

Surface Estimation From Multiple Images

Mads Kessel

Kongens Lyngby 2006

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

This thesis presents an overview of the general problem of surface estimation from 2D images. It describes the currently known approaches, and states the pros and cons of them.

An algorithm proposed by Vogiatzis, using deformable models in a Bayes framework for estimating the surface from a set of images with known intrinsic and extrinsic parameters, is thoroughly described and reflected upon. The algorithm is implemented in C++ using OpenGL and the OpenGL Shading Language to help performance. A number of deviations to the work of Vogiatzis is proposed and implemented as well.

The implementation is thoroughly tested in both a synthetic environment, where the ground truth is known, and using several real world datasets. The result is used to conclude on the algorithm by Vogiatzis, and the proposed deviations to it.

Several conceptually new proposals are described, some of which has been implemented and evaluated.

keywords: Surface estimation, 3D Reconstruction, Structure from Motion, Deformable Models, Simulated Annealing, Bayes analysis, OpenGL, GPGPU, Image Metric, Surface constraints, Mesh Simplification.

Resumé

Denne afhandling præsenterer en gennemgang af det generale problem - overflade estimering fra 2D billeder. Den beskriver de nuværende kendte fremgangsmåder og beskriver deres fordele og ulemper.

En algoritme foreslået af G. Vogiatis, hvori deformerbare modeller er brugt i et Bayesiansk regi for at estimere overfladen fra et set billeder med kendte interne og eksterne parameter, er grundigt beskrevet og reflekteret på. Algoritmen er implementeret i C++ med brug af OpenGL og OpenGL Shading Language, for at forøge ydelsen. Et antal ændringer fra Vogiatis's arbejde er foreslået og implementeret ligeså.

Denne implementering er grundigt gennemtestet i både et syntetisk miljø, hvor den sande model er kendt, og ved brug af flere dataset fra den virkelige verden. Resultatet er brugt til at konkludere på Vogiatis's algoritme, og de foreslåede ændringer deraf.

Flere konceptuelt nye forslag er beskrevet, hvoraf nogle er implementeret og evalueret.

Nøgleord: Overflade estimering, 3D rekonstruktion, Strukturer fra bevægelse, Deformerbare modeller, Simuleret nedkøling, Baysiansk analyse, OpenGL, GPGPU, Billed sammenlignings mål, Overflade restriktioner, Mesh simplificering.

Preface

This thesis was prepared at the Institute of Mathematical Modelling at the Danish Technical University from September 2005 to April 2006, as a partial fulfillment of the requirements for acquiring the degree of Master of Science in Engineering, M.Sc.Eng.

The thesis deals with the general problem of surface estimation from a set of input images. A known algorithm using a deformable model in a Bayes framework is discussed in detail, together with a set of improvements.

It is assumed that the reader has a basic knowledge in the areas of statistics, image analysis and computer graphics.

Kgs. Lyngby, April 28, 2006

Mads Kessel - s001840

[mads@kessel.dk]

Acknowledgements

Writing a thesis like this requires something of the writers surroundings. First of all, my supervisor Henrik Aanæs has been of great help, always available, when one of my crazy ideas should be discussed with a competent person. His direct approach and friendly attitude make you feel like 'one of the guys'.

Second I would like to thank Andreas Bærentzen, for helping me with some of the more challenging problems in OpenGL and graphics in general. Even all the documentation of the world can't stand up to an experienced human being. Thirdly I would like to thank our secretary Eina, always talking and smiling, giving you a good start on a tough day.

And when the time comes to my office mates - even though their mere presence can sometimes drive you crazy, it just would not be the same without. Thanks for the fun, the talks and the desperation late at night. Thanks to Café308, and everybody therefrom for endless ours of wasted time playing whist. It is nice to be able to forget it all and empty your head from time to time.

And finally, a special thanks to the one that has coped with me throughout the last few months, desperately trying to help a man that does not like to be helped.

Mus...

Contents

1	Introduction	1
1.1	Motivation and Objectives	2
1.2	Thesis Overview	3
I	Surface Estimation in General	5
2	The History of Surface Estimation	7
3	Surface Estimation Basics	11
3.1	Definition of the Problem	11
3.2	Classification of Methods	12
3.3	Common Assumptions	13
4	Stereo Vision	15
4.1	Epipolar Geometry	16

4.2	Feature Matching	17
4.2.1	Mosaicing	18
4.2.2	Delaunay Triangulation	18
4.3	Dense Stereo Matching	19
4.3.1	Depth Resolution	20
5	Multiple View Vision	21
5.1	Dense stereo matching and stitching	21
5.2	Multiple View Geometry	22
5.2.1	The Factorization Method	22
5.2.2	Using Bundle Adjustment	23
5.2.3	Using Lines and Curves	24
5.3	Space Carving	25
5.4	Level Set Method Using PDE	25
5.5	Using Deformable Models	26
5.6	Frontier Points	27
5.7	Minimization Using Graph Cuts	28
6	Discussion	29
II	Basic Algorithm	31
7	The Algorithm	33
7.1	Foundation	34

7.2	Formulation	35
7.2.1	Algorithm	36
7.2.2	Mathematical Formulation of the Problem	37
7.3	Image Similarity Measure	39
7.3.1	The Occlusion Problem	40
8	The Bayes Formulation	43
8.1	Bayes Theorem	43
8.1.1	Algorithm In a Bayes Framework	45
8.2	Using Simulated Annealing	46
8.2.1	Numerical Aspects of μ	48
8.2.2	The Annealing Schedule	48
8.2.3	Simulated Annealing in a Bayes Framework	50
9	A Deformable Mesh	51
9.1	Mesh Representation	52
9.1.1	Mesh Operations	52
9.1.2	Mesh enforcements	52
9.2	Defining a Mesh Smoothness Term	53
9.3	Deforming a Mesh	54
9.3.1	Random Deforms	56
9.3.2	Gradient Deforms	57
9.3.3	Correlation Deforms	57

9.3.4	Edge Swap	58
9.3.5	Edge Collapse	58
9.3.6	Vertex Split	60
9.4	Rollback of Deformations	62
9.5	Topological Considerations	63
10	Implementation	65
10.1	Program Structure	66
10.2	Interfacing OpenGL	69
10.2.1	Used Extensions	70
10.2.2	Projecting from a Camera	70
10.2.3	Rendering the Mesh	74
10.2.4	Evaluating a View	75
10.3	Pitfalls	80
10.3.1	A Good Pseudo Random Number Generator	80
10.3.2	Projective Errors	80
10.3.3	Depth Errors	81
10.4	Discussion	82
10.4.1	Limitations	82
10.4.2	Memory Considerations	83
10.4.3	Choosing Constants	84
11	Datasets	87

11.1 Synthetic Datasets	88
11.1.1 The Box	88
11.1.2 The Bunny	88
11.2 The Cactus	89
11.3 The Gargoyle	90
11.4 The Pot	90
11.5 The Face	91
11.6 Masters Lodge	92
12 Results	97
12.1 Synthetic results	98
12.1.1 The Box	98
12.1.2 The Bunny	101
12.2 Architectural results	106
12.2.1 The Pot	106
12.2.2 The Masters Lodge	109
12.3 Difficult objects	111
12.3.1 The Gargoyle	111
12.3.2 The Cactus	113
12.3.3 The Face	114
12.4 Summary of Test	116
13 Discussion	117

13.1 Identified Flaws and Issues	117
13.2 Conclusion	118
III Improvements	119
14 Discussion of Possible Improvements	121
14.1 Coarse to Fine Technique	122
14.2 Improving Convergence	122
14.2.1 Error Based Selection	123
15 The Zooming Approach	125
15.1 Introduction	125
15.2 The Zooming Algorithm	127
15.3 Results	128
15.4 Conclusion	130
16 Use of Feature Maps	131
16.1 Introduction	131
16.2 Implementation	132
16.3 Results	133
16.4 Conclusion	133
17 Error Based Selection	135
17.1 Using Errors	135

17.2 Setup Using Queues	137
17.3 Discussion	137
IV Discussion	139
18 Future Work	141
18.1 General improvements	141
18.2 Model Texture Creation	142
18.3 Using Frontier Points	142
18.4 Higher Order Topologies	142
18.5 Study of the Textures Impact on the Algorithm	143
19 Discussion	145
19.1 Main Contributions	145
19.1.1 A Survey of the Field of Surface Estimation	145
19.1.2 Preparation of Datasets	145
19.1.3 A GPGPU Implementation	146
19.1.4 Presentation and Implementation of Improvements	146
19.2 Conclusion	146
References	148
List of Figures	153
List of Tables	160

V	Appendix	163
A	User guide	165
A.1	Installation	165
A.2	capture.exe	165
A.3	main.exe	166
A.3.1	Job description file	166
B	CD - Thesis Data	169

Introduction

Since the childhood of computers, it has always been a dream to make computers 'see', a task so simple and obvious to the human brain, that we normally neglect it. From the early science fiction book R.U.R by Karel Capek (1920)[15], that first introduced the term robot, to the newest filmatization of Isac Assimovs 'I, Robot' [9], the task of 'seeing' and acting upon the seen has been regarded as something 'that will come' in the not so distant future. However as the scientific fields of image analysis and artificial intelligence has developed, it has become more and more clear that there is no simple solution to the problem of understanding the seen.

One of the more straightforward solutions proposed, is to setup a neural network for processing the input data, to emulate the human approach, i.e. artificial intelligence. The current neural networks however has only just reached the point where they can emulate the behavior of the simple worm *C.elegans* [46], thus there is a great deal of work to do, before one can emulate the billion of neurons working to understand the human visual system in contrary to the 302 neurons in *C. elegans*, [36].

Instead of letting a neural network process the visual input, one can try to induce some information in a more mathematical framework. This, also denoted Image analysis, has largely been a success, as it has been developed to a state where objects, lines, shapes and much more can be identified in images. But for a robot

to navigate in a complex world as predicted of the future, a 3D understanding of its surroundings is unavoidable.

Even though humans base most of their perception of the surroundings on their sight, many different approaches has been taken, both technologically and biologically. Fx bats use an advanced bio-sonar to navigate in complete darkness, while ship and plane navigation just would not be the same without the RADAR first developed in 1904 by Christian Hülsmeyer. Another high precision approach is the well known laser scanner, creating a cloud of 3D points by detecting the position of an intersection between an object and a laser beam. Common for all these approaches are that they are actively interfering with their subject, by emitting a media. This is in contrary to using visual perception, where the self reflected or emitted light of objects are measured. This can limit their use since it is not always feasible to interfere, and it requires the calibration of both sending and sensing equipment together. Further more this equipment tends to be more expensive than the common off the shelf digital cameras needed to percept the world visually. Thus a good surface estimation method using digital images as input, would surely find its place in the modern world.

Also in this research field, many approaches exists, however as the task is very complicated and computational expensive, most approaches only functions under special conditions, and generates 3D information of a various quality. It is thus mostly limited to the research laboratories around the world, however as we shall see, it is in rapid development and it is thus only a question of time before off the shelf digital cameras can be used by common people to bring 3D digital perception into the every day life.

1.1 Motivation and Objectives

Besides childish desires to build robots, there is also a good deal of arguments for investing in the development of surface estimation, arguments of a more economical nature. The future market of robots that can operate together with humans, in the same environment as humans, is obviously huge. However already today estimating the surface of objects, can be used for plenty of things. In the following a couple of the possibilities I see in surface estimation are described.

Imagine an online auction showing a 3D model of the furniture they are selling. One could grab the 3D model and add it to a virtual 3D model of ones living room, already produced by sending a bunch of digital photos to a server

performing surface estimation. For architectural purposes, when an old building from the times before computers are to be modernized. Surface estimation could aid the development of CAD schematics. Or when walking into a museum, searching for vases from 2000 B.C. to 1000 B.C. of Greek origin, on a computer. Every object in the archive has been catalogued and a 3D model has been reconstructed. From this a virtual museum is constructed from the search criteria, ready to be explored by the guest. It is fact that 90% of the objects of the Danish national heritage is stored away in the Danish National museum, and not available to the public, simply because of lack of space. The possibilities are endless and only constrained by your imagination.

However before all this is possible, a robust surface estimation setup is needed. This thesis is a slow walk into the world of surface estimation using digital images, and (hopefully) a contribution of improvements to one of the most robust algorithm existing today. For simplicity the rest of this thesis will use the term *surface estimation* in stead of the longer *surface estimation using digital images*, as it will concentrate on the impact digital images has on this research field.

The objectives of this thesis are:

- To summarize the field of surface estimation, and collect useful mathematical concepts and methods available today.
- To implement a surface estimation algorithm proposed by Vogiatzis using graphics hardware for GPGPU¹.
- To test the quality and convergence of the implementation under various conditions.
- To identify, implement, test and discuss some improvements of the basic algorithm.

1.2 Thesis Overview

This thesis is composed of 4 parts, that naturally leads the reader through the documentation of the work done. The contents of the 4 parts are as described below.

¹General Purpose computing using Graphics Processing Units.

Part I, Surface Estimation in general. Presents a survey of the field of Surface Estimation. Known methods are described and discussed, together with the necessary framework. Basic mathematical concepts are presented.

Part II, Basic Algorithm Describes and analysis an algorithm for surface estimation presented by Vogiatzis in detail. The algorithm is reflected upon, and lesser deviations to it is proposed. Documents an implementation of this algorithm, and discusses uncertainties in Vogiatzis's description of the algorithm. Tests the implementation under various conditions, both synthetical and real, and discusses the results obtained.

Part III, Improvements. Identifies parts of the basic algorithm, that can be improved. Some proposed improvements are discussed in further detail, and an implementation is documented. The proposed improvements are tested, discussed and evaluated.

Part IV, Discussion Discusses ideas for future work, and evaluates and concludes on the work done in this thesis.

Part I

Surface Estimation in General

CHAPTER 2

The History of Surface Estimation

As the name suggests *surface estimation* is the problem of estimating a surface, however, as this has various uses, representations and formulations, it also has many synonyms that all means surface estimation in slightly different interpretations. A short list of some of the more well known synonyms are: surface reconstruction, stereo vision, multiple view vision, shape from ... (fx shading), reverse rendering, computer vision and machine vision. In this thesis the term, surface estimation, will be used as it covers the underlying meaning in these terms. This chapter briefly surveys the history of the field of surface estimation, and introduces the reader to some of the more well known approaches and terms.

Stereopsis, the process in visual perception leading to perception of depth, were first discovered by Charles Wheatstone in 1833. He recognized that because each eye views the world from slightly different horizontal locations, the position of the objects in the view also differs a small amount depending on the distance from the eyes to the object. Until the 60's when computer resources became more widely available, stereopsis was mainly used for the well known binocular views, where the eyes independently are shown two different images of the same scene taken with a small horizontal disparity corresponding to the distance between human eyes. An old example of such a card is shown in figure 2.1.

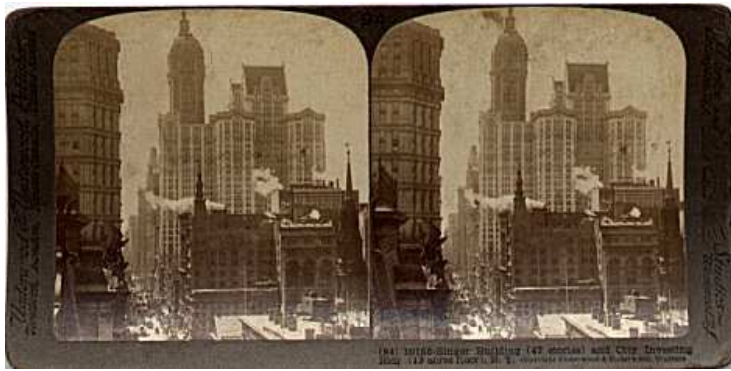


Figure 2.1: A so-called binocular view of a Manhattan scene from approximately 1910. The image is taken from [4].

In the 60's digital image analysis was born, in which computers were used to extract meaningful and useful information from digital images. One very interesting subfield of digital image analysis is to make computers doing stereopsis, ie. make computers see in depth. The first attempts were brute force searches along scan lines¹ to identify the disparity of pixels. Some of the first are [31, 58], that uses correlation of windows around the pixels to be matched to find the best fit. Since then a myriad of different methods based on this basic idea of *stereo vision* has been developed, all trying to improve the matching reliability, or to improve the performance. A good survey of these can be found in [51], where the methods are described and evaluated.

A different approach was taken by B.K.P. Horn in 1970 in his PhD thesis, where he proposed using shades of the objects in the scene to recover their shape [35]. This method, opposed to the other only requires one image, however it is not as robust. Many of these *shape from shading* methods are described in the survey, [49], by R. Zhang *et al.* A fusion of methods using shades and methods using stereo was proposed by P. Fua and Y. Leclerc in 1993 in [26], where shading is used in uniform textured areas of the images, where normal stereo would fail. This method was one of the first to use a deformable model of the surface, which was optimized using the image data in stereo as well as the shading.

It was first in the 90's that computational resources became large enough to accommodate the needs for using more than two images. In the late 90's a couple of completely new approaches saw the light, all using a more general view of the problem of surface estimation. One of the first of these is a method

¹The horizontal lines of digital images.

by Faugeras and Keriven in [24], where a global energy function are minimized using so-called *level sets* for representing the surface. Not much later Kutulakos and Seitz proposed a method where the space is divided into boxes called voxels, that are removed one by one using a similarity measure of the projection of the voxel into the images and a measure of local neighborhood of voxels [39].

