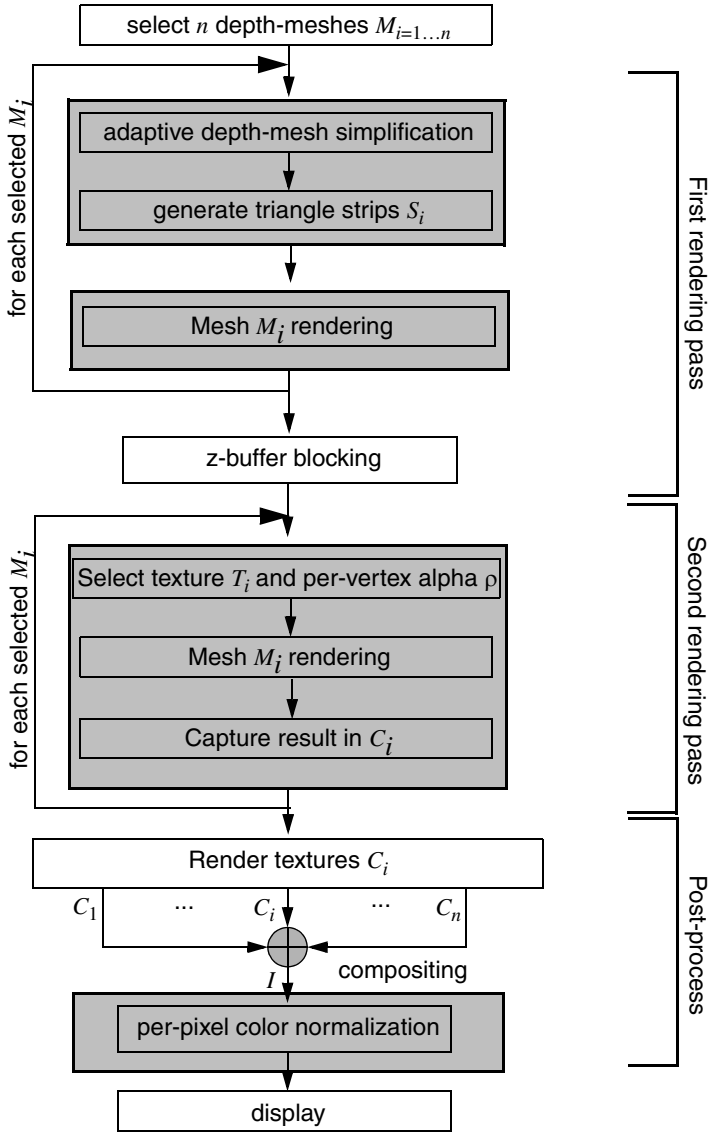


Figure 5.7: Depth-mesh rendering pipeline.

$$w_i(\beta_{v_i,n}) = \begin{cases} 1.0 & \beta_{v_i,n} < 45° \\ cos(\beta_{v_i,n} - 45°) & \beta_{v_i,n} \geq 45° \end{cases}$$

is assigned to the depth-mesh M_i. This angular blending weight avoids popping artifacts and guarantees smooth interpolation between contributing depth-meshes when rotating around the image-base object.

Furthermore, an additional *positional weight* $w_i(d_{r_i,e})$ based on the euclidean distance $d_{r_i,e} = |e - r_i|$ from the depth-mesh viewpoint r_i to the novel viewpoint e is

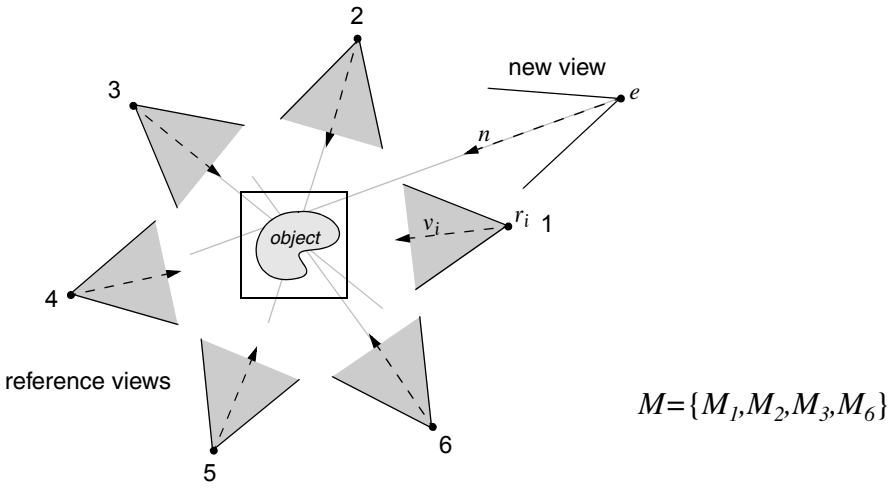


Figure 5.8: Selection of reference views. It is dependent on geometric configuration of the new viewpoint e and its view direction n with respect to the reference view points ri and directions vi. In this case the views selected are M1, M2, M3 and M6, since they meet the 135^o criterion.

associated with depth-mesh M_i. The final blending weights w_i are computed as $w_i = w_i(\beta_{v_i, n}) \cdot w_i(d_{r_i, e})$ and are then normalized such that $\sum_i w_i = 1.0$.

Each selected depth-mesh M_i is simplified view-dependently for the new viewpoint e according to an image-space geometric error tolerance of τ pixels. This is performed by a recursive top-down traversal of the quadtree triangulation hierarchy of M_i. The adaptively triangulated depth-meshes M_i are segmented and represented by a set of triangle strips S_i which contain vertices with texture coordinates into the reference view texture T_i and per vertex quality values ρ used as RGBA components.

3.2.2 Rendering pass

To achieve a smooth blending between overlapping depth-meshes M_i the rendering must allow some tolerance in the z-buffer visibility test. This is because the triangulation of the depth buffers is an approximation of the real surface, and depending on the settings and parameters of the reference views, two overlapping meshes are not

guaranteed to be identical and the triangles might inter-penetrate each other. Moreover if no tolerance is allowed the standard depth test of the 3D API's, the zBuffer test, will keep for each pixel only the contribution of closest mesh to the view point, avoiding the weighted blending of multiple meshes. Therefore multiple depth-meshes that for a pixel cover the same surface region within some tolerance ε should be blended together, and only if the z-buffer values are sufficiently different, larger than ε, the front-most depth-mesh determines the final color of that pixel

This ε-z-buffer rendering is achieved by first drawing the depth-mesh triangle strips S_i without any shading, illumination or texturing enabled to initialize the z-buffer to Z_e for the desired viewpoint e. Since the buffers require to be cleared during the following steps, a supporting stencil buffer is set to store which areas of the frame buffer will be overwritten by the rendered depth-meshes.

In a second pass and using the previously computed z-buffer Z_e in read-only mode, the meshes S_i are rendered again separately into individual color buffers C_i with a small negative offset ε along the view-direction. To initialize the background in the stencil area of the depth-meshes for each buffer C_i, a background quadrilateral is rendered with its alpha channel set to the corresponding blending weight w_i. After this, shading, alpha-blending, texturing and per-vertex color components are enabled and S_i are rendered.

Since each vertex color is set to its quality measure (ρ_i, ρ_i, ρ_i, ρ_i), Gouraud shading and texture modulation with the depth-mesh's texture T_i results in per-pixel weighted colors of the warped depth-mesh in C_i. Every pixel p in the frame buffer C_i with

interpolated quality $\bar{\rho}_p$ from Gouraud shading and texture coordinates *s,t* finally store the desired weighted color

$$C_i(x, y) = (w_i \cdot \bar{\rho}_p \cdot R_p, w_i \cdot \bar{\rho}_p \cdot G_p, w_i \cdot \bar{\rho}_p \cdot B_p, w_i \cdot \bar{\rho}_p)$$

At this point the rendered color buffer C_i is copied into texture memory for further processing as outlined in the next section. Note that rendering the selected depth-meshes S_i twice is not very costly because rendering of textured triangle strips with only a few thousand triangles is extremely efficient.

3.2.3 Compositing and normalization

The final rendering stage must now perform an image composition between the different reference view renderings C_i. By rendering a set of k screen aligned quadrilaterals, textured with their respective captured image and setting the appropriate blending function for each pass, a final image I can be obtained as

$$I(x, y) = \sum_i C_i(x, y)$$

The addition of the images C_i yields an image with intermediate pixel colors $c' = (\alpha R, \alpha G, \alpha B, \alpha)$, the α component contains the accumulated blending weight for the given pixel. Similarly as the previous rendering approach, to generate the correct colored image a normalization process needs to be performed, that is further explained in Section 3.4, Image Normalization (page 162).

As a summary of the DMesh rendering paradigm, Figure 5.9 shows a model of a cup reconstructed from 13 views. The geometry has 132,000 vertices and 265,000 faces. The geometry renders at about 4 frames per second in PC P4 2.0Ghz with and NVIDIA

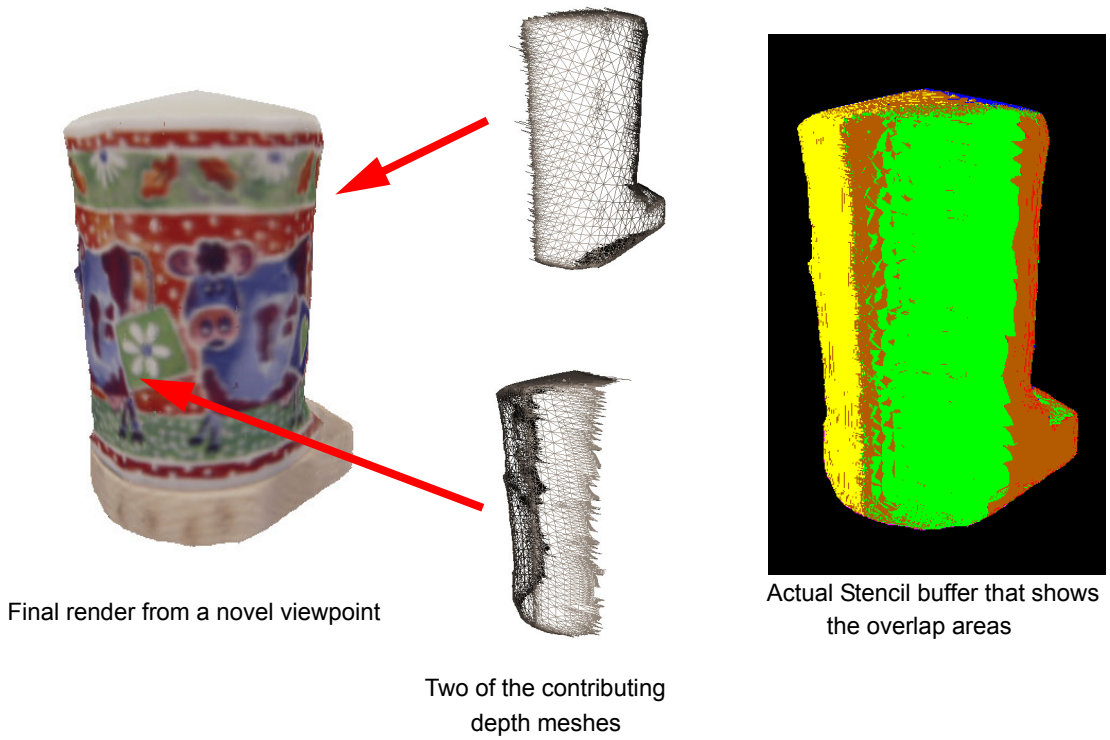


Figure 5.9: DMesh rendering pipeline example.

GeForce4 Ti 4600. The DMesh model achieves up to 15 frame per second in a view dependent mode and 125 frames per second using static triangulations.

3.3 Multitextured Mesh Rendering

As seen in Chapter 4, a *MTMesh* model consists of a single set vertices $v_1,...,v_n$ with their respective normals $n_1,...n_n$, the face connectivity $f_1,...,f_{nf}$, and then for each reference view used in the modeling pipeline it contains the intrinsic camera parameters $K_1,...,K_m$, the extrinsic ones $P_1,...,P_m$, the reference images $I_1,...,I_m$ and a set of visible faces.

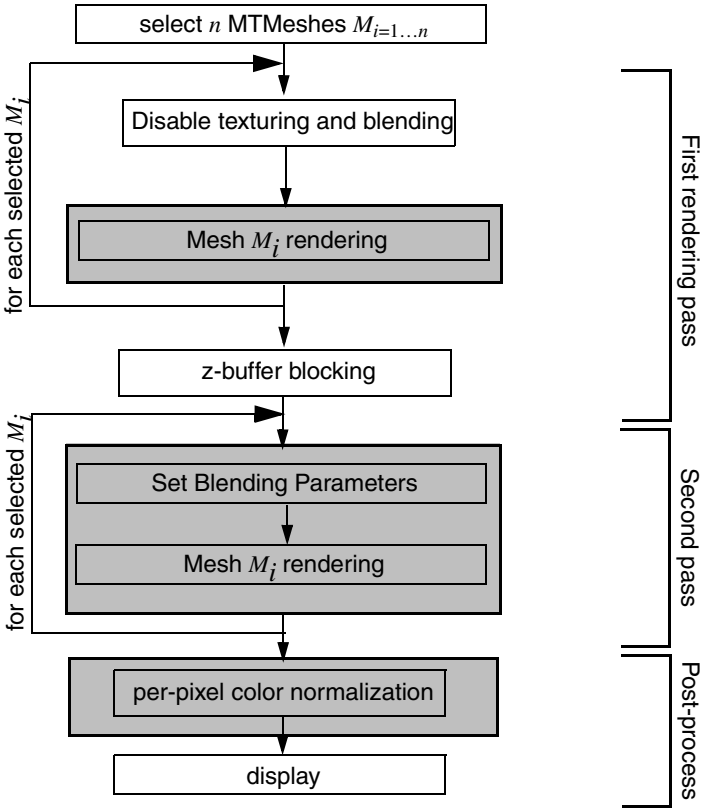


Figure 5.10: MTMesh rendering pipeline.

The complete rendering algorithm can be outlined as shown in Figure 5.10. Basically, during rendering a subset of the *m* views must be selected based on the user point of view. Then the faces are sequentially rendered for each of the active views and combined at the same time using the quality factor maps stored in the alpha channels of the image textures.

The rendering pipeline is very similar to the one proposed for the DMesh model representation. The following subsections provide the details about the different parts of the rendering algorithm.

3.3.1 Reference view selection

A similar approach as presented in Section 3.2.1 is used to select view-dependently the set of k reference views meshes M_i to be used to generate the current view. The same directional weight combined with a positional one provide the final global wight $w_i = w_i(\beta_{v_i, n}) \cdot w_i(d_{r_i, e})$ per view. Moreover, each of the selected meshes has a per-pixel quality embedded in the respective texture that accounts for the local surface orientation.

3.3.2 Render pass

The rendering is performed identically to the on proposed for the DMesh rendering algorithm. A first pass creates an occupancy depth-buffer that is blocked for the second pass so only the closest triangles, within an ε interval, to the novel viewpoint are finally rendered. In this second pass each mesh is independently rendered using the weight w_i as the default color of the mesh, modulated by the texture color, giving a final color per pixel of

$$C_i(x, y) = (w_i \cdot \bar{\rho}_p \cdot R_p, w_i \cdot \bar{\rho}_p \cdot G_p, w_i \cdot \bar{\rho}_p \cdot B_p, w_i \cdot \bar{\rho}_p)$$

being $\bar{\rho}_p$ the interpolated quality factor obtained during the modeling process (see Chapter 4, Section 4.3.2, Quality factors (page 130)).

Instead of copying the framebuffers as in the DMesh approach, we set the appropriate blending parameters and render each new mesh M_i compositing the result with the existing framebuffer. The final result is an image correctly blended but with a per-pixel weighting that does not sum up to one, requiring a normalization step (explained in Section 3.4, Image Normalization (page 162)) as a post-process.

As a summary of the MTMesh rendering method, Figure 5.11 shows an example of the same cup model, presented before, rendered using the MTMesh approach.

3.4 Image Normalization

A common trend of the 3 proposed rendering paradigms is the use of quality factors that weight the per-pixel combination of multiple color sources. The resulting image I contains pixel color values that correspond to an intermediate blending result $c' = (\alpha \cdot R, \alpha \cdot G, \alpha \cdot B, \alpha)$ where the α component contains the accumulated blending weight w. These

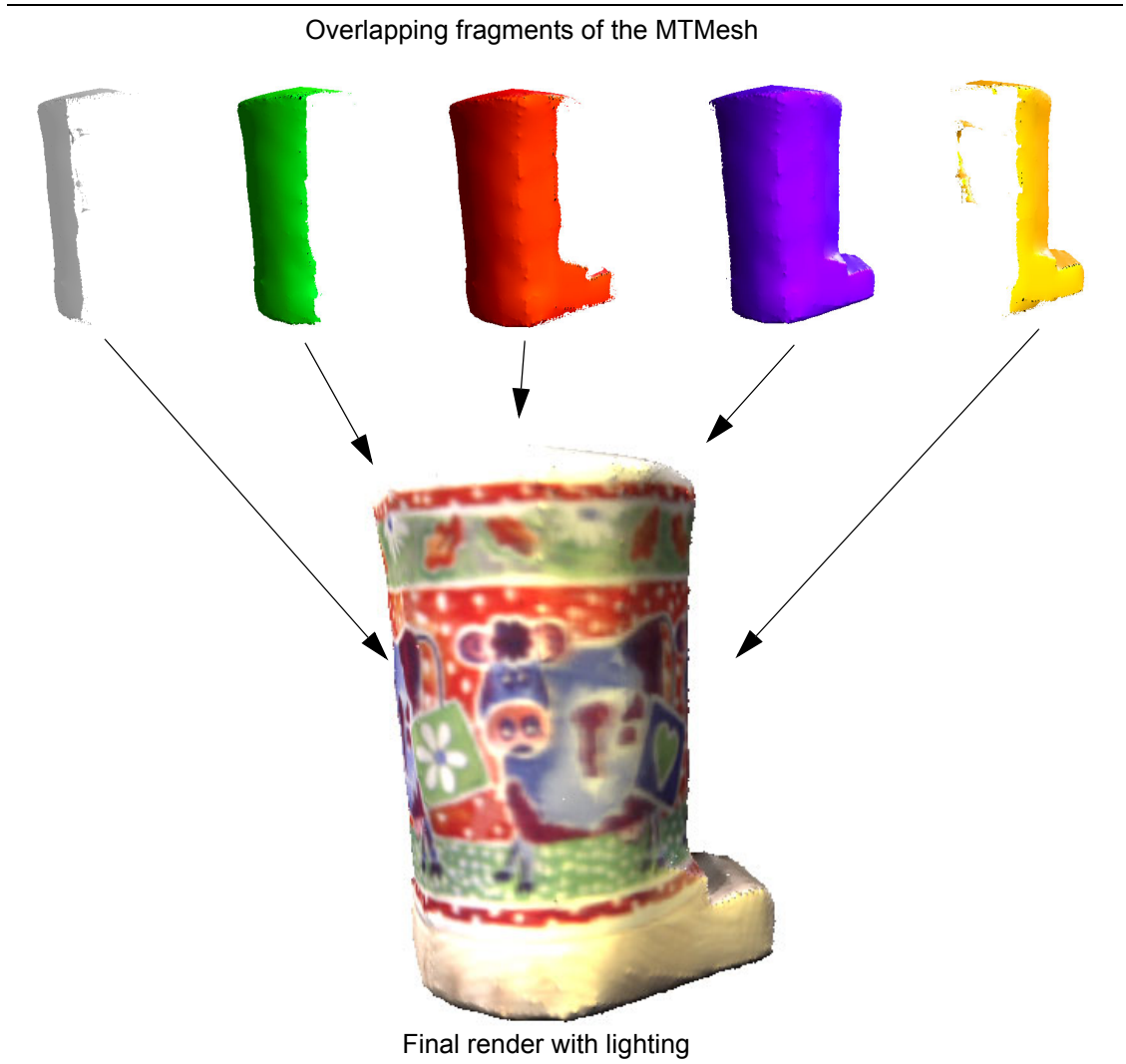


Figure 5.11: MTMesh rendering example. The current view is the superposition of 5 reference images.

color values constitute the correct proportionally blended color values, however, since the the α values are not 1.0, the final color is a scaled version. To get the final desired color $c = (R, G, B, 1.0)$ each color component of c' has to be multiplied by α^{-1}. This normalization is performed as an image post-process stage on the intermediate blending result.