In the 90s when the gaming industry had been established, graphics hardware evolved rapidly. Thus it became possible to make fast projections of images unto a 3D model and back into other images. This new resource was exploited by G. Turk and P. Lindstrom to simplify 3D models in [56], and to perform image driven triangulation in [45] by D. Morris and T. Kanade. These methods are not surface estimation methods in themselves, however the idea to use image data and 3D models together lead to such a method in 2001 by Li Zhang [60]. In 2003 Vogiatzis proposed to setup the problem in a Bayesian framework using simulated annealing, [57], and let it control the deformations of the model.

The field of surface estimation is by far exhausted, as one can see by the number of current articles on the subject. There is therefore a good chance that we, with the combined efforts of the research and the ever increasing computational resources, will see methods that will break the laboratory barrier and be a part of every day life.

Surface Estimation Basics

As described in the last chapter, surface estimation is a very broad research field. This chapter will try to identify some of the common basic properties and define common terms and concepts.

3.1 Definition of the Problem

The task of performing surface estimation in general can be formulated as:

From a set of *input images* ($I_1 \dots I_n$) of a *scene*, to estimate the *surface* of the scene.

The following will clarify what is meant by *input images*, *scene* and *surface*

input images. An image in general can be defined as a discretization of the intensity of the light passing through a plane in space. Mostly the light passes through a lens beforehand, and the measuring of the light is arranged in a rectangular matrix, as we will assume here. The intensity of the light can be measured in different wavelength giving rise to color images.

scene. A scene is a limited world, synthetic or real, from which images can be captured. Associated with the capturing of an image is a *camera* defining how the image are captured. In most of the literature, the pinhole camera model is used, describing a camera using 5 intrinsic parameters for its *calibration* and 7 extrinsic parameters describing the *view* or *pose*[18].

surface. The surface of a scene is the orientable 2-manifolds defined by the boundary of the solid objects in it. As we shall see, the surface can be represented in different ways. These ways are for most surfaces only approximate, since a surface is stepwise continuous.

In other words the problem of surface estimation is the problem of extracting 3 dimensional information of a scene of which a number of images have been taken.

3.2 Classification of Methods

In general doing surface estimation can be divided into two steps, that may be done iteratively. The first is to estimate the cameras capturing the input images, that is the internal calibration and the viewpoint and pose of the cameras. Many methods assumes that these parameters are known, which is a fair assumption since there exists good semi or fully automatic methods for estimating these parameters, see [59].

The second step is to estimate the surface using the cameras and the input images. This is the more difficult step where surface estimation methods differ the most. One good and traditional classification of the different methods, is to divide them into 3 main groups depending on the number of input images. This gives the following classification:

Mono vision using knowledge of light distribution (shape from shading) and empiric knowledge of objects, textures etc. Thus a single input image I is used to estimate the surface. These methods generally produce low quality results, as they are build on empiric knowledge of the scene, and the light distribution in it, and therefore lack actual stereo information.

Stereo vision emulating the human stereopsis by estimating disparities between objects, features or pixels in two input images, commonly denoted by the left and right input image, I_l and I_r . These methods are fast since only two images are compared, however they can obviously not estimate

the full surface of scenes from the two input images. Most of these methods are based on the two cameras aligned as the human eyes or the images being rectified beforehand to emulate this.

Multiple view vision taking a more general approach to the problem, by using the mutual information given in a set of n input images, $(I_1 \dots I_n)$. In many cases n can be 2, and thus being used for stereo vision, however most of these methods does not restrict the cameras to be aligned in a specific way.

Unless extra information is given or empiric knowledge of the objects in the scene are used, calibration from the input images are only given up to scale, which is called an *euclidian reconstruction*. Other methods neglects the internal calibration of the cameras which results in an *projective reconstruction*, since the intrinsic parameters define the projective nature of the camera. Similar images captured in an orthogonal view can be used for an *orthogonal reconstruction*. The reconstructions can however easily be transform into a *metric reconstruction* as a post processing of the result, if the information lacking for the surface estimation method is obtained in other ways, fx using physical measurements.

The next two chapters will briefly describe some of the more important concepts and methods in the last two of these groups. The group only using a single image is not discussed further in this thesis, however a good example can be found in [35].

3.3 Common Assumptions

As it is with other research fields, most methods solving the surface estimation problem assumes that the world is a rather pleasant place to be. As one might have notices, the definition of surface estimation used does not require any temporal constraints of the capturing of the images. Thus the objects in the scene may not be in the same position in the input images, and may even have different shapes as it is deforming. A good example of this is the individual frames of a movie showing a human face talking. Most, but not all methods, assumes that the scene is static or rigid. A relaxation of this assumption is to assume that the images are taken in a known order rapidly following each other such that the intermediate movement and deformation is small, and can be measured together with the surface. These methods are aimed for the images captured by video cameras, as they have this behavior, see fx [17, 47]. This of course adds a great deal of complexity to the method.

Another common assumption is that the surface material reflects light in a complete diffuse, also called the Lambertian assumption. The problem arising if this is not the case, is that images may not perceive the same color information for the same point in space, which may lead to difficulties when searching for correspondences in the images. This mostly occurs when the objects have specular (shiny) surfaces producing specularities in the images. The problem can be avoided using a completely diffuse light source, which makes the assumption viable, but at the same time constraints the use of the method.

Even if the scene is assumed to be static and Lambertian, another lighting problem can occur. This is due to different lighting conditions when and where the images are captured, which can cause a modern camera to auto-adjust for lighting. The lighting conditions can be described as an affine transformation in color space [16]. This can be accounted for by normalizing them to the same mean and deviation, which some methods assumes. This can easily be done beforehand, however it removes some of the information in the images.

The last and seldom mentioned assumption described here is that the objects in the scene are opaque, and that the air medium is completely transparent. It would impose great complications, to model fog properties and reflections in semi transparent materials, which makes the assumption understandable.

CHAPTER 4

Stereo Vision

Even though it has been shown that computers can deduct some depth information from a single image [35], and that humans to some extent can use empiric knowledge of the world to estimate the depth to objects using only a single view [11], it is necessary to have at least two views of a scene to obtain a robust surface estimation. This chapter will describe this group of methods, in which the methods are based on two and only two images.

Two different approaches to do stereo vision is, dense stereo matching and sparse feature matching. The first estimates the depth of every pixel in one of the input images, and is thus dense. The other identifies strong features in both images, finds likely matches, and calculates the depth of these points. Common for the two approaches is that they find corresponding points in the two images and make use of the geometry of the two cameras to calculate the depth of the point, usually denoted the camera geometry or the epipolar geometry of the setup. In the following this geometric dependency will be explained, before the two groups of stereo vision are described.

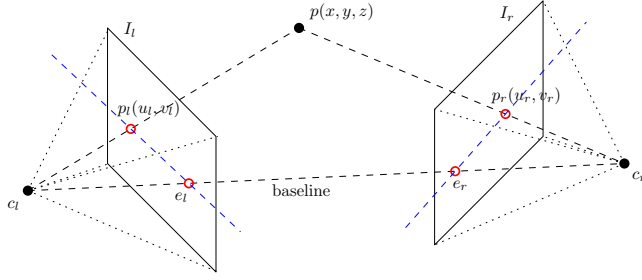


Figure 4.1: The epipolar geometry in a stereo setup

4.1 Epipolar Geometry

From figure 4.1, it can be seen that there is a clear geometrical relation between a point, p , in space and the projected points, p_l and p_r , in the two images, I_l and I_r . The two images center of projection, c_l and c_r , together with p forms a plane denoted the *epipolar plane*. The interesting observation is that if only p_l is known, then p must be on the line connecting c_l and p_l and therefore p_r must be on the intersection between the epipolar plane and the image plane of I_r . This line, the *epipolar line* (the blue line), is formed by the projection of c_l in I_r , the *epipole* c_r , and the projection of an arbitrary point along the line from c_l and p_l . This linear relationship between the points in the images can be written

$$p_l^t F p_r = 0 \quad (4.1)$$

, where the matrix F is called the *fundamental matrix*, and is a 3×3 matrix. To describe the relation using F , p_l and p_r must be written as projective coordinates or *homogeneous coordinates*, adding an extra projective dimension. As F is 3×3 it has 9 parameters, however it only has 7 degrees of freedom, as it is constrained to have rank 2, that is $\det(F) = 0$ and only up to scale. The fundamental matrix is independent to the contents of the scene, however it can be estimated using corresponding image coordinates of the same points in space. One elegant method is to use the 8 point algorithm as described in [33].

The fundamental geometry only describes the relationship between the image planes of the two cameras, and thus does not include the internal calibration of the cameras. This can be seen from the number of freedom degrees in F , as it corresponds to the 7 external parameters describing the view and pose of a

camera. Thus if nothing else is known, the left camera can be placed at origo of a world coordinate system, which then defines the 7 external parameters of the right camera using F . The *essential matrix*, however defines both the external and internal relationship between the cameras. As a transformation between the image plane, and the actual pixels can be described using a so-called calibration matrix K , the essential matrix E is defined as:

$$E = K_l^{-1} F K_r \quad (4.2)$$

4.2 Feature Matching

Using the epipolar geometry of a stereo setup, one can calculate points in space from corresponding points in images. Thus if one can identify such pairs in the images one can obtain a point cloud of the scene. In general this can be done by finding a set of features in the images and matching them, hence the term *feature matching*. A feature is something that can easily be identified in the images, for example because it has strong gradients or because it has a unique texture. One good and robust feature detector is the Harris corner detector [32], that relies on the greatest local gradient. An example of using it can be seen in figure 4.2.

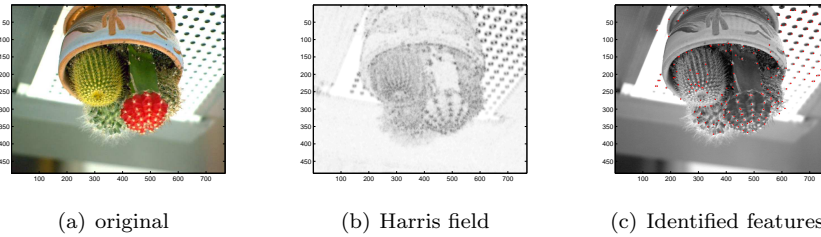


Figure 4.2: The Harris corner detector used on a cactus being upside down for reasons being clarified in Chapter 11.

When features in the two input images are collected, they are matched to find the pairs. If the epipolar geometry is known, only the matches that lie on or close to the epipolar line need to be searched, otherwise a brute force search is needed. Evaluating a match is usually done by comparing a so-called feature descriptor. This descriptor can either be a controlled sampling in the area of the image around the feature, or some means of describing it. In large images the number of features can be very high. It is important to limit the set of

features, as every feature must be matched to every other feature, which can be computationally expensive and error prone. Therefore a lot of work has been put into making the matching as reliable and fast as possible. An example of the inventiveness of the field is a method by M. Brown *et al* [14] using Gaussian image pyramids to sample a gradient oriented normalized window from which 3 wavelet coordinates are extracted and compared to other descriptors using the summed square error. To further avoid errors only the matches of which the best match is substantially larger than the second best are trusted.

When matching point pairs have been obtained the point cloud can be calculated directly using the epipolar geometry. If this is not available then it can be estimated first using the 8 point algorithm. A single wrong match, however, could ruin such an estimation and render the whole result useless. This can be accounted for using the RANSAC¹ algorithm, see [25], iteratively estimating the geometry using only a subset of the point pairs. After a number of steps the geometry best capturing the point pairs are used to divide the pairs into good pairs and outliers (wrong matches). The inliers can then be used to estimate the geometry more accurately. One robust method using a variant of this method can be found in [59].

4.2.1 Mosaicing

This framework, for finding features and matching them to find corresponding pairs, are not only used for surface estimation. The task of stitching overlapping images together, also called *mosaicing*, uses the same approach. Instead of matching for the best 3D point, the best match, when projecting the images onto some common surface is used. When projecting all images this surface thus becomes the blended macro image. Stereo vision, and surface estimation in general, can be seen as a generalized mosaicing technique, where not only the matching is searched for, but also the surface to project the images on. More information on mosaicing can be found in [16].

4.2.2 Delaunay Triangulation

The result of a feature matching is a point cloud of the surface of the scene. A point cloud, however, does not represent the surface of the scene, as it does not define the connectivity between the points. It is ambiguous, since every point in the cloud can be thought of as a single floating object in space, and not a part of

¹RANdom SAmple Consensus.

a single or a few objects. To obtain a real representation one can triangulate the point cloud, for example using Delaunay triangulation. This may not result in the correct connectivity, however Morris and Kanade proposed that the initial triangulation could be refined by iteratively evaluating the reprojection of the images onto the mesh, while the connectivity is changed [45]. Thus both the image data, and the point cloud would be used in the surface estimation giving a photorealistic result.

4.3 Dense Stereo Matching

As opposed to feature matching, the dense stereo matching methods do not attempt to find the best points to match. Instead every pixel is matched to all pixels along the corresponding epipolar line in the other image. Therefore dense stereo methods require that the cameras are calibrated beforehand. To increase performance it is normally required that the images are aligned so that the epipolar lines are simply scan lines over the input images. If not, then one or both of the images can be transformed to obtain this, which is called a *rectification* of the image.

The result of the match is a *disparity map* describing the disparity of each pixel and the best match found in the other image. The disparity map can either be in pixel precision or in subpixel precision, if the matching method supports it. Using the epipolar geometry the disparity map can be transformed to a *depth field*, which for every pixel contains the estimated depth to the object. The depth field representation of the surface is normally only regarded as $2\frac{1}{2}D$, as they can not describe anything behind the surface covering the image. It can however be transformed to form a triangular mesh, which can be post processed using techniques like in feature matching or a mesh simplification technique like [56].

There exists a wide range of methods for doing the actual matching. Some methods take the connectivity into consideration such that two neighboring pixels have disparities relatively close to each other, while others smooth the disparity map, to remove outliers. Most methods use some means of a window that is folded over the corresponding scan lines. A good survey and comparison of the methods available can be found in [51].

4.3.1 Depth Resolution

When designing stereo setups for doing dense stereo matching or stereo vision in general, some consideration must be put into the desired depth resolution. It is obvious that the depth resolution increases together with the image resolutions as it allows a more precise measurements. A good rule to obtain satisfactory results is that the ratio $\frac{\text{baseline}}{\text{depth}}$ stays within $[\frac{1}{3} \dots 3]$, [19].

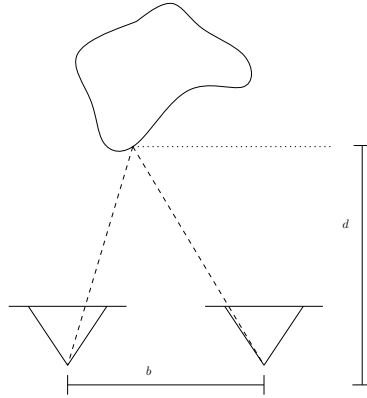


Figure 4.3: Illustration of the baseline and depth in a stereo setup. setup.

The baseline depth ratio is illustrated in figure 4.3, which gives an intuitive understanding of this rule. If the object is too far away then the differences in the two images are too small to estimate details in the object. On the other hand, if the object is too close, then it either falls out of the field of view of the cameras, or too much occlusion occurs. Further more materials tends to reflect the same amount of light in viewing angles closer related, thus making matching easier. It is thus a weighing between the achievable depth quality, and matching reliability.

Multiple View Vision

As the last chapter clarified, stereo vision can be used to estimate the depth from the two cameras to the object in the scene. From this a 3D shell of the object viewed from the cameras can be produced. It can not however in itself be used to reconstruct a whole object as viewed from every angle. Further more the depth resolution for a single stereo rig tends to be small. The direct approach to accommodate for this is of course to increase the amount and distribution of input data, which in this case means adding more input images.

Using more than two input images gives rise to a new problem that can be approached in a number of different ways. This chapter describes some of the more popular and interesting methods, that have been proposed to solve the multiple image surface estimation problem.

5.1 Dense stereo matching and stitching

While stereo vision can be used to estimate the distance from the cameras to the surface, it can not in itself reconstruct the full scene. The *shells* of the surface produced by the stereo vision, can however be stitched together to recover the full surface. Thus a simple multiple view vision method is to use stereo vision

on pairs of the input images, and stitch the result together, see fx [44]. This method however does not use the global mutual information given in more than two images. Where the shells overlap some averaging can be done, however it is not as strong as a real multiple view surface estimation method taking all views into account at the same time.

Using dense stereo matching for multiple view vision either requires that the cameras are calibrated beforehand, or that the cameras are naturally paired with a known mutual calibration. The later can use the best fit when stitching the shells together to calculate the outer calibration of the cameras.

5.2 Multiple View Geometry

As with the two camera case some geometrical considerations can be used to calculate the 3D structure from known matches in the images. The fundamental matrix used to describe the relation between two views is sometimes known as the *bifocal tensor*. The idea can be extended to three and four views to obtain the *trifocal tensor* and the *quadrifocal tensor*. Having m views can be described using a *m-focal tensor*, however as we shall see it makes less sense describing the camera geometry this way, when the number of views goes up.

As the two view geometry showed, the image coordinates of the projections of a point in space M , is all that is needed to calculate it. In the case of three cameras and three projections of M , more information is added, and thus one does not need all information from all cameras to calculate M . It can be shown ([27]) that only 4 parameters are needed, for the two view case it is the 4 coordinates of the projections, for the three view case it is two coordinates of one of the views and one coordinate in each of the other views. Thus using more than 4 cameras makes the problem over constrained, which would result in not all cameras being used directly. It is of course waste of information only using 4 parameters when calculating M . The next sections describe different approaches to make use of the extra information.

5.2.1 The Factorization Method

If the image coordinates are subject to noise, the extra information given using multiple views can be used to estimate M with more accuracy. This elegant method called *factorization*, is somehow an extension to the idea of the 8 point algorithm, that estimates the multiple view equivalent to fundamental matrices.