Without any hardware extensions to perform complex per-pixel manipulations this normalization step has to be performed in software. However, widely available graphics accelerators now offer per-pixel shading operators that can be used more efficiently. In our current implementation, we perform this normalization using the OpenGL Fragment Program [OpGL02]or NVIDIA's Texture Shader [DSSD99] extension.

Using the a fragment program [OpGL02b], the normalization step is very simple. For every incoming fragment, its color $c' = (r, g, b, a)$ is normalized to $c = (R,G,B,1)$ by dividing each channel by the alpha value. We have come with three different solutions depending on the hardware support of the videocard.

3.4.1 Software

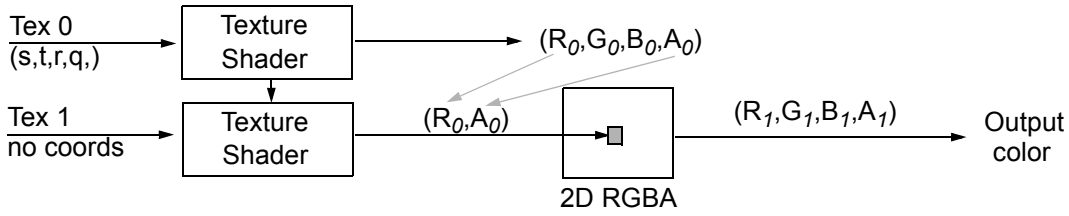
When the hardware support for OpenGL is limited, the most simple solution to correct the weighted colors is to perform a per pixel division by α using the main CPU. Therefore the technique is to download the framebuffer content to main memory from the videocard memory, access each pixel individually and multiply the R,G and B channels by α^{-1}. Finally the image must be uploaded again to the framebuffer in order to be displayed on the screen.

The main bottleneck of this approach turns out to be the downloading/uploading steps. It is a platform dependant problem, since, for example, in the PC architecture is the transfers across the AGP bus the main slowdown of the rendering framerate. Although the new AGP 8x standards provide enough bandwidth, videocard manufacturers do not optimize enough the drivers, specially for downloads. This problem is barely unnoticeable in other platforms such as SGI, where the crossbar architecture and the shared memory reduce the transfer of a complete buffer to just a memory address pointer.

3.4.2 Texture shaders

A remapping of the R, G and B values based on the value of α can be performed to compensate the illumination deficiency using the texture shaders capabilities of some commonly available videocards. During initialization time we construct a texture encoding in (*s, t*) of a look-up table of transparency and luminance values respectively, that maps values ranging from 0 to 255. The pixels of this texture encode the new intensity for a given luminance (*t*) compensated with the α transparency (*s*).

Based on this alpha-luminance map, we proceed to correct each of the *R*,*G* and *B* channels of every pixel. Using NVIDIA's texture shader extension operation `GL_DEPENDENT_AR_TEXTURE_2D_NV` and the multitexture capabilities, the graphics hardware can remap the *R* and α by a texture lookup with coordinates $s = \alpha$ and $t = R$ into our alpha-luminance map as shown in the following diagram



Then, by rendering a quadrilateral with the intermediate image I as texture-one and the alpha-luminance map as texture-two, and setting the color mask to block the G, B and α channels will compensate the red channel by α^{-1} and store it in a new image I_R.

It is important to notice that only the R and α channels are used by this dependent texture replace operation. Therefore, we need to remap the G and B channels to the R channel of two new images I_G and I_B while copying the α channel as well. This can be solved by combining the dependant texture operation with a channel remapping using NVIDIA's register combiners.

Then the complete process consists of three passes that render each a quadrilateral of the size of the visible screen with the appropriate texture mapping and channel redirection achieving the dependent texture replace operation in the images I_R, I_G and I_B. Thus by separating the RGB channels into three different images and using the αR-dependent texture replace operation we get the corrected RGB values stored in the red channel of three new images. Figure 5.12 illustrates this normalization process. Finally the three separated and corrected color channels are composited into the final image again using NVIDIA's register combiners by consecutively rendering three quads with textures I_R, I_G and I_B respectively and for each pass, blocking the writing for the other two channels.

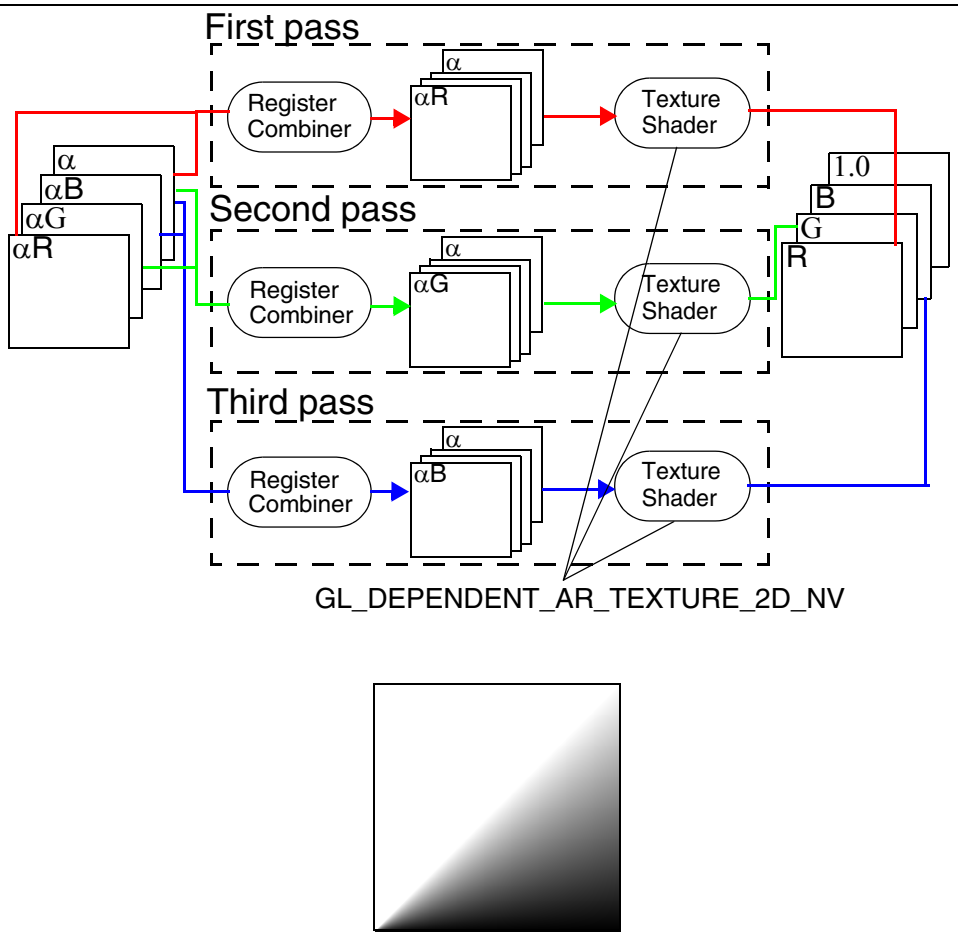


Figure 5.12: Normalization based in OpenGL texture shaders. On the bottom the alpha map texture is shown.

This approach is two orders of magnitude faster than the software method. However there is a dynamic range problem since the alpha-map texture is a discretized version of the real operation. When the weighted intensity values are close to 0, then the normalization generates artifacts in the final render.

3.4.3 Fragment programs

The last method to perform the required color normalization is to perform the same operation as the software one but within the videocard. This is possible since the introduction of the latest videocard technology of the *fragment programs* which allow to

per pixel manipulation of the rendered content. Using this capability is very easy to write a small program that performs a division by α in each of the R, G and B components of a pixel. The following code, written in OpenGL ARB fragment program language, illustrates this concept

```
!!ARBfp1.0
TEMP myColor, factor;

# Fetch the color from the texture
TEX  myColor, fragment.texcoord[0], texture[0], RECT;

# Calculate the reciprocal of the alpha channel
RCP  factor, myColor.a;

# Multiply the r,g and b components by 1/alpha
MUL  factor, factor, myColor;

# Mode result to output
MOV  result.color, factor;

END
```

This normalization process starts by copying the content of the frame buffer, which is the un-normalized render, to a temporal texture T_t, then the fragment program is activated and a quadrilateral filling the screen is rendered using T_t as texture. Then each pixel will be normalized in the fragment program.

This approach combines the simplicity and efficiency of the software approach and the speed of the texture shader based one. Moreover, since the color operations in the videocard are performed in floating point with 128 bits of precision, no artifacts are generated in the output image. However is only suitable for the latest generation of videocards.

3.5 Model re-lighting

A common characteristic of all the presented model representations is that they have been acquired using a set of images taken from a physical object under certain lighting conditions. More specifically, the scene reconstruction algorithm presented in Chapter 3, Section 4.2, Our approach (page 79), is based under the assumption that the scene has Lambertian properties and the lighting can assumed to have diffuse component only, thus the captured images do no present specularities. It is very desirable the possibility of re-lighting the reconstructed objects during the real-time rendering process to achieve high quality and realistic results.

When considering the model representation of polygons with colored vertices, it is obvious that the re-lighting can be achieved using any standard 3D rendering pipeline such as the one supported by the OpenGL API. It is just a matter of setting the appropriate lighting conditions for the scene, the appropriate material properties, using the actual colors as the diffuse values and specifying the specular and shininess components. Then by rendering with lighting enabled and specifying normals per vertex, the 3D API takes care of the rest.

It is not that clear how to integrate specular component of the lighting in the other three rendering paradigms. This is because blending and quality factors appear multiplying the colors and at the end a normalization process is required. The basic lighting equation, the Phong model, for a surface can be broken into the sum of four components: the emissiveness of the object, the ambient, the diffuse and the specular light. The complete equation is the following one

$$
\begin{aligned}
final\ color &= emissive + ambient + diffuse + specular \\
emissive &= K_e \\
ambient &= K_a L_a \\
diffuse &= K_d L_d \cdot max(N \cdot L, 0) \\
specular &= K_s L_s (max(N \cdot H, 0))^{shininess} \times facing
\end{aligned}
$$

where K_e, K_a, K_d and K_s are the material colors for each component, L_a, L_d, L_s the different light color components. N is the surface normal, L is the unitary vector pointing towards the light source and H is the bisector of L and the normalized vector pointing towards the viewpoint. *facing* is 1 if $N \cdot L > 0$ and zero otherwise.