It was first described by Tomasi and Kanade in [54], for orthogonal views of the scene, but later rewritten by P. Sturm and B. Triggs in [53], to the projective case as described here.

The idea is that having m cameras defined by their unknown camera matrices $P_1 \dots P_m$ and the image coordinates of n 3D points $M_1 \dots M_n$ one can describe the projection of each point into each camera by

$$\zeta_i^j m_i^j = P_i M^j, i = 1 \dots m, j = 1 \dots n \quad (5.1)$$

, where ζ_i^j is the projective depth of point j in camera i . These $n \times m$ equations can be stacked into matrix form giving

$$\underbrace{\begin{bmatrix} \zeta_1^1 m_1^1 & \zeta_1^2 m_1^2 & \dots & \zeta_1^n m_1^n \\ \zeta_2^1 m_2^1 & \zeta_2^2 m_2^2 & \dots & \zeta_2^n m_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \zeta_m^1 m_m^1 & \zeta_m^2 m_m^2 & \dots & \zeta_m^n m_m^n \end{bmatrix}}_W = \underbrace{\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_m \end{bmatrix}}_P \underbrace{[M^1, M^2, \dots, M^n]}_S \quad (5.2)$$

In this form only the measured image points in W are known. If however we assume that the ζ 's are known, then W is known and can be decomposed into UDV^T using singular value decomposition. Using the arguments of [53], these can be used to calculate P and S . If on the other hand P and S are known, then the ζ 's can be calculated. The factorization method is thus to initialize the ζ 's to one, and iteratively calculate P , S and the ζ 's until convergence.

The result using the factorization method is good, however there is some drawbacks. As with the two view case, the result is a point cloud of non related points and must be triangulated in some way before a real 3D model is obtained. Further more, the feature matching is error prone, and occlusion in the images makes some of the information unreliable. The problem can be addressed to make a more robust factorization, an example of such research can be found in [8].

5.2.2 Using Bundle Adjustment

A general approach to use all information is the so-called *bundle adjustment* first discussed by D. Brown in 1976, [14]. The name of this method comes from

the underlying idea which involves adjusting the *bundle* of rays between each camera center and the set of 3D points. It is thus a method for calculating both the 3D points and the geometry between the cameras at the same time. This problem is approached using a minimization scheme to minimize the squared geometric error between all image points and the projected world points. Thus if reusing the notation in the last section, the goal is to minimize

$$\min_{P_i, M^j} \sum_i^m \sum_j^n \|m_i^j - P_i M^j\|^2. \quad (5.3)$$

Minimizing such a problem can be a huge task, even using a relatively few points and cameras. For example, having 10 cameras and 100 world points corresponds to a minimization problem solving for above 300 variables concurrently. The nature of the problem, however, allows for a much faster minimization. Each residual in the problem, ie. $\|m_i^j - P_i M^j\|^2$, only depends on a single point and a single view. This can be exploited, to reduce the computational efforts used to minimize the problem considerably. The actual minimization can be done using any minimization technique, however to avoid getting stuck in a minima the Levenberg-Marquardt minimization can be used. Even though the problem has been reduced, it still requires an initial guess, which makes it a suitable choice for polishing the solution of other faster methods. The image points used in this problem should be of high quality, or at least be uniformly distributed around the correct image points.

One of the advantages of using this method, is its generality. It can thus handle features not being visible in all views by leaving it out of the minimization problem. It can not, however, in its basic form discard outliers, to avoid them. A new and thorough description of bundle adjustment can be found in [55].

5.2.3 Using Lines and Curves

Points are not the only useful feature detectable in an image. Some research has been done in extending the theory to using lines, patches and curves of contours in the images. These primitives are larger, which makes them easier to match correctly. Further more they do not only represent a single point in space, and thus contains information on the connectivity of the surface as well. It is however rather difficult identifying such primitives in a deterministic way.

5.3 Space Carving

A conceptually different method is the so-called *space carving* method first proposed by Kutulakos in [39] and based on the idea of *voxel coloring* in [52]. The idea is that if a part of space is a part of the surface of the scene, then it should be apparent in the images of it. Thus if a part of space in one image appear to be white, and in another black, then it is probably not reflecting light and therefore not opaque and part of the surface. To estimate the surface, an initial block of space, in which the scene is assumed to be, is divided into a number of voxels¹. An iterative algorithm attempts to classify the not occluded pixels as being either *empty*, *partial full* or *full*. The classification is of course based on the image data, but can also use the local classification of the voxel neighborhood, and shading etc.

When no more pixels are removed, then the partial full pixels can be subdivided and carved again to achieve greater model resolution. An example of such an estimated surface can be seen in 5.3, where the voxels are colored and painted as small dots. In theory the interior of such a model is solid, ie. full of non-empty voxels. The voxel model however shows some holes and is thus not solid. The direct space carving method only evaluates a voxel once, thus if a voxel has been erroneously labelled empty, then it will remain empty. When the voxels behind the empty voxel are evaluated, they will not match the images data and can thus risk being labelled empty as well. Thus a single voxel can extent into the model making large wholes. Methods for avoiding this includes using the neighborhood of voxels in the evaluation.

The drawback of the space carving algorithm is the limited spatial resolution achieved compared to the needed voxel space. Thus two 1024×1024 images requires a $1024 \times 1024 \times 1024$ voxel space to use the full resolution of the images.

5.4 Level Set Method Using PDE

The level set method is a time varying PDE² formulation of the reconstruction problem (see [37, 50]). The representation of the surface is the zero crossing of a 4D shape in the 4th dimension, a so-called *level set*. The 3D shape is stored as a 3D scalar field, like the voxel space, where each voxel has a scalar associated to it. Thus all positive voxels are defined as being outside the object and all the negative voxels inside the object. Unlike the binary voxel space used in voxel

¹A voxel is the 3-dimensional equivalent to a pixel

²Partial Differential Equation



Figure 5.1: One of the results of space carving by Broadhurst, Drummond and Cipolla in [6].

coloring, the surface is not constrained to lie on actual voxels. To save space not every voxel is stored, but instead the voxels of interest are stored dynamically, that is the voxels near the current surface are stored. The real advantage using level sets is that they can easily handle topological changes, without any further complications.

Using level sets the surface is estimated by evaluating the gradient of some cost function on the surface, and let the level set representation of the surface descent down this gradient, until it is stabilized.

5.5 Using Deformable Models

Many of the methods known for estimating a surface are using some subtle dependency of the information given in the input images to induce 3D information. This method is a more intuitive approach that produces a photorealistic surface, by simply deforming a model of the scene until it resembles the input images and or data. The method can be compared to a child deforming a block of clay, by simultaneously evaluating it to some memory of how the object should look like. The task of implementing such a deformation is not simple and can be done in many ways. In [60] L. Zhang proposes using an object based gradient

of the image cost, while Vogiatzis in [57] takes a more random approach. When the model is deformed it is to be evaluated using the given input data. If the data consists of images, then Vogiatzis proposed that they are projected onto the model, back into each other and compared using a generic image metric. If the input data is geometric primitives, then the spatial error can be calculated.

This approach to do surface estimation is very general. The model in itself can be represented in many different ways of which a few are a triangular mesh, NURBS³, a voxel space and level sets. The last of these model representations indicates that the level set method described earlier is in fact of this type.

5.6 Frontier Points

A Completely novel method is proposed by Vogiatzis in [30], where *frontier points* are exploited to recover the illumination conditions and material properties. This, usually unknown information of the scene, can later be used together with the input images to make a better estimation of the surface, see fx [34]. It is thus one of the few methods relaxing the Lambertian assumption successfully.

A *frontier point* is defined, with respect to two cameras, as a point on the surface of an object, that is tangential to the epipolar plane between the two cameras and the point. This point has the interesting property that the normal to the surface at that point is known as it is perpendicular to the epipolar plane. Frontier points in an image pair can be obtained by extracting the contour of the surface as seen in the images, and reproject it to find the intersection points in which the projections share the same normal.

The results using frontier points to calculate material and lighting and later use this to reconstruct the surface is astonishing. The detail level in the estimated surface is claimed to be comparable (if not better) to that of a laser scanner, see [30]. The method however requires that both the viewpoints and the lighting conditions are changed systematically giving $K \times M$ input images for K different lighting conditions and M camera viewpoints. The amount of computation involved is denoted as about 5 hours using a high end PC, in [34].

³Non-Uniform Rational B-Spline. A common way to specify parametric curves and surfaces.



Figure 5.2: One of the original (left) images and the result (right) using frontier points, taken from [30].

5.7 Minimization Using Graph Cuts

A surprising approach to solve the surface estimation problem is to transform it into a well known problem in graph theory. In [22] V. Kolmogorov shows that a certain group of energy functions can be minimized by solving a maximum *graph cut* problem. This problem can be shown to be the same as solving the *minimum flow* problem of a graph, which is a well research method in graph theory. To use this approach in surface estimation, the problem must be formulated as a special energy function. The basic idea is to assign vertices to a cartesian grid in a volume of interest in the space. Locally a cost is assigned to every composition of triangles defining the connectivity of the local vertices. This cost is calculated using the input images together with a smoothness term. The edge in the graph going from one vertex to another is set to be some cost of letting that cut be a part of the surface. This defines a large graph over the grid of vertices, in which the minimum cut is sought. Vogiatzis showed the use of this in practice in [29]. One interesting property of using a graph algorithm solving such a problem is that it is deterministic. Thus the result of the algorithm is the global minima of the energy function. The downside is that setting up and solving the graph problem can be relatively slow.

Discussion

The last chapters have shown a part of the huge field of surface estimation. Most of this research has been done in the last decade or so, and the field shows no sign of stopping now. We are getting closer to methods that can be used for practically purposes in real life.

One of the more promising method is using deformable models. The deformations of the model can be performed and controlled in many ways, which makes the method very generic. The fact that graphics hardware offer an superior and ever increasing performance, when projecting and evaluating images and meshes, only makes the choice even more attracting. Therefore this method has been chosen as the main focus of the rest of this thesis. In particular one very interesting method by George Vogiatzis is considered [57].

Part II

Basic Algorithm

The Algorithm

As denoted in the last part, the rest of this thesis will concentrate on a specific approach to do surface estimation, i.e. using deformable models on rigid bodies under constant lighting conditions. This part will describe, document and test the algorithm presented by Vogiatzis in [57], denoted as the *basic algorithm*. The algorithm will be reflected upon and used as the base for a discussion on surface estimation using deformable models. This is used to propose deviations to the basic algorithm of which some will be evaluated using the basic algorithm as a foundation.

This chapter will introduce the algorithm in its most simple form, and then gradually add deeper insight, up until the next chapter that will describe it all in a *Bayes* framework using *simulated annealing*. Next the design of a mesh representing the surface will be described in Chapter 9, followed by a documentation of the implementation of the algorithm in Chapter 10. At this point Chapter 11 introduces the datasets used in this thesis, which is used to test and evaluate the basic algorithm and some deviation to it in Chapter 12. Finally the results are discussed and a conclusion on the basic algorithm is made in Chapter 13.

7.1 Foundation

The basic idea that makes the foundation for the algorithm is the general observation that:

Having the correct surface of some object, and a set of input images and the corresponding cameras thereof.
Then projecting an arbitrary input image onto the surface and back into an arbitrary (possibly the associated) camera, will match the image of that camera in the domain of the image being visible from the viewpoints of both cameras.
If on the other hand, the projection does not result in a perfect match, then the surface is not correct. Thus a measure of the 'correctness' of a surface, with respect to a set of input images, can be defined as the how well, the surface reprojects the images into each other.

This is an idealistic presentation of the fundamental idea, since it assumes a number of idealistic assumptions of the world. These assumptions and a short explanation is listed in the following.

1. The object is Lambertian. Thus every point on the surface radiates light of the same color and intensity in all directions, and is thus perceived as having the same color in the input images in which it is visible.
2. The input images must be of infinite resolution. If the images were a quantification of the captured light rays, then the match may fail due to the pixels not being projected perfectly into each other.
3. The images are noiseless and thus resembles the scene correctly as seen from their viewpoint.
4. The projection¹ of images onto the surface and the reprojection into a camera are assumed to be perfect.
5. Lighting conditions are constant.
6. The space outside the object are completely transparent and space inside the object is completely opaque.
7. The images are taken at the same time, or the scene is static.

¹It should be noted that the term *project* is used in a mathematical sense, and not as light would normally be projected as from a projector. In other words the perceived color and intensity of a surface point is invariant to the normal to the surface at that point.

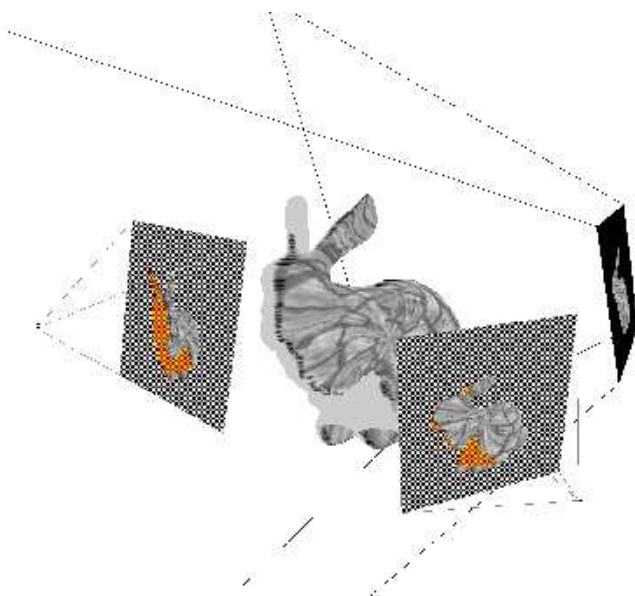


Figure 7.1: Figure showing the famous Stanford bunny textured with a brick texture, and three cameras from different viewpoints. The rightmost camera is projecting its image onto the bunny, and the two other cameras show their perception of the scene. The breast and part of the head is not visible from the rightmost viewpoint, and thus the reprojections sees these areas as occluded (red and yellow).

These assumption can somewhat be relaxed since the underlying idea still holds without them. However when doing so, it is only possible to estimate the surface up to a certain quality. Thus the objects can be a little non-Lambertian, final resolution images with a relatively small noise can be used, reprojections can contain a small error, the space can be a little opaque (like air) and finally the object can be a little different in all the input images. An illustration of a scene with 3 cameras can be seen in figure 7.1.

7.2 Formulation

As described in section 5.5, this method deforms a current *model*, M , by iteratively evaluating the model and discarding bad deformations. Using the measurement of correctness as the evaluation of the model, the problem can be

expressed as a minimization problem.

The evaluation of the model can be done in two different ways depending on the viewpoint of the algorithm, an *object centered* and an *image centered*. If an object centered algorithm is used, a number of points are sampled uniformly over the model. These points are projected into the images fetching colors, which are compared using some similarity measure. This is the approach used in [26, 60]. The image centered approach on the other hand projects points sampled in the image (the pixels) and projects these onto the model, and back into the other images where they are compared using a so-called *image metric*. By letting the number of points sampled go to infinity, the two viewpoints can be described as either integrating a cost function over the surface, or over the images.

The goal of the algorithm is a photorealistic surface estimation, thus it is more natural to use an image centered approach, which is what is done in the algorithm by Vogiatzis. When sampling points on the surface, the density of the samples projected into the images will rarely be uniform. Therefore some parts of the surface is weighed harder in the evaluation, which is not 'fair' as the input data is in the image domain. An example of this is a surface patch oblique with respect to the frustum of a camera. It is only barely visibly in the camera, and thus should be weighed lower than a surface patch parallel to the image plane. In [60] Zhang and Seitz accounts for some of this by classifying primitives of their surface into three classes: *oblique*, *far-away* and *typical*. It would be possible, but computationally expensive to adjust the matching error of each sampled point with respect to each camera using the surface normal and the distance to the cameras.

The image centered approach is chosen in this thesis, following the ideas of Vogiatzis. Having an image metric one can now formulate an *objective function*, which should be minimized. This is done by iteratively deforming a *model* of the surface of the scene, hence the term *deformable model*. For each iteration the quality of the model is evaluated using the objective function, and a decision on keeping the last deformation is based hereon. At a specific step in the minimization the model is thus the current estimation of the surface.

7.2.1 Algorithm

The minimization of this objective function can now be formulated into a very simple algorithm:

Algorithm 1 Basic Algorithm I

```

while converging do
  deform the model
  evaluate objective function
  if better than not deformed model then
    keep the deformed model
  else
    discard the new model
  end if
end while

```

7.2.2 Mathematical Formulation of the Problem

To formulate the problem mathematically, the notation of Pons in [47] are used. The surface is therefore denoted

$$S \subset \mathbb{R}^3 \quad (7.1)$$

, the set of images $I_1 \dots I_n$, and the projection from world coordinates to image i ,

$$\Pi_i : \mathbb{R}^3 \rightarrow \mathbb{R}^2. \quad (7.2)$$

The later also exists in an inverse version for reprojecting the image onto the surface, however it is dependent on the surface, thus it is denoted $\Pi_{i,S}^{-1}$. The part of the surface, S , visible in image i is denoted S_i , and thus the reprojection can be described as a function from S projected into I_i to S_i ,

$$\Pi_{i,S}^{-1} : \Pi_i(S) \rightarrow S_i \quad (7.3)$$

The input images has a number of channels denoted by d , which is normally either 1 for gray scale images, 3 for color images or 4 for color images with an alpha channel. Thus an image can be described as the function

$$I_i : \Omega_i \subset \mathbb{R}^2 \rightarrow \mathbb{R}^d \quad (7.4)$$

, where Ω_i is the part of \mathbb{R}^2 space that image i describes.