Two of the proposed rendering algorithms combine primitives rendered with a per-reference-view quality factor is calculated in vertex basis, and a positional weighting to combine overlapping images. More specifically, these factors are

- In the DMesh approach, a mesh quality factor ρ is calculated for each vertex and encoded as the main color of the vertex, and it is combined with the positional weight w of the view and the color of the texture.
- In the MTMesh rendering approach, the vertices are rendered with a positional weight factor w and modulated with the texture, that encodes the quality factor.

In both cases the lighting components need to be multiplied by the positional weight w, since it determines how much the rendered view contributed to the final image. Since we consider the color of the object for ambient and diffuse to be identical, the application of the light coefficients will properly scale the color value to account for diffuse illumination.

a) b) c)

Figure 5.13: Example of re-lighting of cup model. a) is shaded using the Object Splat rendering approach, b) shows different lighting conditions using the DMesh algorithm and c) shows other illumination conditions using the MTMesh paradigm.

However, the specular component is usually assigned a complete different color, the highlight color, which does not contain the correction for the positional weight *w*. In order to achieve a proper re-lighting of the reconstructed object, it is necessary to carefully correct the specular color K_s by the positional factor of the given reference view. Figure 5.13 shows an example of this re-lighting process onto a reconstructed model.

4 CONCLUSION

In this chapter we have discussed different rendering paradigms for different model representations, ranging from image-based to complete geometry based ones. Then, we

have presented the main outline of our proposed solution for the Object SPlat, DMesh and MTMesh model representations.

As we have seen, very efficient rendering algorithms can be designed, that exploiting the latest improvements in hardware acceleration, achieve very good results visually speaking. A new way of blending textures and surfaces has been presented that is not constraint to the use of blending factors that sum up to one, since a normalization post process can be efficiently implemented.

Moreover, the presented algorithms can manage re-lighting of the reconstructed objects, increasing the realism of the final renders.

CHAPTER 6 RESULTS AND CONCLUSIONS

1 INTRODUCTION

The previous chapters have introduced the theoretical details of the complete pipeline for reconstruction of objects from images. In this chapter we present detailed results of the different stages of the proposed pipeline, pointing out the good features as well as the weaknesses of the system. Additionally, as a conclusion, some reflections on the work, its limitations, its applicability and future work are discussed.

2 PIPELINE WALKTHROUGH

We present this first dataset providing all specific details of the different stages of the pipeline to give a walkthrough of the system. The *monster* dataset is a 22cm tall stone gargoyle statue and an initial set of 16 images, captured with a digital camera, at a

resolution of 1024 *x* 1024 pixels is obtained by placing the object in a turn table and rotating the platform unknown angle increments from the starting position. The background of the images is segmented manually to facilitate the reconstruction process. Moreover, since the figure does not present high color frequency textures on its surface, a set of 48 fiducials positioned on the object are used as marks. The set frames used to reconstruct the object are shown in Figure 6.1.

The set of tracked features consists of the fiducial marks on each frame and both the marks and the correspondences are set manually. It is important to notice that since the correspondence problem is solved manually, some of the features that appear in the first frames that re-appear in the last ones, have been tagged with the same ID. This is noticeable in the bottom left corner of the measurement matrix image.

The calibration algorithm automatically divides the sequence into 7 subsequences using a threshold of common points of 70% and an minimum inliers threshold for keyframe selection of 100%. provides the statistics about the calibration process. The selection time corresponds to the frame fragmentation and keyframe analysis. The calibration is the total amount of time spent during the calibration of all the fragments. Bundle time is the accumulated time for all the Levenberg-Marquardt optimizations. The sum of these three time is approximately the total time (excluding I/O processes).

Table 6.1: Calibration statistics for monster dataset.

Total time (s)	Selection (s)	Calibration (s)	Bundle (s)	Mean Error (pixels)	Stdev Error (pixels)
3.2884506	0.4234	0.9621	1.9026	(x) 0.6926 (y) 0.8211	(x) 0.6420 (y) 0.6775

Figure 6.1: Input data for *monster* dataset. The 16 images used to reconstruct the monster dataset use a set of fiducials that can be seen as red dots on the surface of the object. At the bottom we see an image of the sparse measurement matrix for this dataset.

Mean error and the associated standard deviation error (for both x and y) are measured in pixels of 2D reprojection errors resulting from projecting the recovered points using the recovered camera geometry and parameters. Since background has been previously segmented, the final shape is created mostly by the visual-hull algorithm, being the space carving a small refinement. We have performed reconstructions with 4 different voxel resolutions in order to compare model size and complexity as well to verify the relation between the size and the obtained framerate for the different algorithms. Table 6.2 shows the reconstruction statistics for each of the resolutions.

Table 6.2: Scene reconstruction for the monster dataset.

Voxel Resolution	#VH Voxels	VH comp. time (s)	#SC Voxels	SC comp. time (s)	#SC iterations
50	11410	0.4549	10549	15.885	17
150	308729	1.0126	285239	118.773	61
250	1429475	2.0064	1349548	207.642	43
350	3922455	3.4756	3745389	459.824	11

As we can see from the previous table, the visual hull does an excellent job computing a correct shape. Moreover, since the reconstructed object does not present textures with a large color variability, the space carving approach does not find too many color inconsistencies in the reconstruction from the visual hull. However, most of the carving is done on areas such as the elbow of the statue and the chin, where there are local concavities that the visual hull is unable to detect.

It is important to notice that the proposed implementation of the visual hull is extremely efficient, and most of the calculation time is spent transferring color buffers from the video card to main memory and sweeping the plane to find the voxels. If instead of processing in CPU one wants to store the resulting slices onto a 3D texture

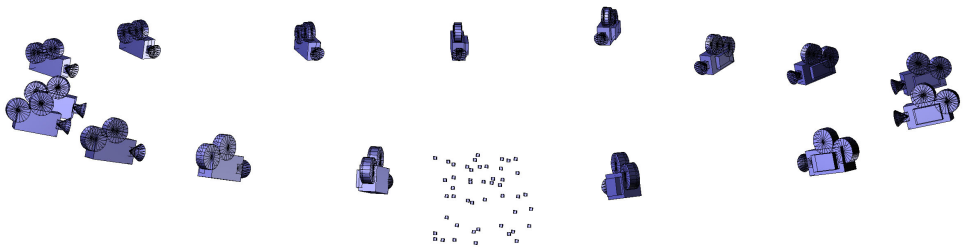
(when fully supported by the hardware) this computation time reduces drastically. Figure 6.2 shows the reconstructed cameras in respect to the object and also the voxelized meshes (before relaxation) for the four resolutions.

The next step in the pipeline is the model reconstruction, that converts the voxel data into different model formats. As a pre-processing of the reconstructed voxels, a standard morphological refinement is performed using erosions and dilations. The result is a smoother voxel dataset, which will generate better reconstructions at the end. Once this is finished, the different model representations can be obtained. In order to avoid the use of excessive number of images as textures, the initial sequence is decimated to 5 images covering as much of the surface of the object as possible.

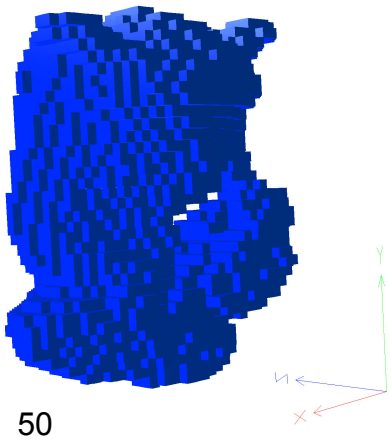
In Table 6.3 we present the computation times for the different approaches. The initial morphological process consists of 3 dilations followed by 3 erosions, and as it can be seen is the most expensive process and is very dependant on the voxel resolution. The reference view preprocessing step extracts a mesh from the voxelized reconstruction and relaxes it to smooth out the voxelized pattern. The rest of the processing to generate each of the model representations presents a low cost in terms of computation time.

Table 6.3: Computation time of the different model representations.

	Morphological time (s)	**Ref View preproc (s)**	**PLY time (s)**	**MTMesh time (s)**	**DMesh time (s)**	**O-Splat time (s)**
50	-	0.6480	2.5067	2.5384	1.0876	0.0737
150	8.9102	3.4483	3.4927	3.2500	2.0688	0.6626
250	41.1023	10.5458	5.4306	4.8924	5.4896	1.9444
350	112.5120	23.5525	8.4156	7.2354	7.5708	4.6749

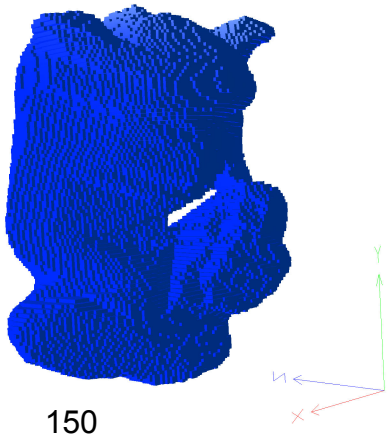


Calibrated image sequence



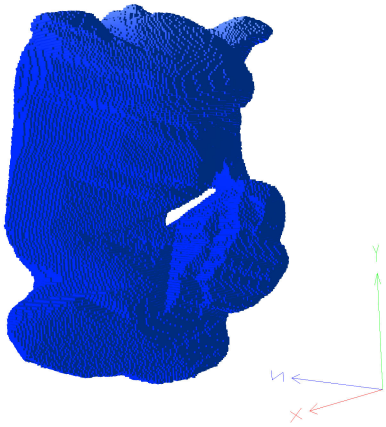
50

5378 vertices
10888 faces



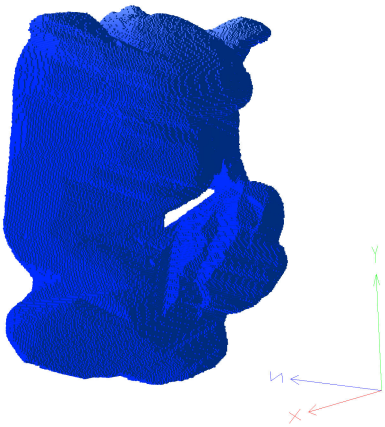
150

51251 vertices
103106 faces



250

146289 vertices
294314 faces



350

291945 vertices
587584 faces

Figure 6.2: Calibrated sequence and reconstructed scene for the *monster* dataset. On top of the figure there is a 3D scene showing the relative camera object configuration. On the bottom, the 4 different resolutions of the reconstructed scene are shown.