Using equation 7.1, 7.2, 7.3 and 7.4 the projection of image i , I_i onto the surface, S and back into image j , I_j can be described by a simple function

$$I_j \circ \Pi_j \circ \Pi_{i,S}^{-1} : \Omega_i \cap \Pi_i(S_j) \rightarrow \mathbb{R}^d \quad (7.5)$$

Thus from an image, i , and the surface, S , one can reconstruct the part of another image, j , that is also visible in the first image. Since I_i is only defined for Ω_i , and the reprojection of image j is only defined for $\Pi_i(S_j)$, the domain of the reconstruction in image i is only defined for

$$\Pi_i(S_j) \cap \Omega_i \quad (7.6)$$

When the reprojection is defined we introduce a generic image metric M , that compares the dissimilarity of two images into a single scalar

$$M : (I_i, I_j) \rightarrow \mathbb{R} \quad (7.7)$$

Using equation 7.7, we can now introduce the final image matching term, \mathcal{I} , of the surface, that compares all the ordered pairs of images (i, j) using the image metric M in the domain visible in both the original image and the reprojected image.

$$\mathcal{I}(S) = \sum_i \sum_{j \neq i} M|_{\Pi_i(S_j) \cap \Omega_i} (I_i, I_j \circ \Pi_j \circ \Pi_{i,S}^{-1}) \quad (7.8)$$

Thus the surface estimation problem can then be expressed as

$$\min_S \mathcal{I}(S) \quad (7.9)$$

Nothing is assumed of the representation of the surface, S . If this representation used a fixed number of variables, then the minimization could be done using the Levenberg-Marquardt method like in section 5.2.2. It would however be beneficial to exploit some of the special properties of this problem, which is what will be done here.

7.3 Image Similarity Measure

Having two images I_i and I_k we wish to be able to calculate a single scalar value describing how similar these two images are. It is also called an *image metric*, since it allows the measuring of the distance between images, like in a metric space. A simple and widely used method is the summed squared error, SSE, or the closely related mean squared error, MSE, the later being simply a normalized version of the first. SSE is defined as:

$$\mathcal{I}(I_i, I_k) = \sum_{u,v} (I_i(u, v) - I_k(u, v))^2 \quad (7.10)$$

When applied to two similar images, SSE will give a low *image cost*, while applied to two different images will give a really large cost because of the quadratic term. This is a desirable property, since small errors are avoidable because of numerical errors and noise and thus should be neglectable in the big picture.

In the real world captured images may have been exposed to different lighting conditions, and or an automatic gain control. The result of this is that images may differ globally, giving rise to high errors even when the ground truth surface is used. It may thus be possible for the model to find a not true surface having a lower image cost. The differences can be modelled as an affine transform between the images color space, see [27].

Vogiatis suggests normalizing the images beforehand to account for these global differences. He then assumes that each pixel intensity in each image I_i is normally and independently distributed about the corresponding image intensity in the image reconstructed using projections. Using this he defines the similarity measure to be SSE. It may however be interesting to use the correlation of the data, CORR, instead. CORR is a similarity measure which is independent of the mean and the variance of the data. This would thus rule out the affine transformation in color space, which could then be neglected. The sample correlation is given as.

$$\frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2)}} \quad (7.11)$$

The downside using correlation is that it requires at least two inspections of every variable, one to calculate the mean and one to calculate the upper term of equation 7.11.

The two similarity measures described up till now, however, are not the only way to compare two datasets. One could use *mutual information* in the two images, by regarding them as dependent populations X and Y , and calculating

$$\mathcal{I}(I_i, I_k) = I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{f(x)g(y)} \quad (7.12)$$

One way to do this is to estimate the entropy, and the joint entropy of the images, which can be done using the Parzen window method as in the algorithm by Pons, see [47]. If these are estimated well, that is not only estimated as a first order entropy, then this image metric would not only give a measure of how close each pixel are to each other, but also use the relationship between the pixels in the image. This metric however is relatively complex and is therefore not considered further here.

7.3.1 The Occlusion Problem

The image metrics discussed here are simply some means to compare data. Thus the input needs not to be a full image. The pixels assumed to be occluded in either I_i or I_k , that is $W/\Pi_i(S_k) \cap \Omega_i$, can therefore be left out of the comparisons. This is both good and bad. It ensures that only relevant data is compared, however, it also enables the model to fool the objective function. The reason for this is that the model can deform in such a way that areas of the input images giving high image cost becomes occluded and thus not counted in. The effect of this is like a so-called lenticular print, that shows two different images if viewed from two different angles, see figure 7.2. The result is that all pixels are occluded in all images. This trick only works for two cameras, however it may still be worthwhile for the model to adopt such a surface, if the matching cost between two cameras are high. Using SSE represents an even worse problem, since SSE is depending on the number of un-occluded pixels. The result can be that the model shrinks, and eventually only covers a few pixels, that has low image cost.

Vogiatis does not mention this problem, and thus does not present a solution to avoid it. A common way to counteract this shrinking is to add a ballooning effect to the mesh. This however does not prevent erroneous occlusion like in figure 7.2 and thus something more sophisticated is needed. The calculation of the image metric involves labelling pixels as either visible or not, and thus a counting of not visible pixels could be done simultaneously. The count of these *occlusion pixels*, could then be used to add a small, but significant penalty for each occluded pixel to the objective function. It should be large enough to

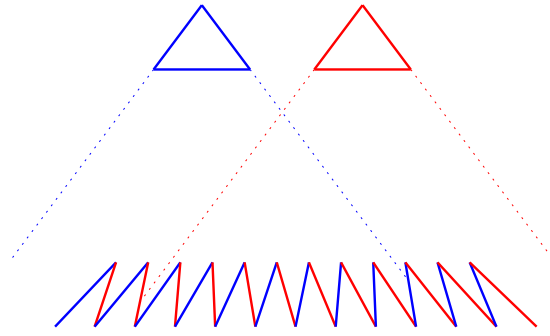


Figure 7.2: A figure of the workings of a lenticular print describing a problem that can occur in the algorithm. All the blue surfaces can only be seen from the blue camera and visa versa.

avoid shrinking, but small enough to make it worthwhile for the model to have occluded pixels, when the true surface has.

To further develop this idea an *alpha pixel* is introduced. The term alpha pixel comes from reasons being described later in Chapter 10. It is introduced because of the observation that a pixel can be not-visible for two reasons. The first is real occlusion, that is the model describes an object in this pixel, but it is not visible in the any other image. These are the pixel denoted as occlusion pixels and penalized as described before. The second is that when comparing an image to a reprojection and either of them defines a certain pixel, that is if $p = (u, v)$ is the pixel, then $p \notin \Omega_k \wedge p \notin \Omega_i$. These are the alpha pixels, that can be used to counteract the shrinking effect. The alpha pixels should be penalized harder than occlusion pixels, since occlusion is bound to happen in real object and alpha pixels should not. A figure illustrating an object with alpha and occlusion can be seen in 7.3. One further advantage of using alpha pixels in the objective function is that they enable a domain mask for the input images. The domain mask can be given using the alpha channel of the images, hence the term alpha pixel.

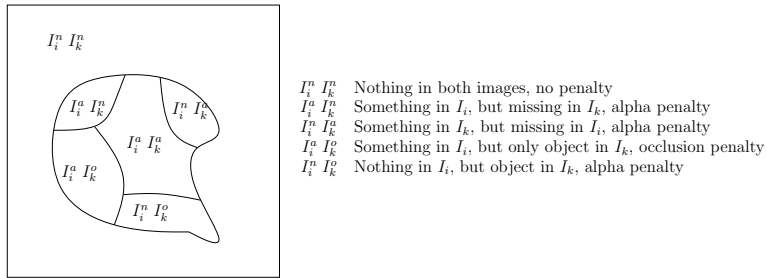


Figure 7.3: Figure showing the different possible compositions when comparing an image I_i to a reprojection I_k . n means that the area is not defined for that image, a means that it is defined and o means that it contains an object, but is not defined.

The Bayes Formulation

As the previous chapter showed, surface estimation can be done by minimizing an objective function in a simple iterative algorithm. However even though the algorithm in time will converge to a minima of the objective function, it can be difficult to control that the minima is the global minima. This problem can be addressed by setting the problem up in a *Bayes* framework using *simulated annealing*, as done in Vogiatzis work. This chapter will introduce this idea together with the related concepts, and use it to improve the basic algorithm.

8.1 Bayes Theorem

The Bayes theorem, introduced by Thomas Bayes in 1764 in [12], describes a relationship between conditional properties and their unconditional counterparts. It can be formulated as: "The probability of a hypothesis, H , conditional on a given body of data, E , is the ratio of the unconditional probability of the conjunction of the hypothesis with the data to the unconditional probability with the data alone", [2]. This can be expressed as the formula

$$p(H|E) = \frac{p(E|H) \times p(H)}{p(E)}. \quad (8.1)$$

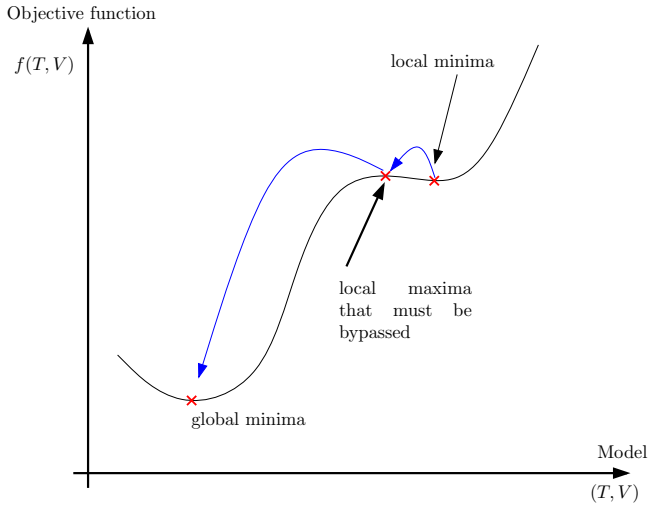


Figure 8.1: Example of a model trapped at a local minima. To continue the descent the local maxima must be bypassed.

, or using the classical notation

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{normalizing constant}}. \quad (8.2)$$

, where the **posterior** is H , conditional on E , the **likelihood** is E conditional on H and the **prior** is the probability of the hypothesis.

Using this to formulate the probability of a model can be expressed as: Having a model, M , the probability for it to be correct is the probability of it given the data, I (the image metric), times the probability of the model alone (a mesh cost) divided by a normalizing term. This gives the simple expression of the posterior of a model given a set of images:

$$p(M|I) = \frac{p(I|M) \times p(M)}{p(I)}. \quad (8.3)$$

As it can be seen, the normalizing term is actually the somewhat artificial probability of the input images and their pose, however, as this term is constant for all iterations it can be neglected when the resulting probability is compared to the probability achieved in another iteration. To calculate the probabilities

negative logarithmic are taken of their cost function. The cost function for the model with respect to the data are already defined as the matching function \mathcal{I} , however the cost of the model is yet to be defined, for now it is denoted as $\mathcal{M}(M)$, where M is the model. If inserted into equation 8.3 this gives:

$$p(M|I) = \frac{\exp(-\mathcal{I}(I)) \exp(-\mathcal{M}(M))}{p(I)} = \frac{\exp(-f(M))}{C} \quad (8.4)$$

, where f is the objective function $\mathcal{I}(I) + \mathcal{M}(M)$ and C is the normalizing constant.

Now the *importance ratio*, μ , of a new model compared to another model can be defined as

$$\mu \leftarrow \left[\frac{p(\mathbf{M}'|\mathbf{I})}{p(\mathbf{M}|\mathbf{I})} \right] \quad (8.5)$$

, where \mathbf{M} and \mathbf{M}' are two different *proposals*. The importance ratio thus describes a measure of belief in \mathbf{M}' compared to \mathbf{M} . If $p(\mathbf{M}'|\mathbf{I}) > p(\mathbf{M}|\mathbf{I})$, then $\mu > 1$ and thus \mathbf{M}' should be chosen over \mathbf{M} . If on the other hand $p(\mathbf{M}'|\mathbf{I}) < p(\mathbf{M}|\mathbf{I})$, then \mathbf{M}' should only be chosen over \mathbf{M} with probability μ . The relationship is shown in figure 8.2, where the flat top indicates that M' is always chosen.

8.1.1 Algorithm In a Bayes Framework

Using the Bayes formulation a new and hopefully better algorithm can be setup:

Algorithm 2 Bayes framework

```

M  $\leftarrow$  M0
while converging do
  deform the surface, to get M'
  set the importance ratio  $\mu \leftarrow \left[ \frac{p(\mathbf{M}'|\mathbf{I})}{p(\mathbf{M}|\mathbf{I})} \right]$ 
  with probability  $\min(1, \mu)$  set M  $\leftarrow$  M'
end while

```

Now, why is this better compared to the straight forward approach where the result of the objective function is compared directly and the best is chosen? The answer is that it is not necessarily better, however, it allows for a more

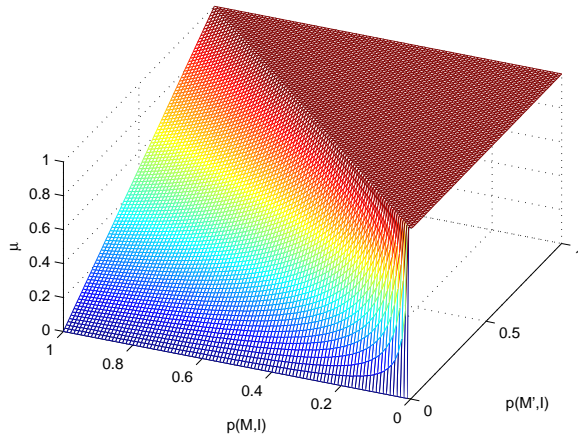


Figure 8.2: Figure of the relationship between $p(\mathbf{M}'|\mathbf{I})$, $p(\mathbf{M}|\mathbf{I})$ and μ . \mathbf{M}' is chosen over \mathbf{M} with probability μ .

lively convergence, that will not always choose the best solution. Thus it has a chance of escaping local minima, and reaching, if not the global minimum, then a better minima. Figure 8.1 illustrates this problem, where the probability that a long jump in search space escaping the local minima is taken is small. If small jumps up hill are allowed, then a small jump in the correct direction, but with a relatively small extra cost, will make the jump escaping the minima more probable.

8.2 Using Simulated Annealing

Another method to avoid being trapped in a local minima is to use simulated annealing, as stated in [20, 41]. The name comes from the idea to simulate the process when metal cools and anneals, which brings it from a state of total disorder to an ordered grid of metal atoms. The minimum energy state for a piece of metal is when it is one large crystal, however normally it consists of a number of smaller crystal structures that are not aligned. One large crystal is seldom achieved, since the metal starts crystalizing at different places of the metal with no constraints on the alignment of the local crystal structure. The size of the crystal structures in the metal varies with the annealing process, and can thus be controlled. This process can be simulated when one wants to

minimize an objective function, and thus give the method a larger chance to converge to the global minima, and not get trapped.

The idea, to simulate the annealing, is to start the algorithm making proposals from a 'wild' proposal distribution and slowly let the proposals be closer to the current model. To do this one can define an artificial temperature T , that starts at some starting temperature T_0 , and gradually descends using an *annealing schedule* described further later. For each iteration, k , a new model M' , the proposal, is sampled using one of the deformations, with the current temperature as a measure of how lively it should be done. Thus when having a high temperature, the spatial movement between proposal and model is larger, and when the system cools down they become smaller. The acceptance of the models should also be affected by the temperature, thus models not being better should have a larger chance being chosen when T is high. The later can be expressed using

$$\mu = \exp\left(-\frac{\delta f}{T}\right) \quad (8.6)$$

, where δf is the difference in the evaluation of the objective function, ie. $(f(M') - f(M))$. From equation 8.6 an expression closer to equation 8.5 used for the Bayes formulation, can be derived:

$$\mu = \exp(-\delta f)^{\frac{1}{T}} = \exp(-(f(M') - f(M)))^{\frac{1}{T}} = \left[\frac{\exp(-f(M'))}{\exp(-f(M))} \right]^{\frac{1}{T}} \quad (8.7)$$

From equation 8.4, using the Bayes framework, we have that $p(M|I) = \frac{\exp(-f(M))}{C}$. Inserting this into equation 8.7 yields the result used in Vogiatzis work:

$$\mu = \left[\frac{p(M'|I)}{p(M|I)} \right]^{\frac{1}{T}} \quad (8.8)$$

From this equation it can be seen that altering the temperature of the system does not change the acceptance when $\frac{p(M'|I)}{p(M|I)} \geq 1$. But if $\frac{p(M'|I)}{p(M|I)} < 1$, then a high temperature will make M' more probable and visa versa with a low temperature.

8.2.1 Numerical Aspects of μ

Evaluating equation 8.8, can give serious numerical errors, if it is done directly. When calculating the posterior of some model from equation 8.4, e is raised to the negative power of the objective function. There is no assumption on the range of the objective function, except that they should be positive. It thus only requires the cost of a model to be above approximately 750 before the result is rounded off to zero, depending on the numerical precision of the system. Dividing by zero gives an error or infinite, and in both cases the choice of model can not be done.

This problem, however can easily be solved by rearranging the evaluation of μ :

$$\begin{aligned}
 \mu &= \left[\frac{p(M'|I)}{p(M|I)} \right]^{\frac{1}{T}} \\
 &= \left[\frac{\exp(-f(M'))}{\exp(-f(M))} \right]^{\frac{1}{T}} \\
 &= \exp(-f(M') + f(M))^{\frac{1}{T}} \\
 &= \exp\left[\frac{f(M) - f(M')}{K} \right]; \tag{8.9}
 \end{aligned}$$

This formulation avoids using exponential factors, before all other calculations are done. The numeric precision can still be overflowed, however if so, the result will be infinite, which ensures that the new model is chosen anyway.