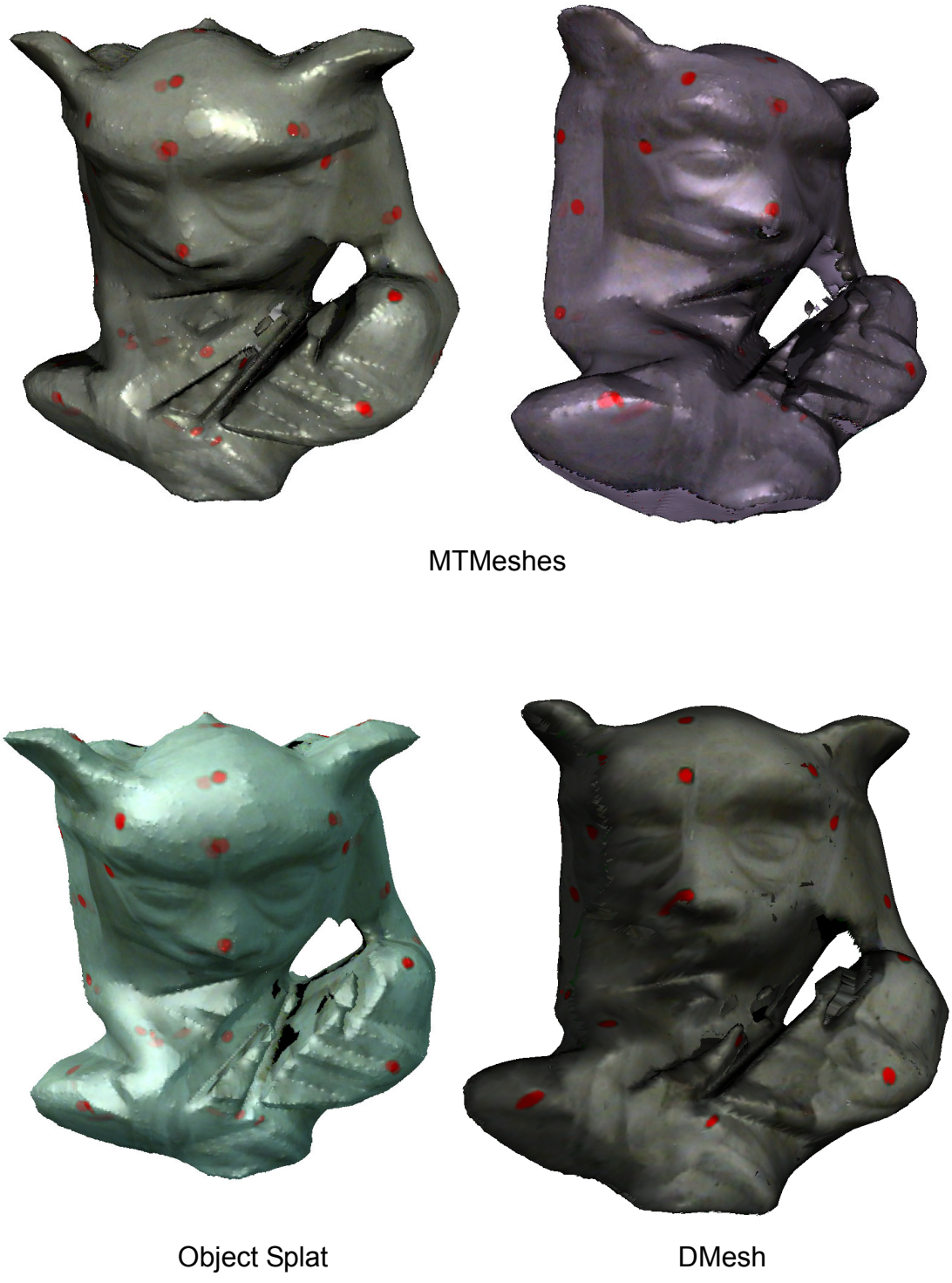


Figure 6.3: Rendering examples. Different lighting conditions, using the proposed model representations are presented.

Once the model representations have been created, the results can be rendered using the proposed rendering schemes. In Table 6.4 we show the different frame rates for the 4 calculated reconstructions. The PLY column can be used as the cost reference, since it is a pure colored geometric model, rendered using standard procedures. The object splat methodology is extremely efficient, specially when large mesh resolutions are used. The DMesh representation in its non-view dependant version is very fast, however the visual quality is inferior to the other approaches.

Table 6.4: Average frames per second for the different representations.

	PLY	Object splat	DMesh(static)	DMesh(v dep)	MTMesh
50	62	45	250	21	30
150	31	10	450	18	20
250	10	8	450	17	11
350	5	5	420	16	7

Figure 6.3 illustrates the re-lighting capabilities of the proposed algorithms. Different light colors and material properties have been used to obtain the images, and, as it can be seen, the specular highlights blend perfectly with the object while maintaining the original color of the texture and the details such as the eyes.

3 PERSEPOLIS

Other research fields such as archeology are becoming aware of the great advantages of 3D visualization systems and computerized databases of images. Since in their daily routine archeologists take a large amount of images and measurements of their sites for documentation purposes, a system to extract 3D reconstructions of the

archeological sites can be very beneficial to enhance the quality of the work as well as its diffusion to the general public.

As a case study of the techniques developed throughout this work and as an application to the archeological field, we have obtained some reconstructions from the site of Persepolis. This magnificent palace complex was founded by Darius the Great around 518 B.C., although more than a century passed before it was finally completed. Conceived to be the seat of government for the Achaemenian kings and a center for receptions and ceremonial festivities, the wealth of the Persian empire was evident in all aspects of its construction. The splendor of Persepolis, however, was short-lived; the palaces were looted and burned by Alexander the Great in 331-330 B.C. The ruins were not excavated until the Oriental Institute of the University of Chicago sponsored an archaeological expedition to Persepolis and its environs under the supervision of Professor Ernst Herzfeld from 1931 to 1934, and Erich F. Schmidt from 1934 to 1939.

The dataset shown here, the *two-headed-horse*, contains 702 frames of 720x480 pixels each. We use our implementation of the KLT tracker [ShTo94], in order to obtain up to 300 features per frame with automatic replacement of lost ones. Table 6.5 shows the statistics for the calibration process, that uses a threshold of common points of 50% and an minimum inliers threshold for keyframe selection of 90%.

The sequence of keyframes is decimated to a final set of 12 frames (top of Figure 6.4) that will be used during the volumetric reconstruction. The carving process is set to use an initial resolution of 200x176x138 voxels and it performs 165 iterations in 4225.6

Figure 6.4: Reconstruction of *two-headed-horse* dataset. The top mosaic shows the final 12 frames used in the final reconstruction of the model. The lower image is a novel view generated by rendering the reconstructed model.

Table 6.5: Calibration results for the *two-headed-horse* dataset.

#frames	#key frames	#frags.	Total time	Selection time	Calibration time	Bundle time
702	94	4	541.6919	11.0237	47.39421	483.2740

Mean Error X	0.6297	**Error Stdev X**	0.5930
Mean Error Y	0.5959	**Error Stdev Y**	0.6456

seconds, reducing the occupancy of the initial volume to a 43%, generating a final solid model of 4857600 voxels.

Although the overall carving time is large, this is because no background information is available from the images and the visual hull is unable to help in the reconstruction by pre-carving the model. This example is a clear case that shows the potential of the voxel carving techniques for background segmentation and object reconstruction. Bottom of Figure 6.4 shows the final reconstructed object in a novel rendering position.

4 OTHER EXAMPLES

Several datasets, both synthetic and real imagery have been used throughout the development of the pipeline in order to verify the correctness of each step. Furthermore, we have tried to maintain a certain degree of generality in the system to avoid strict constraints by providing a wide variety of inputs.

The proposed pipeline is design to use long video sequences, provided that tracked measurements can be obtained. The following example, the dataset *head*, consists of a video sequence with 370 frames taken with a handheld camcorder by moving it around a

person's head. To extract the 2D tracking features we use again our implementation of the KLT tracker [ShTo94], in order to obtain up to 500 features per frame with automatic replacement of lost ones. Figure 6.5a shows an image of the resulting sparse measurement matrix.

The calibration statistics are presented in Table 6.6. The achieved calibration shows a very low reprojection error even in the presence of outliers and noise in the input measurements.

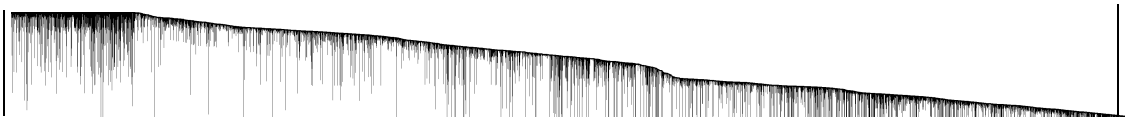
Table 6.6: Calibration results for the *head* dataset.

#frames	#key frames	#frags.	Total time	Selection time	Calibration time	Bundle time
370	148	5	639.0254	9.7777	126.3247	501.29223

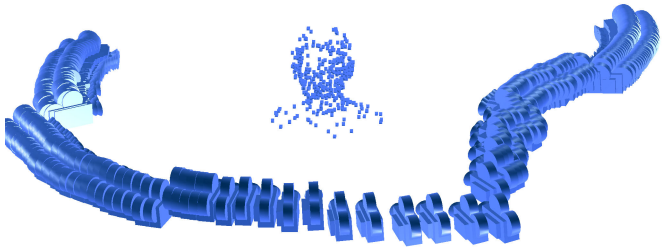
Mean Error X	0.4875	**Error Stdev X**	0.5737
Mean Error Y	0.7113	**Error Stdev Y**	0.660

A graphical reconstruction of the recovered cameras positions and recovered points is shown in Figure 6.5b, together with some example frames of the sequence.

The scene reconstruction stage is performed using a decimated set of 16 frames, 3 of them with the background segmented manually so the visual hull provides a close to final reconstruction. This is large improvement over a pure carving reconstruction since there is not much color and contrast difference between the foreground and background in some of the frames. If a carving based reconstruction is attempted we get a result that, although is compatible with the images is not a desirable reconstruction (as Figure 6.6 illustrates).

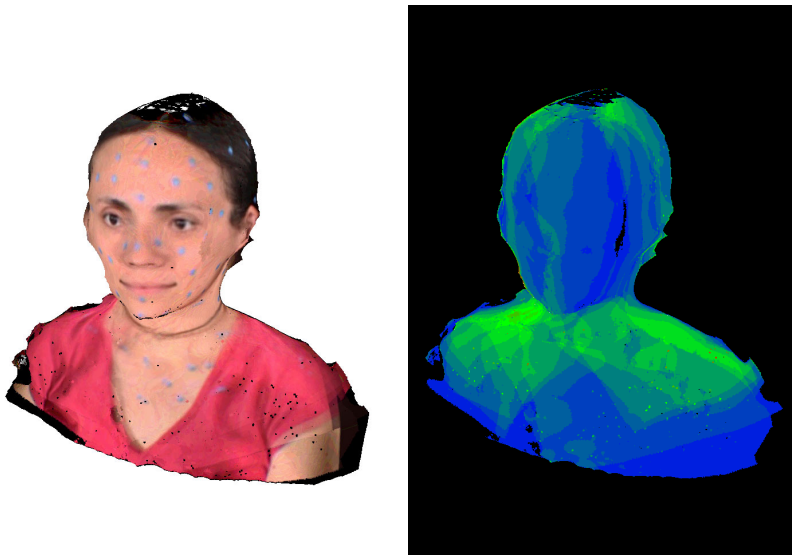


a) Sparse measurement matrix



b) Recovered camera path

c) Some reference views



d) Novel view and overlap of selected reference views

Figure 6.5: *head* dataset. a) image of the measurement matrix, b) recovered scene structure, c) some of the reference views and d) novel view and the overlap of reference views.

Figure 6.6: Example of space carving based reconstruction. The quality input images is low in contrast and color differences between foreground and background.

Using a few manually segmented frames improves considerably the reconstruction by means of the visual hull, as shown in Figure 6.5d where an novel view of the reconstructed model together with its associated view overlap map is shown.

Other reconstructed objects have been obtained from a variety of sources, some of them with calibration information pre-calculated, some of them with tracking information readily available and some other ones with just the initial image set. In Figure 6.7 we show some of these reconstructions.

Figure 6.7: Some of other reconstructed objects.

5 CONCLUSIONS AND FUTURE WORK

The principal objective of this work is to develop a software pipeline, based on image based modeling techniques, that allows the automatic reconstruction of physical objects with all their properties (shape, color and texture) properly recovered. Based on different representations of the reconstructed models, the appropriated rendering algorithms are be provided to maximize the utilization of the recovered data.

The first stage of the proposed requires a set of features tracked on the sequence of images. As we have mention before, the selection and tracking of these features is not a part of this project. However, it is a very important initial preprocessing that influences considerably the overall results. In order to be able to run experiments we have used different techniques to obtain a reliable set of features suitable to be calibrated. Tracking-wise, the best case scenario comes when using a video sequence, since then automatic trackers can be used. More specifically we have used our implementation of the KLT algorithm [ShTo94] as our main automatic tracker. A limitation the current implementation is that the tracking process is a forward tracking without prediction. There are several approaches to enhance this tracker and we leave as future work a careful review of them in order to improve our limited tracker implementation.

The proposed pipeline starts with a camera calibration process which is a critical problem in applications such as augmented reality and image based model reconstruction. When constructing a 3D model of an object from an uncalibrated video sequence, dealing with large amounts of frames and managing self occlusions of parts of the objects are common and difficult problems. In this work we present a different novel

approach based on a divide and conquer strategy to fully calibrate a long sequence of images with a high degree of feature occlusions such as video sequences of objects in rotating platforms. The complete sequence is automatically divided into subsequences and, in each of them, a set of keyframes is selected and calibrated, recovering both camera parameters and structure of the scene. When the different subsequences have been successfully calibrated a merging process groups them into a single set of cameras and reconstructed 3D features of the scene. A final non-linear optimization is performed in order to reduce the overall reprojection error.

One advantage of the presented approach is that it allows to recover an Euclidean reconstruction of the scene without any initial solution or prior information, which is one of the drawbacks of most of the existing methods. Another important feature is that the entire calibration process is based on solving linear systems using the SVD decomposition algorithm. The knowledge of the geometric meaning and rank properties of the different transformations represented by the matrices of the process allows to enforce a valid Euclidean reconstruction. The proposed solution is designed to be versatile in respect to the input data allowing the use of (1) automatically tracked video sequences, (2) manually tracked sequences which usually contain less frames or (3) a set of still images with features and correspondences manually selected. Here on, we assume that the input data is given as a sequence of images and a list of features in each image and the correspondences with the rest of the frames.

There is still work to do in this stage of the pipeline, since there are some camera configurations that we are unable to calibrate. Also, the projective reconstruction approach should be enhanced with a more robust algorithm that presents a faster

convergence. Moreover, the sequence merging is performed in metric space, once each fragment has been calibrated, which is conceptually easy to perform, however when between two fragments there is a very different focal length settings, this merging process becomes extremely difficult, and sometimes impossible. A projective space merging can a be a potential solution, and although we have obtained some unsuccessful results in this direction, further analysis should be conducted.

The second stage of the pipeline, the scene reconstruction, has the objective of extracting a voxelized reconstruction based uniquely on the reference views and the calibration information. As one can imagine this is not an easy task at all. Moreover, in the case that a solution is obtained, it does not necessarily need to be what one was expecting. This is because the reconstruction from images is ill-posed problem unless a large amount of images is provided, covering all possible features of the model, or additional information is introduced in the pipeline. This is the main reason why we introduce the visual hull calculation: it is a very fast way of carving away thousands of candidate voxels by just using a few background/foreground segmented images. Moreover, as we have seen in the previous examples, the complete sequence does not need to be segmented, but just a few of the frames. Therefore, when no automatic background segmentation is available, we can manually process 2 or 3 frames and use those to reduce considerably the initial voxel set to be used in the color consistency based carving process. In this way, we can consider the color carving as a refinement process.

Surface reconstruction from images is still an open problem, and is a very difficult one. Carving algorithms prove to be a decent approach, specially when no segmented

information is available, since the color consistency is a well established theory. However they are highly dependent on the implementation and also on the quality of the input images. Moreover when reconstructing objects with flat surfaces (such as buildings or rooms) the reconstructions rarely present flat surfaces, thus generating very geometrically heavy models. We believe that this stage in the pipeline has to be extended into a hybrid approach that integrates a solid primitive based modeling approach with space carving and long baseline stereo reconstruction, in such a way that the methods complement each other introducing constraints on the final shape.

Continuing with the pipeline, in the modeling stage we have introduced three different representations using different types of primitives. On one hand we have the Object Splat approach, which models the object as a set of unconnected points, and uses a point-octree hierarchy that provides support for level-of-detail. The resulting models are very compact and efficient to render. The concept the homogenous covariance matrix develop by Pajarola *et al.* introduces a very nice and clean way of building a hierarchy at a low computational cost.

The second model representation, DMesh, uses terrain triangulation algorithms to obtain meshes of range images. When an initial mesh is reconstructed, it is very easy to obtain a depth image for each of the reference views. Then we triangulate it using the image itself as a texture map and we obtain a novel perspective of the object by rendering each of the reference views. As we have seen, two flavours of this depth-mesh approach are proposed: (1) a non-view dependent or static triangulation and (2) a view dependant one that can be re-triangulated at interactive speeds.

The third model representation, the MTMesh, is in some sense similar to the DMesh approach. An indexed list of fragments of the original geometry is created for each reference view, together with a quality factor that weights the contribution of a polygonal face to the reference view. This approach does not use the mesh approximation of the DMesh method and that will give more accurate renderings, since the original geometry has been created to be compatible with the input images. The drawback is the cost of rendering, which is closer to rendering the complete mesh.

There are several improvements that can be further investigated such as the LOD support on the MTMesh representation, improvements on the mesh approximation of the DMesh approach (so the geometry gets closer to the real one). Other representations could studied, such the ones based on Lumigraphs and Lumispheres and a better way of integrating the texture information from overlapping views. The Object Splat representation can be enhanced by adding more precomputed information in the hierarchy to unload the rendering pipeline processing, specially by pre-calculating the required transformations from the generic triangle to the object space triangular splat. This will surely introduce more complexity in the data structure as well as more storage, but the acceleration should be considerable. Another potential solution is to have a precalculated set of sampled screen projected triangles, store them in the videocard memory and during runtime determine which one has to be rendered.

The final stage of the presented pipeline is the rendering loop. We propose different rendering paradigms tailored to each of the corresponding model representations. All of them achieve interactive rendering speeds while maintaining and overall image quality, which is one of the goals of this work. Moreover the three approaches rely extensively

in the use of the OpenGL API as well as the latest technological improvements in hardware acceleration. The Object Splat algorithm has proven to be very reliable with excellent rendering quality at decent speeds. We have notice that the best performance of this algorithm is for very large models, where keeps an interactive framerate, while standard geometric approaches reduce considerably their performance. On the other hand the DMesh approach has shown to be the fastest one, however the final quality is not as good as the other approaches. This is the trade off price of the extreme mesh simplification that the Restricted Quadtrees provide.

REFERENCES

[ACW+99] D. Aliaga, J. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *Proc Symposium on Interactive 3D Graphics*, pp. 199–206. ACM SIGGRAPH, 1999.

[AlLa99] D. Aliaga and A. Lastra. Automatic image placement to provide a guaranteed frame rate. In *Proc SIGGRAPH 99*, pp. 307–316. ACM SIGGRAPH, 1999.

[AlBW01] B.D. Allen, G. Bishop, G. Welsh, *Tracking: Beyond 15 Minutes of Thought*, SIGGRAPH 2001 Tutorial 11.

[Amat02] J. Amatller, *Surface reconstruction from a sequence of depthmaps*, MSc. Thesis, University of California, Irvine, January 2002.