8.2.2 The Annealing Schedule

The annealing schedule is the rate at which the system cools down. In [28], Bounds formulated the choice of the annealing schedule as follows: “choosing an annealing schedule for practical purposes is still something of a black art”. The annealing schedule should take the temperature from some starting temperature T_0 and gradually descent it. This can be done in many ways, however two of the most used is a *stepwise linear* and an *exponential* method. The cooling can both be based on the number of iterations of the algorithm, i.e. the number of proposals, or the number of successful iterations, i.e. the accepted proposals. The last is generally more robust, since it ensures that something is happening in the system while the temperature is lowered. In the rest of this thesis the term *iterations* is used of accepted proposals. The following describes the

two mentioned annealing schedules together with the slightly different schedule proposed by Vogiatzis, which is of exponential nature.

Exponential Using a constant α smaller than, but close to 1, each successful step the temperature is lowered by

$$T_{k+1} = \alpha T_k \quad (8.10)$$

This way the temperature will go down fast to start with and then gradually descent slower and slower. α is usually chosen to be 0.95 or higher to prevent a too fast descent.

stepwise Linear For every L successful iterations, the temperature is lowered by ΔT

$$T_{k+1} = T_k + \Delta T. \quad (8.11)$$

Using this schedule allows the system to stabilize at every level.

Vogiatzis's Schedule The annealing schedule proposed by Vogiatzis is a little different, however it is of exponential nature. It is defined by

$$T_k = \frac{C}{\log(1 + k)} \quad (8.12)$$

, where C is a constant.

The three schedules used are shown in figure 8.3. The schedule proposed by Vogiatzis is shown using $C = 0.4$, which is equivalent to the schedule used in his paper. The other are shown with $\alpha = 0.998$, $T_0 = 0.4$, $L = 20$ and $\Delta T = 0.008$. Vogiatzis proposal descends extremely fast to $T \approx 0.1$ in under 200 iterations. In his work, it is mentioned that approximately 2500 iterations are needed to converge. Thus 2300 iterations are done using a relatively stable temperature. This makes the probability of choosing a proposed model with $\delta f = 1.0$ in the magnitude of 10^{-5} . It may thus be interesting to investigate the impact the annealing schedule has on the overall performance of the algorithm.

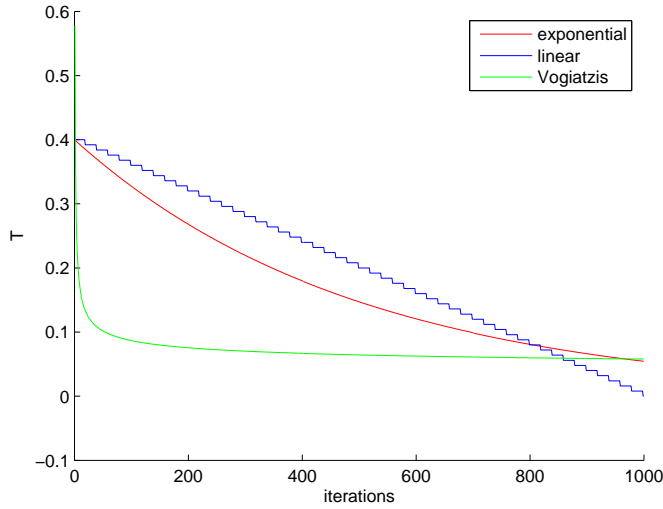


Figure 8.3: The three annealing schedules

8.2.3 Simulated Annealing in a Bayes Framework

Putting these two methods together using the annealing schedule proposed by Vogiatzis yields the final algorithm, which is the one used in his work and also the one that we will base our implementation on.

Algorithm 3 Simulated annealing

```

(V, T)  $\leftarrow$  (V0, T0)
 $k \leftarrow 1$ 
while converging do
   $K \leftarrow \frac{C}{\log(1+k)}$ 
  deform the surface, to get (V', T')
  set the importance ratio  $\mu \leftarrow \left[ \frac{p(\mathbf{V}', \mathbf{T}' | I)}{p(\mathbf{V}, \mathbf{T} | I)} \right]^{1/K}$ 
  with probability  $\min 1, \mu$  set (V, T)  $\leftarrow$  (V', T')
end while

```

A Deformable Mesh

One of the cornerstones of the algorithm is the mesh, which represents the current model. Up until now, the presence of such a deformable mesh has been assumed, however designed it is no trivial task. The design of the mesh should fulfill the requirements listed in the following:

- Represent the surface.
- Be deformable, both in shape and connectivity.
- Capable of *rolling back* deformations.
- Define a *mesh metric*.

This chapter will describe the design of the mesh chosen, and how the requirements of the mesh is met. The actual implementation of the mesh is first discussed in Chapter 10.

9.1 Mesh Representation

A mesh can be viewed as a piecewise linear approximation of a surface. It can be defined as a set of vertices, $V = (v_1, \dots, v_m), v_i \in \mathbb{R}^3$, giving the *shape* of the mesh, and a set of triangles, $T = (t_1, \dots, t_n), t_j = (v_a, v_b, v_c)$, determining the *connectivity* of the mesh, i.e. $M \equiv (V, T)$. It should be noted that since the model of the algorithm and the mesh are equivalent, they are both denoted M and the terms are used interchangeably.

The representation using triangles are not the only known. Another well known approach is the *half edge* mesh, where so-called half edges determine the connectivity of the mesh. The half edge mesh has the advantage over triangular meshes, that it can represent polygonal surfaces as simple primitives in contrary to the triangle mesh, that can only represent triangles. It is however unlikely that a mesh in this algorithm will contain a large number of coplanar triangles that could benefit from half edges. Because of this and reasons being described further in section 10.2.3, the triangular mesh has been chosen.

The next two sections will describe operations and enforcements on the mesh.

9.1.1 Mesh Operations

To ease the description of the mesh deformations, a set of simple operations are introduced here, inspired by the operations in [56]. As a reference figure 9.1 shows a triangle and the vertices it connects.

- $\lceil v \rceil$: The owners¹ of the set of vertices v .
- $\lfloor t \rfloor$: The vertices of the set of triangles t .
- $Edges(M)$: A set of all the edges in M .
- $NBE(M)$: A set of all non-border edges in M .

9.1.2 Mesh enforcements

To simplify operations on the mesh, some enforcement on the connectivity are made. These enforcements must be held in the initial mesh, and whenever the

¹An owner of a vertex is a triangle having the vertex as a corner.

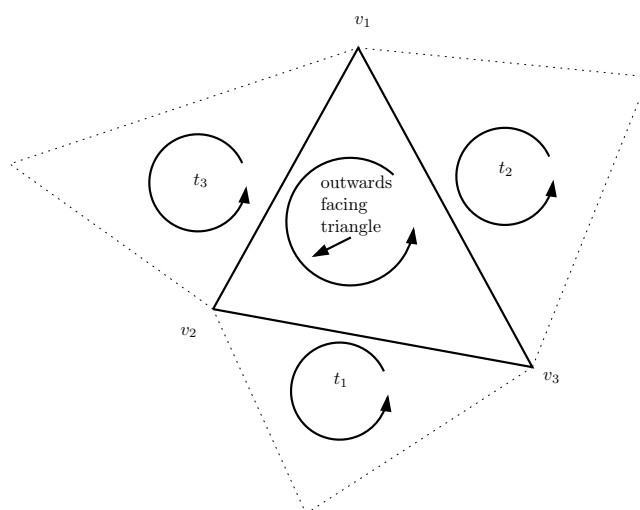


Figure 9.1: Figure showing a triangle connecting three vertices. Notice that because of the righthand rule, the triangle is outwards facing.

mesh is deformed to ensure persistent mesh properties. The first is to ensure that all triangles have the same orientation. Figure 9.1 shows the orientation of a triangle and its neighbors. Consistent triangle orientation is useful in two ways. First, it allows deterministic searches which again allows deterministic operations on the mesh. If there were no orientation enforcement on the mesh, the same operation on the same triangle, but with different orientation, could have different results. Second, the mesh is to be rendered on graphics hardware, which distinguish between clock wise and counter clock wise triangles. Usually the right hand rule is used to define a front face and a back face of a triangle, which can be rendered differently.

The other enforcement is that an edge $e \in Edges(M)$ may never be shared by more than two triangles. This greatly simplifies searches and operations on the mesh, while making the structure for containing the mesh simpler. Since a surface is defined as a 2-manifold, it does not constraint the solution space.

9.2 Defining a Mesh Smoothness Term

The image metric is useful for measuring how well the surface fits the images, however using this alone to do surface estimation could easily end up with ill

formed surface, that would not resemble the object being reconstructed. In other words, if nothing is said about the surface, it is likely that it will be deformed into a spiky surface, that fits the images according to the image metric, but does not have the smoothness, that is normally expected of real world objects.

To account for this, one can add a *mesh smoothness term*, \mathcal{M} , to the objective function, to make it penalize spiky surfaces or surfaces having other unwanted properties. Vogiatzis proposed a smoothness term penalizing the angles between the normal of the triangles and the size of the mesh. A non-border edge $e \in NBE(M)$ has two neighboring triangles with the normals n_1 and n_2 . A simple way to calculate a measure of the angle, $\cos \vartheta(e)$, between the two normals is to use the dot product, ie. $\cos \vartheta(e) = n_1 \cdot n_2$, which gives 1 when they are parallel and -1 when they are opposing. A non-negative error can thus be formulated as:

$$\mathcal{M}(\mathbf{M}) = A \text{size}(V) + B \sum_{e \in NBE(M)} (1 - \cos \vartheta(e) \|e\|) \quad (9.1)$$

, where A and B are constants. This smoothness term has a number of desirable properties. The mesh is to be deformed into a shape defined by the image data. This requires the mesh to adjust its resolution locally, which should not change the cost of the mesh, except the extra cost of the new vertices. The smoothness term defined has this property, since the angular term is defined on the global curvature. Thus both triangles and edges can be divided without changing the angular cost, as illustrated in figure 9.2.

Research has been done into defining more advanced smoothness terms. Instead of simply evaluating all edges, the local neighborhood could be used. Thus it could be penalized harder if a sudden edge appears in an otherwise smooth surface, while it should be acceptable if the edge is part of a larger structure. Other regularization terms, that could be added to the smoothness term is a triangle skewness penalty to avoid too skew triangles or an odd-size penalty to obtain a smooth mesh resolution.

9.3 Deforming a Mesh

Each loop in the algorithm samples a new model from a proposal distribution around the current model. The model in this context is the mesh, thus the mesh must be changed or more correctly expressed deformed, using this distribution.

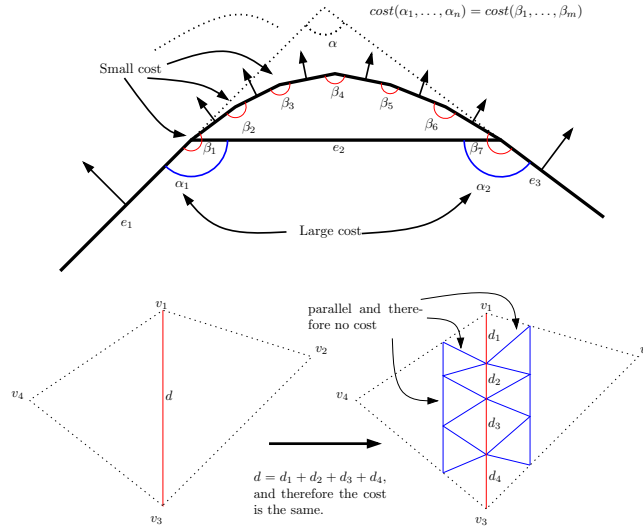


Figure 9.2: Figure showing how the angular smoothness term are invariant to the division of triangles and edges.

A deformation is defined as an operation on the mesh taking it from $M \rightarrow M'$. Gelman *et al* [7] lists 5 criteria for a good proposal distribution.

1. It is easy to sample from the proposal function, $M \rightarrow M'$, i.e. it is easy to deform the mesh.
2. It is easy to calculate μ , or in other words the objective function.
3. Each deformation jumps a reasonable distance in parameter space. If not the model would deform too slowly.
4. Proposals are accepted with a reasonable frequency. Otherwise too many proposals are rejected making the model standing still.
5. Any model can be reached using a series of deformation from any other model.

Vogiatzis proposes two different groups of deformation to account for this, *vertex deformations* altering the shape by changing the position of the vertices and *connectivity deformations* changing the connectivity of the mesh. The vertex deformations can take the model reasonably around the parameter space, both at random, and using the image data as a guidance to speed up the convergence.

The connectivity deformations ensures that all parts of parameter space can be visited. All the deformations are designed such that they are relatively simple and easy to implement to satisfy the easy sampling, while it is always easy evaluating the objective function as already described.

Sampling a model in the proposal distribution is thus simply to deform the surface using one of the deformations at random. However as will be exploited later, sampling is not limited to a single deformation, and thus a series of deformations can be merged to allow both connectivity and shape changes in a single deformation. To define the liveliness of the deformations the δ -value are introduced. The temperature defined in section 8.2 is used to control the δ -value, however it can not be used in itself to define the liveliness as no constraints on the spatial size of the scene has been made. The δ -value is defined as

$$\delta = C_\delta T \sum_{c \in (cams)} P(c, ||c - M_{\text{center}}||) \frac{1}{||cams||} \quad (9.2)$$

, where C_δ is a constant, M_{center} the center of gravity of the mesh calculated by $\frac{\sum_{v \in |M|} v}{size(M)}$ and $P(c, d)$ is a function calculating the width of a pixel at distance d from the camera c . The δ -value is thus a measure of the average distance in space at the center of the mesh corresponding to a single pixel modified by the temperature and the constant C_δ . As the mesh is deformed, the δ -value must be updated regularly.

Now we can start describing the actual deformation types. Vogiatzis proposes 6 of these, 3 changing the shape, and 3 changing the connectivity. These are described in the following sections with the 3 vertex deformations first. It is of uttermost importance that all deformations preserves the properties of the mesh. A thorough description of how this is achieved is included, however as it can be technical and outside the scope of the reader it can be skipped.

9.3.1 Random Deforms

The random deformation type is the simplest vertex deformation. It simply samples a displacement vector, δv , from a normal distribution with spread equal to the current δ -value. This is added to a random vertex, v , yielding $v \leftarrow v + \delta v$.

In [47] J. Pons *et al* mentions, that for small vertex changes only the change along the surface normal at that vertex changes the visual appearance of the mesh. This especially holds if the mesh is smooth, which is expected as the

mesh is regularized by the smoothness term. Using this it would be interesting to evaluate the differences between displacing a vertex at random or along the normal.

The random deformations, and the vertex deformations in general does not change the connectivity of the mesh and does therefore automatically preserve the enforcements.

9.3.2 Gradient Deforms

Moving in parameter space at random may take a long time to converge. To speed up performance some of the input data can be used to guide the deformation. A gradient deform is a small displacement of a random vertex, v , in the opposite direction of the gradient of an estimate of the image metric when varying the coordinates of v . Like the full image metric, this estimate can either be image centered or object centered. Vogiatzis proposes an object centered approach, where a number of sample points on $[v]$ are projected into the images. The resulting colors are compared using SSE, however other metrics could be used as well.

On the other hand an image centered gradient deform samples a number of pixels in each input image around the projections of v . Thus oblique triangles would be weighed lower like using the full image centered matching term.

9.3.3 Correlation Deforms

Another way to exploit the input images in the deformations is to use the epipolar geometry of a random vertex, v and match a small window of image data along the corresponding epipolar lines in the images using one of the similarity measures. To do this one image is chosen as the *base image* from which epipolar lines in the remaining images are found. To avoid outliers the search is limited to a distance d around the projections of the vertex. Because of its connection to the epipolar geometry, the correlation deforms are also denoted as *epipolar deformations*.

Basically this is the same as is done in stereo matching, and thus the interested reader is referred to [51] for more information. Like with the random deformations, it may be beneficial to move the vertex along the normal vector instead of the epipolar lines. The search would still be a search along lines in the images, but the search would be object centered, which in this case is more natural than

selecting a single image as the base.

9.3.4 Edge Swap

The edge swap deformation takes an edge e and swaps it in the quadrilateral in such way that the edge now connects the two vertices it did not connect before, see figure 9.3.

The input for a swap deformation is the edge, e , consisting of the two connected vertices v_1 and v_2 . From these the two triangles t_1 and t_2 making up the quadrilateral can be found using $(t_1, t_2) = \lceil v_1 \rceil \cap \lceil v_2 \rceil$. These can again be used to get the two other vertices v_3 and v_4 using $(v_3, v_4) = \lfloor t_1, t_2 \rfloor / (v_1, v_2)$. The two neighboring triangles n_1 and n_2 can be found in a similar way using the correct pair of vertices. It should be noted that these operations can be done in an oriented manner exploiting the orientation of the triangles, which returns an oriented list rather than a set. If this was not the case, then operations on the mesh like this would not be deterministic. For the mesh to be correct after the swap operation the following must be ensured:

- t_1 must swap v_1 with v_4 .
- t_2 must swap v_3 with v_2 .
- ownership of the two vertices v_1 and v_2 must be set to t_2 and t_1 .
- n_1 must swap neighbor t_1 with t_2 .
- n_2 must swap neighbor t_2 with t_1 .
- t_1 and t_2 are notified that their position have changed.

It is important that one can only swap e if both the triangles t_1 and t_2 are present, thus e cannot be a bordering edge.

9.3.5 Edge Collapse

The collapse deformations are the deformations used to simplify the mesh. It takes as input an edge e connecting two vertices v_1 and v_2 , see figure 9.4. The two neighboring triangles sharing e , $(t_1, t_2) = \lceil v_1 \rceil \cap \lceil v_2 \rceil$, are removed from the mesh together with v_2 , as v_1 takes over the triangles connected to v_2 . The edge

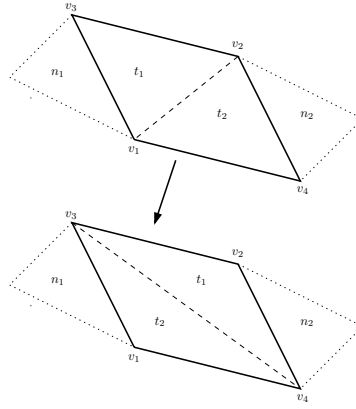


Figure 9.3: Illustration showing the swap deformation.

e can be a border edge, in this situation either t_1 or t_2 does not exist and thus the corresponding set of neighbors can be ignored. To ensure mesh correctness the following steps must be performed:

- The necessary triangles and vertices must be obtained from the two known vertices v_1 and v_2 . If t_1 exists n_1 , n_2 and v_3 are obtained, and if t_2 exists n_3, n_4 and v_4 are obtained using the operations described.
- v_2 is swapped with v_1 for all triangles around v_2 , $\lceil v_2 \rceil / (t_1, t_2)$, since v_2 will cease to exist.
- neighboring relation (connectivity) between n_1 and n_2 and between n_3 and n_4 are created if they exist.
- v_1 's owner is set to be one of n_1 , n_2 , n_3 and n_4 , that exists.
- v_1 is repositioned to the center of e .
- All triangles around v_1 in the deformed mesh are notified, that their relationship and vertices may have been changed.
- v_2 , t_1 and t_2 are removed from the mesh.

It can happen that n_2 and n_4 or n_1 and n_3 are the same triangle. If so no special care needs to be taken, since they should be altered as both neighbors, at the same time, and not as a single. It should be noted that the n_i triangles cannot be the same as t_1 or t_2 since two triangles cannot share more than one edge.

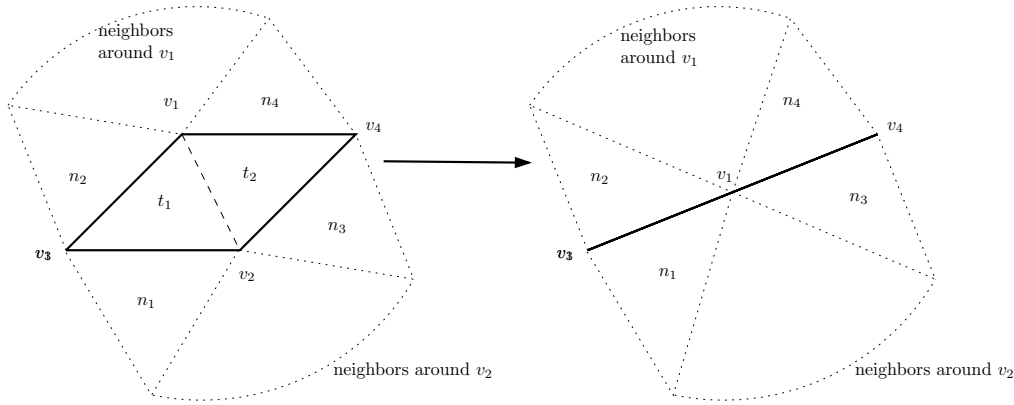


Figure 9.4: Illustration of an edge collapse deformation.

When collapsing further care must be taken to avoid spawning a baby mesh². This involves checking that e has sufficient neighbors when collapsing it.

A simple edge collapse deformation assumes that positioning the resulting vertex between v_1 and v_2 is a fair guess. In many cases, especially near sharp edges or the border, this is not the case, which makes the probability that the model is kept low. One solution to this is to allow a more advanced choice based on the local curvature and connectivity. Thus a collapse near a sharp edge or a border should preserve the edge or border if possible. Another approach is to let a vertex deformation use the image data to position the vertex after the collapse. This way merging two deformations into one may improve the chance of them both, the collapse, since the vertex is positioned better and the vertex deformation since it is used on a vertex that is very likely to be positioned wrong.

9.3.6 Vertex Split

The split deformation is used to increase the quality of the mesh. The operation on the mesh can be seen as the opposite of collapsing an edge. To do a split deformation in a deterministic way, it is necessary to specify where the new edge e should be placed and therefore also the possible two new triangles. One way to do this is to specify the last vertex in the two new triangles t_1 and t_2 , ie. v_3

²A small mesh connected to the *mother mesh* by only a single vertex, or two edges connecting the same two vertices.

and v_4 , see 9.5. Another way of describing this is that one should specify the two edges from v_3 to v_1 and from v_1 to v_4 , where the two new triangles should be placed. If either v_3 or v_4 are not existing, for example if v_1 is a border vertex, then the corresponding triangle should not be inserted. The following must be done to ensure the correctness of the mesh:

- The neighboring triangles (n_1, n_2, n_3 and n_4) should be obtained from the 3 vertices, if the corresponding vertex is existing using the mesh operations.
- Get all the triangles around v_1 in the current mesh using $[v_1]$. Divide them into two groups, one group for all the triangles lying on one side of the edges v_3 to v_1 and v_1 to v_4 . These respectively are the ones that will have v_1 and the new vertex v_2 as a corner. There is no simple operation doing this, however it can be done by walking on the mesh around v_1 picking up the triangles, and use n_i to determine which vertex will be their new owner.
- Create the new vertex v_2 and the new triangles t_1 and t_2 if the corresponding vertices exists.
- Set the neighborhood between t_1, t_2, n_1, n_2, n_3 and n_4 if they exist.
- Position v_1 and v_2 . The position is given by calculating the average of the endpoints of the n triangles if they exist. The new position is then $\frac{2}{3}$ from the old v_1 to this point.
- Swap v_1 with v_2 in the group of triangles corresponding to v_2 . This is not necessary to do for the other group, since they already know their corner v_1 .
- Set the ownership of v_1 and v_2 to be one of the new triangles.

As with collapse some of the n triangles can be one triangle, however no special care should be taken. Since the split operation is expanding the mesh, it can not spawn baby meshes. Further more the split deformation could benefit of being merged together with two vertex deformations as well, using the same argumentation as with the collapse deformation. On the other hand the deformation of existing vertices could be avoided by introducing another deformation, the *triangle split*. Like the vertex split, it will refine the mesh, however it will not move any existing vertices, which may render the deformation improbable in the probabilistic framework. The idea is simply to insert a new point into the center of a triangle and thus divide the triangle into three new triangles.

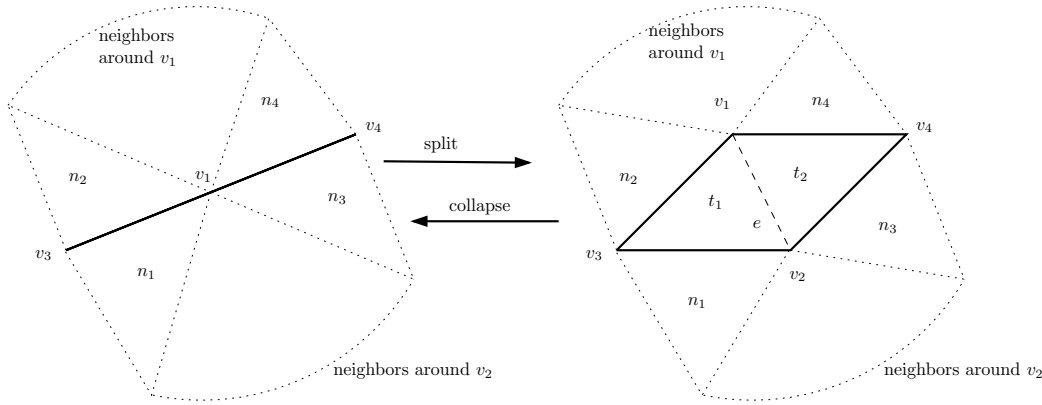


Figure 9.5: Illustration of a split deformation.

9.4 Rollback of Deformations

When deforming the mesh, which is done when sampling from the proposal distribution, the mesh itself is changed and the old mesh is thus lost. If the proposal is discarded, then the old mesh is to be used, and thus it should either be possible to *roll back* the deformations or to have a local copy of the old mesh. Copying the mesh back and forth can be a demanding task, if the mesh is large and should therefore be avoided. By recording the deformations of the mesh into a deformation queue, the deformations can be rolled back in such a way that the old mesh is obtained.

To roll back a deformation, an *inverse deformation* is needed for every deformation. For the shape changing deformations this is simply to deform the involved vertices back again. An edge can only be swapped between two different states and thus edge swap is its own inverse deformation. As already mentioned the edge collapse and the edge split can be seen as inverse operations on the mesh. In fact, the deformations have been designed specifically to be inverting operations of each other. Thus the mesh can be considered to be the exact same after a deformation and rolling it back. This can then be extended to series of deformations, since each rollback will bring the mesh to be in the same state as the previous deformation record expects.

It should be mentioned that numerical errors can occur, and that the triangles and vertices involved need not be the exact same. For example, if they have been removed from the mesh in the deformations and another triangle is added in the rollback. The flow, when deforming a mesh, and rolling back is shown in figure 9.6. If the

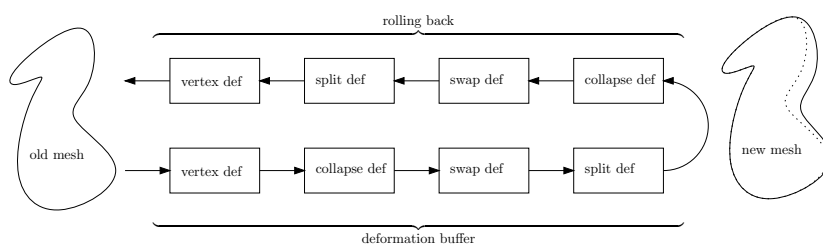


Figure 9.6: Figure of the flow, when rolling back the deformation buffer. Every deformation is matched by its inverse.

proposal is accepted, the deformation buffer can be emptied, as the information is no longer needed.

Vogiatzis does not discuss the actual implementation of the mesh he uses. It is however assumed that he uses some means of saving resources as it would otherwise become a bottleneck for larger meshes.

9.5 Topological Considerations

One of the drawbacks of using a mesh to represent the surface is that it is difficult to change the topology of a mesh. The normal way to explain topological changes in a surface is that it would require a tearing or gluing (or both) of the mesh. In short this means that if the topology of the mesh is static, then the mesh can not be divided, wholes can not emerge, and the mesh can not merge together with another mesh, or itself.

Many real world objects, like a cup, have holes in them, and thus its surface can not be derived from a simple sphere mesh. In [10] Attene and Spagnuolo proposes a method for punching wholes in a closed mesh. It however complicates the algorithm considerably, and is therefore not discussed further in this thesis.

Implementation

To be able to achieve results within a satisfactory time frame, an implementation of an algorithm must have a certain performance. On the other hand, this thesis is not an investigation into how fast this algorithm can be implemented, and thus the emphasis put into the performance has to be weighted up against the actual gain.

One of the properties of the approach chosen to do surface estimation is the ability of it to be performed by modern highly optimized graphics hardware. This is thus a good start to achieve good performance. As an interface to the graphics hardware the Open Graphics Library (OpenGL) have been chosen as it is open, fast, well documented and already widely used in the research community. C++ has been chosen as the implementation language, as it is fast, interfaces OpenGL nicely and gives the choice of object oriented design. Further more the free Borland C++ compiler has been chosen for its platform independency. The current implementation of the program is done for Microsoft Windows, but a port to fx linux should be fairly simple.

In this chapter the implementation of the basic algorithm will be described. It is divided into two sections, the first documenting the general program structure used, major implementation issues, and the general flow. The intention is not to provide an in depth walk-through of the source code, but rather to give the reader an idea of how the program is structured. The second section documenting

the parts of the implementation related to OpenGL. Here emphasis has been put on describing the important uses of the graphics hardware to solve some of the computational heavy parts of the algorithm, i.e. GPGPU. To be able to distinguish program elements from the concepts described elsewhere, these elements will be printed using the 'typewriter font'.

10.1 Program Structure

There are two main tasks that should be handled by such an implementation. The first, and most important, is of course the surface estimation, however it would be beneficial to be able to test the algorithm using synthetic and perfect input data, and therefore the other task is to be able to capture such datasets. The program is built as two executable files, one for capturing synthetic data of textured models, and one for performing the surface estimation, however they share many of the same elements. In the following these elements and how they function together will be described. Please refer to figure 10.1 giving an overview of the program structure and its elements.

main , the main executable file for doing surface estimation. It interfaces and initializes OpenGL, and creates a **Scene** to load and operate on the input data given as a **job description** file. It provides a set of keyboard shortcuts to set parameters and options and perform actions on the **Scene**, f.x. starting the surface estimation algorithm using the given input data. It thus handles the human interface to the program. A user guide is included in Appendix A, describing how to install and use the program with its various shortcuts.

capture , the executable for capturing synthetic datasets. The input data is given as either a **.ms3D** file or a **.X3D** file. The input file is used to create a **Model**, that can be displayed. The user can then manually choose an arbitrary number of views, from which images are captured, and camera parameters saved into a **job description**. This **job description** can then be executed using **main.exe**.

Mesh , handles, represents and provides deformation functionality of a triangular mesh. The mesh is implemented using triangles and vertices as described in Chapter 9 and is described in more detail in the next section.

Camera , contains image data and camera parameters for one of the input images. It provides functionality to project the image, take snapshots of the **Scene**, and finally compare its image data to other images or snapshots

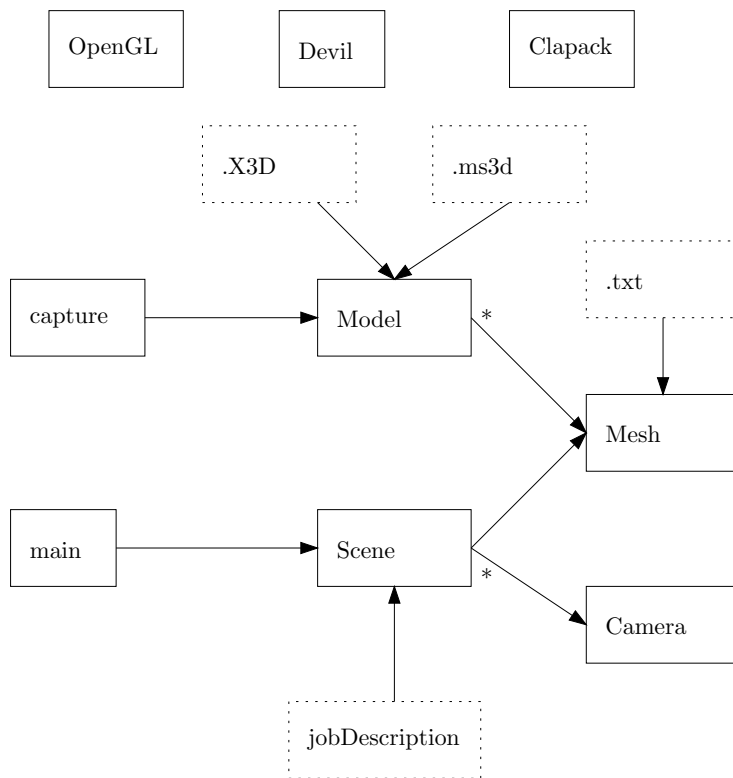


Figure 10.1: Overview of the program structure showing the major elements.

using one of the similarity measures. The **Camera** is implemented with heavy use of GPGPU, and thus contains most of the functionality related to OpenGL.

Scene , represents a scene that holds a **Mesh** and a set of **Cameras**. The **Scene** can perform a fit of the **Mesh** to the set of **Cameras** using the algorithm described in this thesis, with respect to a set of options describing how the fit should be done.

Model , represents a 3D model, that is a set of **Mesh**'s each having a mesh appearance connected to it. A **Model** can be loaded from either an **.X3D** file or from a **.ms3d** file, and can be used to save a **Mesh** to an **.X3D** file. The mesh appearance describes the material and the texture of a mesh. This division of the mesh data and the mesh appearance has been chosen, to speed up the algorithm where no material or texture of the mesh is used.

job description , contains data of a *job*. A job is a set of images, with corresponding camera parameters, an initial mesh, and possibly a set of options and parameters describing how the algorithm should run. The **job description** file is a plain text file where lines starting with '#' are ignored. An explanation of the contents of a **job description** can be found in Appendix A.3.1.

.X3D , is an open ISO standard for representing real time 3D graphics. X3D is the successor to VRML¹, and gives good functionality to share 3D objects and scenes. It has therefore been chosen as the output data type of the estimated surface. More information on the standard can be found in [5].

.ms3d , is a simple file type containing *Milkshape* textured 3D models. The accompanying program can transform a wide variety of different 3D data types into **.ms3d**, and has thus been chosen to be able to load many different models into the program. See [21], for more information.

Devil , is an open source library of many image loading and saving routines. It is used as the interface to loading and saving images. It enables the program to use a wide range of input image types. It should be noted that only a few of these supports an alpha channel needed to specify the image domain Ω_i , if used in the algorithm. Refer to the web page at [1] for download and documentation.

Clapack , stands for C - Linear Algebra PACKage. It is a C translation of a fortran library for solving simultaneous linear equations. In the program it is used to perform some of the more advanced mathematical operations.