[AzHP93] A. Azerbayejani, B. Horowitz and A. Pentland, Recursive estimation of structure from motion using relative orientation constraints, in *Proc. of the International Conference of Computer Vision and Pattern Recognition*, IEEE Computer Society Press, pp.294-299, June 1993.

[AzPe95] A.J. Azarbayejani and A. Pentland, Recursive estimation of motion, structure, and focal length, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 318(6), 562-575, 1995.

[Blin88] J. Blinn, *Me and my (fake) shadow*, in IEEE Computer Graphics and Applications, vol. 9, no. 1, pp.. 82-86, 1988.

[BoPe98] J.Y. Bouguet and P. Perona, 3D Photography on your Desk, in *Proc. 6th Int. Conf. Computer Vision*, Bombay, India, pp. 43-50, January 1998.

[BrCi00] A. Broadhurst and R. Cipolla, *A statistical consistency check for the space carving algorithm*, in *Proc. 11th British Machine Vision Conference*, pp. 282-291, 2000.

[BrDC01] A. Broadhurst, T.W. Drummond and R. Cipolla, A probabilistic framework for space carving, in *Proc. of International Conference on Computer Vision*, I, pp. 282–291, 2001.

[BrCe86] T.J. Broida and R. Chellappa, Estimation of Object Motion Parameters from Noisy Images, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1), pp. 90-99, 1986.

[BrCC90] T.J. Broida, S. Chandrashekhar, and R. Chellappa, Recursive 3-D motion estimation from a monocular image sequence, in *IEEE Transactions on Aerospace and Electronics Systems*, Vol 26, No 4, July 1990, pp 639-656.

[BrFA98] T. Brossky, C. Fermuller and Y. Aloimonos, Shape for video: beyond the epipolar constrain, in *DARPA Image Understanding Workshop*, pp. 1003-1012, 1998.

[BuBM01] C. Buehler, M. Bosse, L. McMillan, S. Gortler and M. Cohen, *Unstructured Lumigraph Rendering*, in *Proc. SIGGRAPH'01*, pp. 425-432, 2001

[ChBL99] C. Chang, G. Bishop and A. Lastra. Ldi tree: A hierarchical representation for image-based rendering. In *Proc SIGGRAPH 99*, pp. 291–298. ACM SIGGRAPH, 1999.

[ChQS99] R. Chellappa, G. Quian and S. Srinivasan, Structure from Motion: Sparse Versus Dense Correspondence Methods, in *IEEE Transactions on Aerospace and Electronics Systems*, pp. 492-499, 1999.

[ChMe99] Q. Chen and G. Medioni, Efficient, iterative solution to M-view projective reconstruction problem, *in Proc. IEEE Computer Vision and Pattern Recognition* (1), pp. 55-61, 1999.

[Chen00] Q. Chen, *Multi-view Image-Based Rendering and Modeling*. PhD. Dissertation, Computer Science USC, 2000.

[ChWi93] S. E. Chen and L. Williams, View interpolation for image synthesis, In *Proceedings SIGGRAPH 9*3, pp. 279–288, 1993.

[Cano] Canoma by MetaCreations, http://www.metacreations.com/

[CrAn97] P. Crossno and E. Angel, Isosurface Extraction Using Particle

Systems, in *Roni Yagel and Hans Hagen, editors, Proceedings of Visualization '97* (Phoenix, AZ, October 1997), pp. 495-498, IEEE Computer Society Press, 1997.

[CrAn99] P. Crossno and E. Angel, “Spiraling Edge: Fast Surface Reconstruction from Partially Organized Sample Points”, in *IEEE Visualization*, Oct 1999.

[CuMS99] W. B. Culbertson, T. Malzbender, and G. Slabaugh. *Generalized voxel coloring*. in *Proc. Int. Workshop on Vision Algorithms*, volume 1883 of Lecture Notes in Computer Science, pp. 100-115. Springer-Verlag, 2000.

[DaCV97] L. Darsa, B. Costa and A. Varshney. Navigating static environments using image-space simplification and morphing. In Proc *Symposium on Interactive 3D Graphics*, pp. 25–34. ACM SIGGRAPH, 1997.

[DaCV98] L. Darsa, B. Costa and A. Varshney. Walkthroughs of complex environments using image-based simplification. In *Computers & Graphics*, 22(1):25–34, 1998.

[DeTM96] P. E. Debevec, C.J. Taylor, and J. Malik, Modeling and rendering architecture from photographs: a hybrid geometry and image-based approach, in *Proc. of SIGGRAPH'96*, pp. 11-20, 1996.

[DeBY98] P. E. Debevec, G. Borshukov, and Y. Yu, Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping, in *9th Eurographics Rendering Workshop*, Vienna, Austria, June 1998.

[DoSp01] S. Dominé and J. Spitzer. *Texture shaders*. NVIDIA Developer Documentation, 2001.

[DSSD99] X. Decoret, G. Schaufler, F. Sillion and J. Dorsey. Multi-layer impostors for accelerated rendering. In *Proc EUROGRAPHICS 99*, pp. 61–72, 1999.

[Eber01] D. Eberly. *3D Game Engine Design*. Morgan Kaufmann Publishers. ISBN 1-55860-593-2.

[EiSG99] P. Eisert, E. Steinbach and B. Girod, Multi-Hypotheses Volumetric Reconstruction of 3-D Objects From Multiple Calibrated Camera Views, in *Proc. of the International Conference on Computer Vision*,

Vol. 1, pp. 415-425, 1999.

[Delt] DeltaSphere, http://www.deltasphere.com/

[FaKe98] O. Faugeras and R. Keriven, Complete dense stereovision using level set methods, in *IEEE Transactions on Image Processing*, Vol. 7, pp 336-344, March 1998.

[FiZi98] A. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. in *Proc. European Conference on Computer Vision*, pp. 311-326. Springer-Verlag, June 1998.

[FuTV97] A. Fusiello, E. Trucco, and A. Verri. Rectification with unconstrained stereo geometry. In A. F. Clark, editor, *Proceedings of the British Machine Vision Conference (BMVC `97)*, pp. 400-409, University of Essex, September 1997. BMVA Press, United Kingdom.

[Fusi00] A. Fusiello, Uncalibrated Euclidean reconstruction: a review, in *Image and Vision Computing*, 18, pp. 555-563, 2000.

[ChWi93] S. Chen and L. Williams. View Interpolation for Image Synthesis. In *Proc. of SIGGRAPH'93*. pp 279-288. ACM SIGGRAPH, 1993.

[Gibs98] S. Gibson, Constrained Elastic SurfaceNets: Generating Smooth Surfaces from Binary Segmented Data, in *Proc. Medical Image Computation and Computer Assisted Interventions*, Oct. 1998, pp. 888-898

[GCH+02] S. Gibson, J. Cook, T.L.J. Howard, R.J. Hubbold and D. Oram. Accurate Camera Calibration for Off-line, Video-Based Augmented Reality. in *IEEE and ACM International Symposium on Mixed and Augmented Reality*, Darmstadt, Germany, September 2002.

[Giro93] B. Girod, What's Wrong with Mean-squared Error?, in *A. B. Watson, editor, Digital Images and Human Vision*, pp. 207-220, MIT Press, 1993.

[GoVa96] G.H. Golub, C.F. Van Loan, *Matrix Computations. Third Edition*, Ed. Johns Hopkins, 1996.

[GGSC96] S. J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen. The lumigraph, in *Proceedings SIGGRAPH 9*6, pp. 43–54. ACM SIGGRAPH,

1996.

[Gros01] M. Gross. Are points the better graphics primitives? In *Computer Graphics Forum 20(3), 2001. Plenary Talk* Eurographics 2001.

[HaKa00] M. Han and T. Kanade, Creating 3D Models with Uncalibrated Cameras, in *IEEE Computer Society Workshop on the Application of Computer Vision* (WACV2000), **9**(2), pp. 137-154, 2000.

[HaZi00] R. Hartley and A. Zisserman, *Multiple View Geometry*, Cambridge University Press, 2000.

[HeBS99] A. Heyden, R. Berthilsson and G. Sparr, An iterative factorization method for projective structure and motion from image sequences, in *Image Vision and Computing*, **17**, pp. 981--991, 1999.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. Surface reconstruction from unorganized points, in *Proc. SIGGRAPH 1992*, pp. 71-78. 1992.

[Hopp96] H. Hoppe, Progressive meshes. Computer Graphics (SIGGRAPH 1996 Proceedings), pp. 99-108. 1996.

[Imag] Image Modeler by RealViz, http://www.realviz.com/

[JeAP99] T. Jebara, A. Azarbayejani, and A. Pentland, 3-D Structure from 2D motion, in *IEEE Signal Processing Magazine, "3D And Stereoscopic Visual Communication",* May 1999, Vol. 16. No. 3.

[KuSe98] K. Kutulakos and S. Seitz, “What do N photographs tell us about 3D Shape?”. TR680, Computer Science Dept. U. Rochester, January 1998.

[KaTr99] F. Kahl and B. Triggs, Critical Motions in Euclidean Structure from Motion, in *Proc. CVPR99*, vol 2, pp. 366-372, 1999.

[KuSe00] K. Kutulakos and S. Seitz, A theory of shape by space carving, in *International Journal of Computer Vision*, 38(3), pp. 198-218, 2000.

[Laur94] A. Laurentini, The Visual Hull Concept for Silhouette-Based Image Understanding, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2), pp. 150-162, February 1994.

[LeHa96] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings SIGGRAPH 9*6, pp. 31–42. ACM SIGGRAPH, 1996.

[LPC+00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings SIGGRAPH 2000*, pp. 131–144. ACM SIGGRAPH, 2000.

[LiMe95] C.W. Liao and G. Medioni. Surface Approximation of a Cloud of 3D Points, in *Graphics Models and Image Processing*, 57(1):67-74, 1995

[Lok01] B. Lok, Online Model Reconstruction for Interactive Virtual Environments, in *Proc. of Symposium on Interactive 3D graphics*, 2001.

[LoCl87] W.E. Lorensen, and H.E. Cline, Marching Cubes: a high resolution 3D surface reconstruction algorithm, in *Proc. SIGGRAPH'87*, Vol. 21, No. 4, pp. 163-169, 1987.

[Max96] N. Max. Hierarchical Rendering of Trees from Precomputed Multilayer Z-Buffers. In Proc. Eurographics Rendering Workshop 96, pp 165-174. Eurographics, 1996.