¹Virtual Reality Modelling Language, see [3] for further information

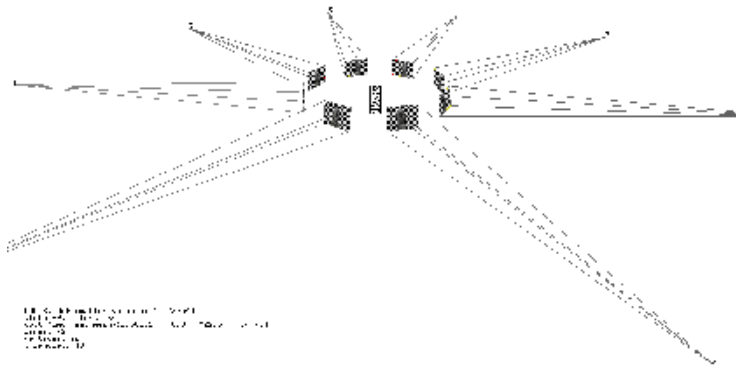


Figure 10.2: A screen shot from the program showing 8 narrow angled **Camera**'s displaying their input image, and an initial mesh.

It has been necessary to modify it slightly to allow a compilation with the Borland compiler used.

This ends the description of the elements of the program. A screen shot example of the implementation can be seen in figure 10.2.

10.2 Interfacing OpenGL

The graphics hardware in a modern system is interfaced through a driver provided by the graphics card vendor. In OpenGL this is done by series of function calls, that alters the current state of the graphics driver. Whenever rendering, this state defines the parameters describing how it should be done. Interfacing OpenGL can thus be boiled down to setting up the state, and rendering primitives.

The main task that we would like the graphics hardware to handle is the fol-

lowing:

1. Make a **Camera** project its image data.
2. Render the **Mesh** onto a framebuffer, using the current projections.
3. Evaluate the view from a **Camera**.

Most graphics hardware and the associated drivers are designed to handle computer graphics. This means that they are optimized for creating contents for the screen, which is usually in the rgba color format². In image analysis, gray scale images are very often used as they are faster to handle and represents most of the information of the corresponding color image. The design of the graphics hardware makes the extra computational cost for handling colors almost negligible. Therefore colors will be used wherever possible in this implementation, as they do provide some extra information.

The rest of this section describes how these tasks together with a summation of the OpenGL extensions used.

10.2.1 Used Extensions

The basic OpenGL implementation provides many standard procedures, however to fully utilize the computational power available, some of the many extensions must be used. Table 10.1 shows the extensions that is used in the implementation, and a short description of the functionality they provide. Almost any medium to high end graphics card supports these features, allowing the program to be run on a wide range of systems.

10.2.2 Projecting from a Camera

When loading a **job description** the camera parameters for the projection Π , defined in equation 7.2, are specified as a 3×4 camera matrix C . Since only linear dependencies can be described in this way, radial and tangential distortion is not taken into consideration. This distortion can however be corrected beforehand and thus ignored in the algorithm. The camera matrix is defined as

²Color with a Red, Green, Blue and an Alpha channel.

name	description
Rectangular texture	Provides rectangular textures that unlike the standard textures need not be in power of 2. Furthermore they are indexed as a double array instead of the normal $[-1..1]$.
FBO	Frame Buffer Object - Provides offscreen rendering to textures and a wider choice of input and output channels. Thus it allows the result to stay in graphics memory and avoids slow copying from the framebuffer.
NV float buffer	Makes a new type of texture representation available, that consist of floating point channels, that are not clamped to the normal $[0..1]$ range. This allows to use textures as general floating point arrays in the computations. The precision used is 32 bits. The extension is developed by NVidia ³ , but is supported in most cards today.
Shaders	Enables actual programming of the graphics hardware, through small programs called shaders. A C-like language is used, which is compiled at runtime. A shader program consists of a vertex shader working on the vertices, and a fragment shader handling the fragments of the primitives when rendering.

Table 10.1: The extension used in the implementation.

$$C = K[R|t] \quad (10.1)$$

, where R is a 3×3 rotation matrix, t , a translation vector and K the calibration matrix defined by

$$K = \underbrace{\begin{bmatrix} \frac{2}{width} & 0 & -1 \\ 0 & \frac{2}{height} & -1 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{normalizing term}} \underbrace{\begin{bmatrix} \lambda f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{normal calibration}} \quad (10.2)$$

Here f is the focal length, s is the skew, (x_0, y_0) the coordinates of the principal point in pixels and finally λ the aspect ratio of the pixels. As can be seen it has been chosen to define K in a normalized form making the domain of the image in the range $[-1..1] \times [-1..1]$. The reason for this is twofold. First it corresponds to the domain of the framebuffer in OpenGL. This makes it possible to apply it directly when capturing images. Second the dimension of the corresponding image is already given in the image data, and thus C can easily be adjusted to any image size as we shall see later.

To project the image of a **Camera** onto the **Mesh**, one must calculate the corresponding texture coordinates for vertices in the mesh. This can be calculated on the fly in OpenGL by specifying a 4×4 texture matrix T that are applied to the vertex coordinates. This corresponds to the projection defined by C , however as T has one more row than C an extra value is calculated. If setup correctly this can be used to calculate the depth from the vertex to the projecting camera. When compared to a *depth buffer* this depth can be used to determine if it is visible in the projecting camera or not. This requires that a depth buffer has been rendered beforehand, by rendering the **Mesh** to the **Camera**. To obtain maximum depth precision two clipping planes, a near, n , and a far, f , is defined covering the mesh as seen from the camera. According to the OpenGL specifications in [42], the depth can then be calculated using

$$d = \underbrace{\begin{bmatrix} 0, 0, -\frac{n+f}{f-n}, -\frac{2nf}{f-n} \end{bmatrix}}_{T_{z_c}} \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \quad (10.3)$$

, where (x_c, y_c, z_c, w_c) are a vertex in camera coordinate system. To adjust for the rotation and translation of the camera matrix, from the camera to the world

coordinate system, these are applied giving

$$T_{z_w} = T_{z_c} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Flipping z direction}} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \quad (10.4)$$

As can be seen, the z-direction are flipped to be consistent with OpenGL. We do not however want to adjust for the internal calibration of the camera as the depth has nothing to do with it. To further complicate matters, the domain of the rectangular textures are $[0..width] \times [0..height]$, which requires a transformation, that yields the final texture matrix:

$$T = \begin{bmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ & T_{z_w} & & \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix} \quad (10.5)$$

This also shows how easy the size of an image can be changed. Thus to reduce the amount of input data, the images can be down sampled without changing the camera calibration parameters in C . Another advantage of this method is that everything is handled by the OpenGL projections. Thus the different images can have arbitrary resolutions and still be used together. It is however recommendable to chose resolutions relatively close to each other, because the sampling rate in the images should be comparable to avoid super sampling.

To summarize, we setup the projection from a **Camera**, by activating a texture with the image data, and specify a texture matrix, T , to calculate texture coordinates. If a comparison with the depth buffer at the texture coordinate is positive, it is used again to lookup in the image data. The final result can either be saved in the frame buffer or used directly in a similarity measure together with the evaluating cameras image data in a shader program. This approach can further be used to project multiple cameras at the same time reducing the number of renderings, however there is a hardware dependent limit on the number of active textures and texture matrices.

10.2.3 Rendering the Mesh

OpenGL provides 3 different ways of sending triangles to the graphics hardware for rendering. The first and simplest is to specify the vertex coordinates of the triangles using function calls. This however is highly inefficient for larger meshes and is thus not an option. The second is to arrange the vertex coordinates in an array and the indices of the vertices in this array, for the corners of the triangles in another array. These arrays can now be sent to OpenGL for rendering by specifying a pointer to them using an OpenGL call. This is much faster, since only a few function calls are needed, however it requires that the triangles and vertices are aligned nicely in memory. The third method is to use *buffer objects* to store the arrays of vertices and triangle vertex indices in graphics memory. This eliminates the sending of the data for each draw, since the data is already available to the graphics hardware, however the mesh used is to be highly dynamic, and it would thus require re-specifying at least parts of the data every time the mesh is deformed.

For simplicity sake, the second solution has been chosen, even though it would be faster to use the third. If the third method was to be used it should identify the mesh operations that changes the mesh, and incorporate a repair of the OpenGL image of mesh every time it is altered.

Arranging data in arrays

Using the chosen rendering method puts some constraints on how the mesh can be arranged in the memory. As mentioned the vertex coordinates and the indices herein for the triangle corners, must be aligned in two arrays. Since these are sent to OpenGL for rendering, they cannot contain any rubbish data or holes of triangles or vertices that should not be rendered. This is not completely true, since the vertex array can contain vertices not to be drawn, however as they are sent to the graphics hardware anyway, it is desirable that this is limited.

Arranging the vertices and the triangle vertex indices into arrays could be done every time the mesh was to be rendered using a dynamic data structure for representing the mesh, fx the half edge data structure⁴. When rendering, the dynamic data structure would have to copy the data into the arrays. This however becomes a demanding task, when the mesh grows, and as such would not be a good solution.

⁴A mesh data structure where primitives are stored as circular lists of its edges, termed *half edges*. Every element of the list has a reference to the next vertex and the neighboring half edge, being a part of the neighboring primitive. See [43] for more information.

Another way to achieve this is to design the mesh in such away that the triangles and the vertices are already arranged in arrays, and preserves this property when deformed. Figure 10.3 shows the design that has been chosen for the mesh. The aligned data sent to OpenGL is the vertex array and the vertex index array describing the indices of the vertices associated to the corners of the triangles. The dynamic functionality is implemented by interleaving extra data in each vertex, and by having an extra array of triangles keeping track of the connectivity of the mesh, the vertices and the vertex indexes used. Note that interleaving data is only present in client memory and is not sent when rendering. From the figure it can be seen that each vertex knows a single owner triangle, from which a lookup in the triangle connectivity can be used to do the mesh operations.

To limit the amount of holes in the arrays, whenever a vertex or a triangle is no longer needed, it is added to a set of dead vertices or dead triangles. New vertices are taken from this list, or if there is none, a completely new triangle is created in the end of the array. This way one can keep track of the parts of the arrays used, and limit the amount of dead vertices sent. To completely avoid holes in the index array, a 'dying' triangle automatically swaps its vertex indices with the rightmost triangle.

This design allows dynamic deformations while requiring a minimum of copying and memory consumption. It preserves the dynamic functionality, while arranged in arrays and are thus always ready to be send for drawing.

10.2.4 Evaluating a View

A *view*, in this context, is defined as what can be seen from one of the **Camera**'s. Thus to calculate the objective function of some model, the view of every camera must be evaluated and the result summed. Evaluating a view is done projecting one or more of the other **Camera**'s, and take a *snap shot* from the viewpoint of the **Camera** to be evaluated. The snap shot can then be compared to the image data of the evaluating **Camera** using one of the similarity measures described in Section 7.3. Capturing such a snap shot is done similar to taking a real world photograph using the same camera. Recall that the pinhole camera model, used to define the camera parameters in Section 10.2.2, is an attempt to emulate what is actually happening in the real camera. Therefore setting the projection matrix to the camera matrix, and setting the model view matrix to an identity matrix, will project the mesh into the image like the vertices where light rays reflected through the lens. This is similar to when projecting from a **Camera**, however this time the projection matrix is set and no transformation to image domain is done. Further more, when projecting into the framebuffer, OpenGL

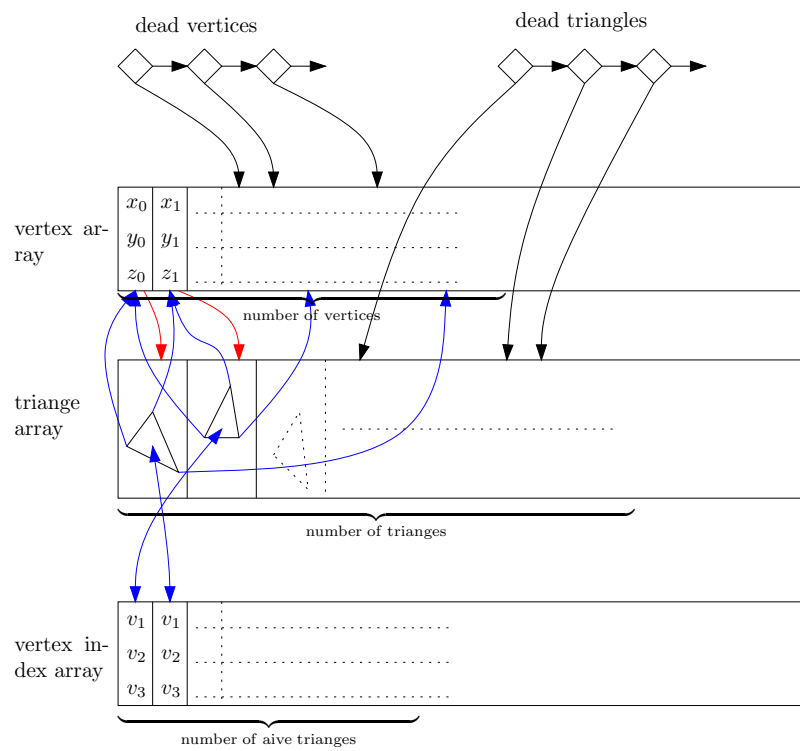


Figure 10.3: Overview of the memory design of the mesh.

can handle the depth test for visibility automatically.

As mentioned not all the **Camera**'s need to project at the same time. This gives rise to the following proposals for evaluating a view, differing in which **Camera**'s are projected and how the snap shots are collected.

pairing. This is the most idealistic and simple approach where each **Camera**'s image data is compared to a snap shot of every other **Camera** projecting. This approach is slow since the number of comparisons is $n(n - 1)$, for n input images. The amount of information mutual between two images of an object taken at opposite direction is relatively low. Thus this approach is best with a few **Camera**'s within a relatively close viewing angle.

blending. This approach is slightly faster, and the one used by Vogiatzis. Each **Camera** lets all other **Camera**'s project their data into the same frame buffer blending the result using alpha blending. This still requires $n(n - 1)$ projections, however the result is only compared n times. The tradeoff is that occlusion is only evaluated globally and thus some version of the lenticular print surface discussed in Section 7.3.1 may result.

only neighbors. This approach is even faster, by only comparing b neighbors of each **Camera**. Thus the number of projections is down to $n \times b$, and the number of comparisons can be either n if blending is used or $n \times b$ if not. If b is chosen sufficiently low (1 or 2), one could fit both the projections and the comparisons into a single shader using only a single sweep. The neighbors of each **Camera** should be chosen using the $\frac{\text{baseline}}{\text{depth}}$ ratio described in section 4.3.1. Unfortunately this interesting capturing method has not been implemented because of lack of time.

Figure 10.4 shows a sketch of how the evaluation is done for the three methods. An example of an original image, its depth buffer, the snap shot taken and a visualization of the error can be seen in 10.6. The error is calculated using SSE, as it is more friendly for visualization. The evaluation of this and the evaluation of correlation is described in the two next sections.

Evaluating SSE Using Shaders

Using shaders, the SSE of a pair of images can be calculated in a simple and fast way. The direct approach is of course to acquire the snap shot from the graphics card to local memory and let the CPU handle the calculations. This however is slow since the snap shot must then be transported over the bandwidth limited

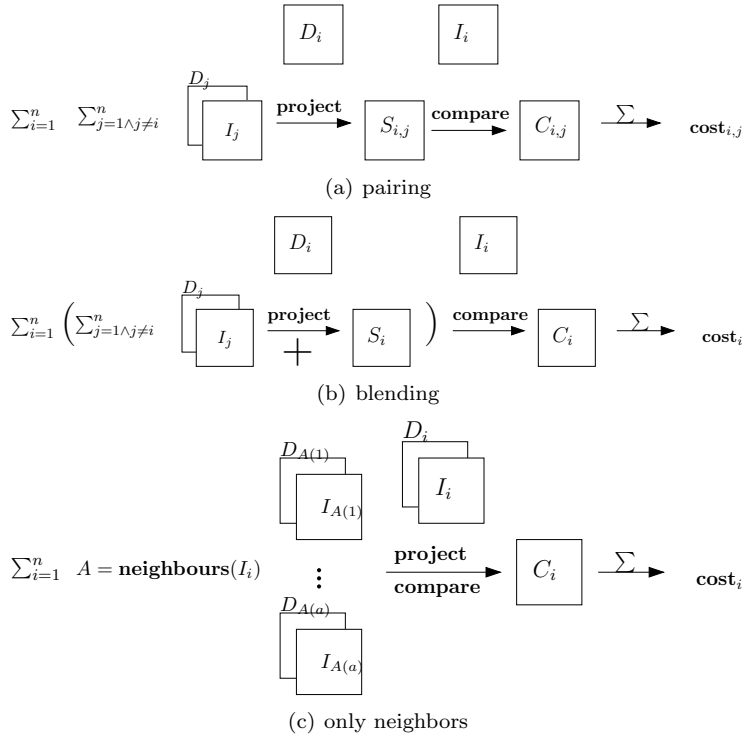


Figure 10.4: A sketch of the evaluation of the three proposed methods for evaluating a view.

graphics bus. Instead the parallelism in the GPU can be exploited. A shader is setup to calculate the squared error of every set of pixels if both are visible, and save the result in the red color channel. At the same time the other 3 color channels are used to indicate the type of the pixel (normal, occluded, alpha), by storing 1.0 in the channel associated to the type.

After this the resulting texture is used as input for a summing algorithm using another shader. For every iteration 4 pixels are summed channel-wise and stored in a single pixel. By swapping input and output texture each iteration, the values in the texture gets summed to a single pixel. This is illustrated in figure 10.5, where the texture is summed using only 5 iterations for summing the 32×32 colors. Reading the single pixel yields no problem for the bus. The result is 4 values, where the first is the SSE and the three others are counts of the type of pixels, which can be used in a more sophisticated cost evaluation of the image pair.