[Mill95] L. McMillan. *A list-priority rendering algorithm for redisplaying projected surfaces*. Technical Report UNC-95-005, University of North Carolina, 1995.

[MaHe00] S. Mahamud and M. Hebert, Iterative projective reconstruction from multiple views motion, in *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR '00), **2**, pp. 430-437, 2000.

[MaMB97] W. Mark, L. McMillan and G. Bishop. Post-rendering 3d warping. In *Proc Symposium on Interactive 3D Graphics*, pp. 7–16. ACM SIGGRAPH, 1997.

[MBR+00] W. Matusik, C. Buehler, R. Raskar, L. McMillan, and S. Gortler, Image-Based Visual Hulls, in *Proc. of SIGGRAPH'00*, 2000.

[MaBM01] W. Matusik, Chris Buehler, and Leonard McMillan, Polyhedral Visual Hulls for Real-Time Rendering, in *Proc. of the 12th Eurographics Workshop on Rendering*, 2001, pp. 115-125.

[Matk97] K. Matkovic, *Tone Mapping Techniques and Color Image Difference in Global Illumination*, PhD Thesis, Vienna University of Technology, 1997.

[NiBr95] W. Niem, H. Broszio, Mapping Texture From Multiple Camera Views Onto 3D-Object Models For Computer Animation, *in Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*, Santorini, Greece, 1995.

[OpGL02] OpenGL Architecture Review Board. ARB Vertex Program. OpenGL Vertex Program Documentation, 2002.

[OpGL02b] OpenGL Architecture Review Board. ARB Fragment Program. OpenGL Fragment Program Documentation, 2002.

[OvBi99] M. M. Oliveira and G. Bishop. Image-based objects. In *Proceedings Symposium on Interactive 3D Graphic*s, pp. 191–198. ACM SIGGRAPH, 1999.

[Paja98] R. Pajarola. Large Scale Terrain Visualization Using the Restricted Quadtree Triangulation. in *Proc. IEEE Visualization 98*, pp. 19-26, 515, 1998.

[PaSG03] R. Pajarola, M. Sainz, and P. Guidotti. Object-Space Blending and Splatting of Points. to appear in *SIGGRAPH Sketches and Applications*, 2003.

[PaYS02] R. Pajarola, Y. Meng and M. Sainz. Fast Depth-Image Meshing and Warping. in *Technical Report UCI-ECE-02-02*, The Henry Samueli School of Engineering, University of California Irvine, 2002.

[PaSaY03] R. Pajarola, M. Sainz and Y. Meng. Depth-Mesh Objects: Fast Depth-Image Meshing and Warping. *Technical Report UCI-ICS-03-02*, The School of Information & Computer Science, University of California Irvine, 2003.

[Perz97] M. A. Perzl. *Development of 3D models for the simulation of fluid dynamics and particle transport in realistic airway geometries*. PhD Thesis, Munich University of Technology, Faculty of Mathematics. 1997

[Ply] PLY format. http://www.cc.gatech.edu/projects/large_models/ply.html

[PZBG00] H. Pfister, M. Zwicker, J. van Baar, and M Gross, Surfels: Surface Elements as Rendering Primitives, in *Proc. SIGGRAPH'00*, pp. 335–342. Los Angeles, CA, July 2000.

[Phot] PhotoModeler, by Eos Systems Inc., http://www.photomodeler.com/

[PoKa93] C. J. Poelman and T. Kanade, *A paraperspective factorization method for shape and motion recovery*, Technical Report CMU-CS 93-219, School of Computer Science, Carnegie Mellon University , December 1993.

[Poll99] M. Pollefeys, *Self-calibration and metric 3D reconstruction from uncalibrated image sequences*, PhD. thesis, K.U.Leuven, 1999.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.

[RuLe00] S. Rusinkiewicz and M. Levoy. QSplat: A Multiresolution Point Rendering Sys-tem for Large Meshes, in *Proc. SIGGRAPH'00*, pp. 343-352. Los Angeles, CA, July 2000.

[RWP+95] H. Rushmeier, G. Ward, C. Piatko, P. Sanders, B. Rust, Comparing Real and Synthetic Images: Some Ideas About Metrics, in *Sixth Eurographics Workshop on Rendering*, Dublin, Ireland, pp. 82-91, 1995.

[SaBS02] M. Sainz, N. Bagherzadeh and A. Susin, Recovering 3D Metric Structure and Motion from Multiple Uncalibrated Cameras, in *Proc. of IEEE Conference on Information Technology: Coding and Computing*, pp. 268-273, 2002.

[SaBS02a] M. Sainz, N. Bagherzadeh and A. Susin, Carving 3D Models from Uncalibrated Views, in *5th IASTED International Conference Computer Graphics and Imaging* (CGIM 2002) August 12-14, Kauai, Hawaii, USA. pp 144-149, 2002.

[SaBS02b] M. Sainz, N. Bagherzadeh and A. Susin, Hardware Accelerated Voxel Carving, in the *1st Ibero-American Symposium in Computer Graphics* (SIACG 2002),pp 289-297, July 2-5, 2002 Guimarães, Portugal.

[SaSB03] M. Sainz, A. Susin and N. Bagherzadeh. Camera Calibration Of Long Image Sequences With The Presence Of Occlusions. in *Proc. IEEE*

International Conference on Image Processing. 2003

[Same84] H. Samet. The quadtree and related hierarchical data structures. In *Computing Surveys*, 16(2):187–260, June 1984.

[Same89] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, Massachusetts, 1989.

[SeDy96] S. M. Seitz and C. R. Dyer, View Morphing, in *Proc. SIGGRAPH 96*, pp. 21-30, 1996.

[SeDy99] S. M. Seitz and C. Dyer, Photorealistic scene reconstruction by voxel coloring, in *Proc. International Journal of Computer Vision*, 35(2), pp. 1067-1073, 1999.

[SGHS98] J. Shade, S. Gortler, L. Hee and R. Szeliski. Layered depth images, in *Proc. SIGGRAPH 98*, pp. 231–242, 1998.

[SiSa92] R. Sivan and H. Samet. Algorithms fro Constructing Quadtrees Surface Maps. In *Proc. 5th International Symposium on Spatial Data Handling*, pp 361-370, August 1992.

[SCMS01] G. Slabaugh, B. Culbertson, T. Malzbender and R. Schafer, A Survey of Methods for Volumetric Scene Reconstruction from Photographs, in *Proc. of International Workshop on Volume Graphics*, pp. 2001.

[SmBr97] S. Smith and J. Brady, SUSAN - a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45-78. 1997.

[ShTo94] J. Shi and C. Tomasi, Good Features to Track, in *Proc IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593-600, 1994

[Slam80] C. Slama, *Manual of Photogrammetry*, American Society of Photogrammetry, Falls Church, VA, USA, 4th edition, 1980.

[Snap] SnapeSnatcher, by Eyetronics, http://www.eyetronics.com/

[SeGE00] E. Steinbach, B. Girod, P. Eisert and A. Betz, 3-D reconstruction of Using Spatially Extended Voxels and Multi-Hypothesis Voxel Coloring, in *Proc. of International Conference on Pattern*

Recognition, Vol. 1, pp. 774-777, 2000.

[StCM02] M. Stevens, B. Culbertson, T. Malzbender. A Histogram-Based Color Consistency Test for Voxel Coloring in *Proc. of International Conference on Pattern Recognition*, Vol. 4, pp. 118-121, 2002.

[StTr96] P. Sturm and B. Triggs, A factorization based algorithm for multi-image projective structure and motion, in *Proc. of the 4th European Conference on Computer Vision*, pp. 709-720, Cambridge, UK, April 1996.

[Stur97] P. Sturm, Critical motion sequences for monocular self-calibration and uncalibrated Euclidean reconstruction, in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Puerto Rico, pp. 1100-1105, June 1997.

[Stur02] P. Sturm, Critical Motion Sequences for the Self-Calibration of Cameras and Stereo Systems with Variable Focal Length, in *Image and Vision Computing*, Vol. 20, No. 5-6, pp. 415-426, March 2002.

[Swai90] M. Swain, *Color Histograms for Object Indexing*. PhD. Thesis, University of Rochester. 1990.

[SzTo91] R. Szeliski and D. Tonnesen, *Surface modeling with oriented particle systems*. Technical Report 91/14, Digital Equipment Corporation, Cambridge Research Lab, December 1991

[Szel93] R. Szeliski, Rapid Octree Construction from image Sequences, in *Computer Vision, Graphics, and Image Processing: Image Understanding*, Vol. 58, pp 23-32, July 1993.

[ToKa92] C. Tomasi and T. Kanade, Shape and motion from image streams under orthography: A factorization approach, in *International Journal of Computer Vision*, 9(2):137-154, 1992.

[Trig96] B. Triggs, Factorization methods for projective structure and motion, *in Proc. IEEE CVPR96*, pp. 845-851, 1996.

[Trig97] B. Triggs, Autocalibration and the Absolute Quadric, in *Proc. IEEE CVPR97*, pp. 609-614, 1997.

[TMHF00] B. Triggs, P.F. McLauchlan, R.I. Hartley and A. Fitzgibbon. Bundle

adjustment - a modern synthesis. In *B. Triggs, A. Zisserman, and R. Szeliski (eds.), Vision Algorithms: Theory and Practice*, pp. 298--472. Springer-Verlag, 2000.

[TuLe94] G. Turk and M. Levoy. Zippered Polygon Meshes from Range Images. in *Proc. SIGGRAPH'94*, 28(3):311-318, July 1994

[WAA+00] D. Wood, D. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin and W. Stuetzle, Surface Light Fields for 3D Photography, in *Proc. SIGGRAPH'00*, pp. 287-386, 2000.

[ZDSS01] L. Zhang, G. Dugas-Phocion, J.S. Samson and S. Seitz, Single View Modeling of Free-Form Scenes, in *Proc. Computer Vision and Pattern Recognition*, 2001.

[ZaSe01] L. Zhang, S. Seitz, Image-Based Multiresolution Shape Recovery by Surface Deformation, *in Proceedings of SPIE: Videometrics and Optical Methods for 3D Shape Measurement,* January, 2001

[ZPBG01] Zwicker, M., Pfister, H., van Baar, J. and Gross, M., Surface Splatting, in *Proc. SIGGRAPH'01*, pp. 371-378, August 2001