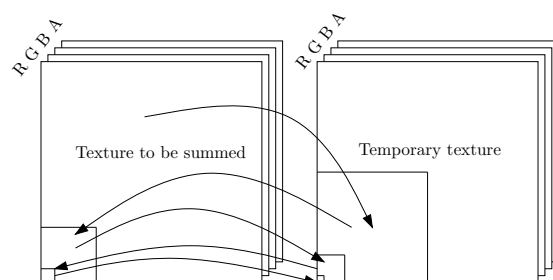


Figure 10.5: Figure of the summing algorithm using shaders.

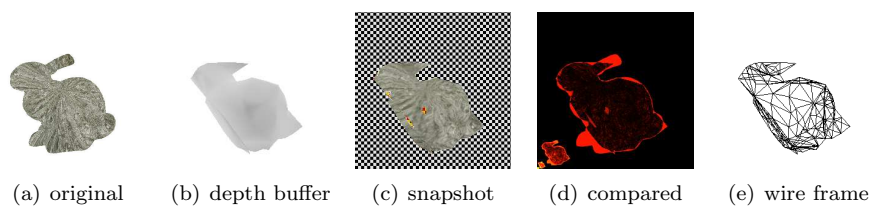


Figure 10.6: Example of the buffers used evaluating a view. The black and white checkerboard pattern in the snapshot means that the alpha channel is 0, while the red and yellow checkerboard pattern is occluded areas. The small figure of the bunny in the lower left corner of the comparison buffer is the effect of the summation, that reuses this texture to save space.

Evaluating Correlation Using Shaders

Calculating the correlation is not as simple as the SSE, as it requires a minimum of two sweeps of two different shaders and a sum of the values like in SSE. Further more, to avoid using too many textures only the correlation at a gray scale level is calculated. The first sweep calculates the sum of the gray scale colors in the visible pixels for each of the two images. These sums takes up two channels, thus the third and fourth are free for a pixel count and an occlusion count. The texture is summed using the summing algorithm, and the pixel count is used to calculate the mean of the two images. This is used as input to another shader calculating the squared value of the gray scale colors in the two images, together with $(x - \bar{x})(y - \bar{y})$, where x and y are the gray scale colors. The fourth channel is used to indicate if the pixel is an alpha pixel. Another sum sweep is used, and now the correlation can be calculated as given in equation 7.11. Like before the resulting correlation is accompanied by a count of the type of pixels involved.

10.3 Pitfalls

This section presents some of the pitfalls, that may degrade the performance of the algorithm if not taken care of. It is not an exhaustive listing of problems encountered, but a short list of the most important.

10.3.1 A Good Pseudo Random Number Generator

Many of the procedures in the algorithm are based on some degree of randomness. True random numbers is only available in special hardware, thus some *pseudo random number generator* (PRNG) must be used. Unfortunately most standard PRNG's, that comes with the compiler or the operating system, only have a resolution of 2^{15} , that is, a call to the PRNG gives a pseudo random number between $0 \dots 32767$. Further more poor PRNG's tends to repeat themselves in relatively short periods. The number of calls to the PRNG in an implementation of this algorithm is high, and it is thus of uttermost importance that the PRNG is of good statistical quality (close to uniform distribution), have high resolution (at least 31 bits) and a long period. Two such generators are the *Mersenne Twister* and *r250*, of which the last is used in this thesis. The *r250* PRNG has the same resolution as the machine architecture, a period of 2^{250} , hence the name, and a good statistical nature. More information can be obtained in [38].

10.3.2 Projective Errors

One problem, that surprisingly is not mentioned in any of article concerning projections mentioned in this thesis, is that projective errors occur, when interpolating the reprojected texture coordinates. When rendering a snap shot, the texture coordinates of the vertices in the projecting images are calculated using T . When coloring the fragment of a primitive, the texture coordinates are interpolated in the domain of the camera taking the snap shot. Interpolating the projected vertices represents no problem, however the texture coordinates are reprojected from another domain, giving rise to a projective error. This is illustrated in two dimensions in figure 10.3.2.

There is no easy solution to this problem, since it would require a reprojection of every fragment coordinate onto the mesh and project them into the projecting **Camera** to get the correct coordinates. A subdivision of the mesh minimizes the problem, but to fully avoid it would require a subdivision corresponding to the

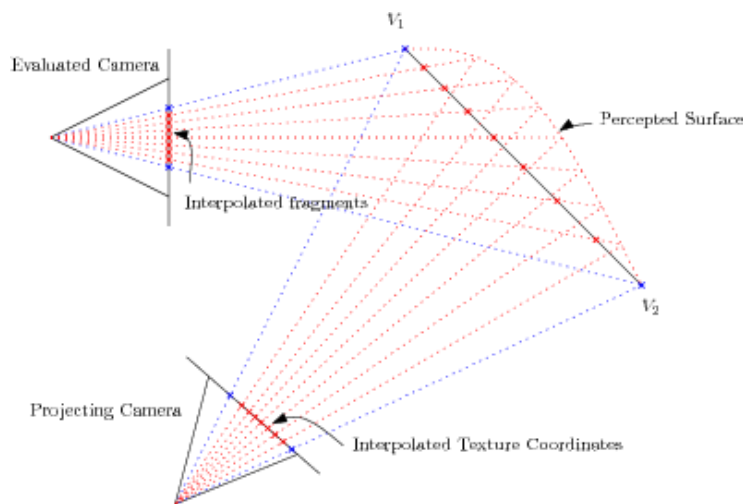


Figure 10.7: Figure showing the projective error due to the interpolation in texture space and not in world space. The red curve shows how the line would be perceived after the projection.

resolution in the textures, i.e. approximately a triangle per pixel. The problem however is only significant when having large (compared to the resolution in the images) planar surfaces. The resulting error would then draw the model towards a finer grained mesh than actually required by the ground truth surface, which is not destructive to the algorithm. An example showing the original texture, the projective error and the projective error corrected by using a finer grained mesh can be seen in figure 10.8.

10.3.3 Depth Errors

A common problem in computer graphics is to use the depth buffer directly for comparisons to determine if some object is visible (used for example in shadow mapped light). First of all, if the depth stored in the depth buffer is of a lower resolution than the depth used in the comparison, or if the two depths are not calculated in the exact same way, giving rise to numerical errors, then the comparison may maliciously fail. This simply requires good programming practise to solve. Another harder problem comes from the quantized nature of the depth buffer. The values in the depth buffer are only samples of the real depth of the scene. When comparing to the depth buffer at sub-pixel accuracy, as is the case when checking visibility, then only an interpolated depth is available. Figure 10.3.3

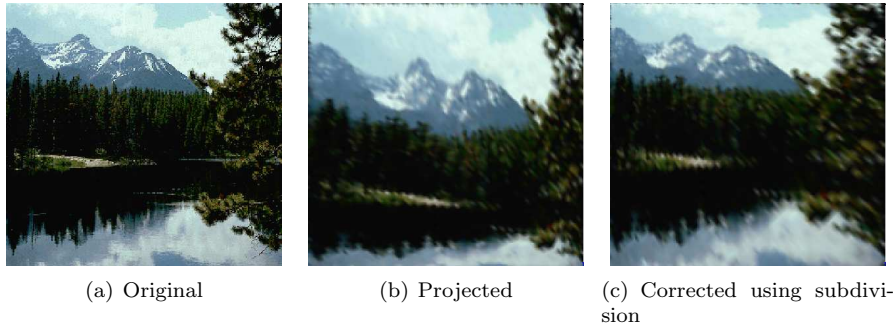


Figure 10.8: An example of the projective error. From left, is the original texture, a square made of 2 triangles on which the texture is projected from a **Camera** at a skew angle, and right the same square subdivided sufficiently to minimize the projective error. The bad quality of the projections is an unavoidable effect of the reprojection.

shows how this will fail, when the mesh is not simple.

This problem can somewhat be minimized by adjusting the near and far clipping plane to match the actual spatial span of the mesh, but the error is still notable. Many other solutions have been proposed, some involving filter operations on the depth buffer. To keep the visibility check fast, a simple solution is chosen where a pixel is accepted as being visible if it is within a certain range of the depth. Making the range too large, however, will accept otherwise occluded pixels, and thus the range must be chosen with care.

10.4 Discussion

This section will discuss some issues of the implementation not fitting in elsewhere.

10.4.1 Limitations

OpenGL is designed to be dynamic, both as a programmer interface, and as a hardware implementor interface. To accommodate for the ever changing hardware changes, it has some build in hardware dependent limitations. Thus there is a maximum of how large images can be used, and how many. As new graphics

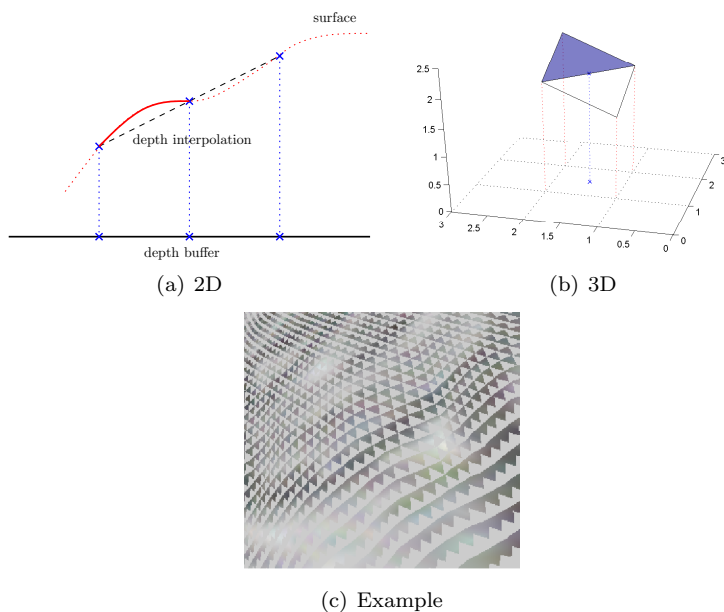


Figure 10.9: Figure illustrating the depth error in 10.9(a) in 2 dimensions, 10.9(b) 3 dimensions and 10.9(c) an example from the program, where the light gray is occluded pixels.

hardware is developed it is expected that these limits will go up.

As an example, the computer used in this thesis has a maximum texture size of 4096×4096 and can only have 8 sets of textures and depth textures at the same time. This should be plentifully for the basic implementation, as most digital images today have lower resolution, however it makes it impossible to have more than 8 Cameras projecting at the same time.

10.4.2 Memory Considerations

While images can be compressed considerably when stored on a hard drive, they must be in an uncompressed form, if used for fast rendering. This section will try to make an estimate on the amount of video memory used for the various textures involved. It is clear that the amount of video memory is a limited resource. As an example the computer used in this thesis for testing

the implementation has 256 Mb of video memory. The implementation uses 4 textures for each **Camera**, one for the image data, one for the depth buffer, one for the captured snapshots and finally one for the error calculations. Some of these textures could be shared between the **Camera**'s, however they can be used to visualize the flow of the convergence. If b_{tex} is the number of bytes used for a texture the following estimation of the memory consumption can be written

$$mem = w \times h \times (n \times (b_{img} + b_{snap} + b_{depth} + b_{err}) + b_{gen}) \quad (10.6)$$

, where w and h are the width and height of the input images, assumed to be the same. b_{gen} is a general shared texture used i.a. the summing algorithm. The precession used in the implementation are 4 bytes pr color channel, except the input images, where 2 bytes are used. Using equation 10.6 on an example with eight 512×512 **Camera**'s uses approximately 70 Mb. Thus it would not be necessary to swap the image data back and forth between client memory and graphics memory, which would be catastrophic for the performance.

10.4.3 Choosing Constants

Throughout the description of the basic algorithm many constants controlling the convergence has been introduced. Table 10.2 shows these constants and the default values chosen. Some of these constants overlap in terms of controlling the algorithm, however they have been chosen this way to clarify their use.

The weighing constants controlling the proposal distribution are at default chosen to be uniform. It is however very likely that some of the deformations will perform better than others. Therefore an automatic adjustment has been implemented, based on the performance of the deformations a number of proposals back. Thus if having a model with far too many vertices to represent the structure, deformations removing vertices are likely to perform well. Therefore they should be allowed to continue at a high rate, while other deformations are chosen less. To ensure that all deformations are used, a lower limit to the auto-adjustment are set.

Name	Description	Value
T_0	The initial temperature when using simulated annealing	0.4
pixel penalty	The weight of the cost when comparing pixels.	2
vertex penalty	The cost of each vertex in the mesh	0.3
angular penalty	The weight of the smoothness term.	1.0
alpha penalty	The cost of every alpha pixel. Should be slightly higher than the expected mean pixel error.	0.7
occlusion penalty	The cost of an occluded pixel. Should be lower than the alpha penalty, but relatively high compared to the expected mean pixel error.	0.6
δ -modifier	Adjusts the δ -value before the effect of simulated annealing. Thus determines the average number of pixels a deformation should take.	2.0
deformation weights	The weights of the deformations in the selection of the proposal distribution. A uniform selection has been chosen as default.	1.0

Table 10.2: The constants used in the implementation together with a small description and the default value.

Datasets

5 different datasets has been chosen together with the synthetical datasets produced by `capture.exe`. Emphasis has been put on choosing different datasets, with different attributes. Unfortunately a dataset showing specular highlights from a shiny surface has not been available. A listing of the datasets are:

- Synthetic
 - The Box
 - The Bunny
- The Cactus
- The Gargoyle
- The Pot
- The Face
- The Masters Lodge

These names will be used as a reference to the datasets when testing. Please refer to the **Dataset/** directory provided on the CD. All datasets are accompanied

by a coarse mesh, either hand annotated using Matlab, or included from the source. Further more some of the datasets have been used to produce cropped versions of the input images only showing the actual object. In the following the datasets, their origin and their attributes will be presented.

11.1 Synthetic Datasets

Synthetic datasets are chosen in order to provide images and camera parameters sampled from the same environment as the algorithm will be tested in. Thus the images are subject to no noise except that of numerical errors, and the associated **Camera**'s project the light rays in the exact same way as when the images were captured. Furthermore global illumination is assumed, and thus the material will act under the Lambertian assumption. Using OpenGL to capture the images it is also possible to include the ground truth alpha channel of the scene in the images, and thus simulate a perfect cropping. Two 3D models are used, a simple and detailed.

11.1.1 The Box

The first model is a simple box textured using a landscape image. The box is one of the most simple real 3D structures possible, and is thus used to test that the basic functionality of the implementation. An example dataset of the box with two cameras can be seen in figure 11.1.

11.1.2 The Bunny

The second model used for a dataset is the well known *Stanford Bunny*, which can be seen in figure 11.2. The bunny presents a more detailed surface, with different levels of details. The model is textureless, and thus a texture must be applied. It is expected that the texture used has a high influence on the amount of 3D information that can be extracted from the input images. The images are captured using ambient lighting and thus a uniform texture would result in images which only contains information of the outline of the object. The texture chosen can be seen in figure 11.3 together with an image of the textured bunny. This texture has been chosen for no specific reason, except that it is a real world texture, and contains both large and small details.

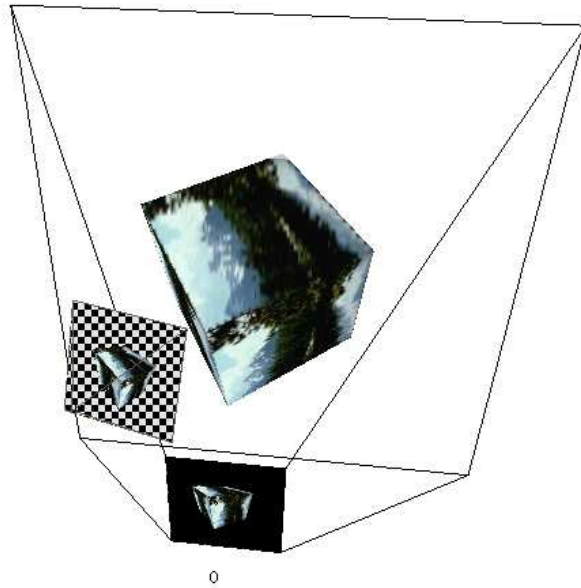


Figure 11.1: Simple example of a dataset of the Box.

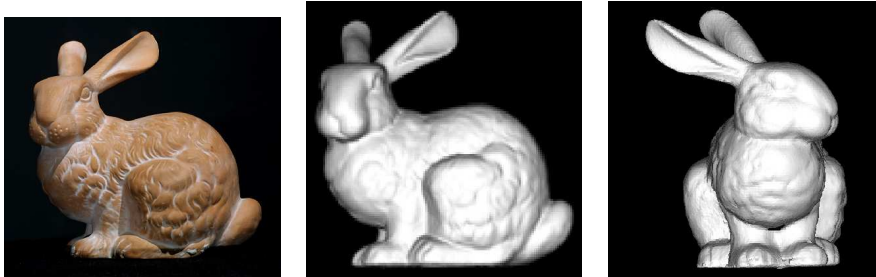


Figure 11.2: A real image of the bunny, and two rendered images of the the model.

11.2 The Cactus

This dataset and the next, the *Gargoyle* have been obtained at [40]. They have been widely used in a great deal of the articles concerning surface estimation. The dataset consists of 30 images with a resolution of 768×484 and are in the



Figure 11.3: The left image is the texture used in this text, and the right is textured Stanford bunny.

tif format. They have been obtained using a robot arm making a controlled circular movement around the object. This may explain why they are upside down. The intrinsic and extrinsic parameters are given in the *Vista* format, for more information see [48]. A small Matlab program has been implemented to convert the Vista format into the one used in the `job description`. Image 0, 1 and 2 can be seen together with the annotated mesh and a full setup of all cameras in the scene in figure 11.4.

11.3 The Gargoyle

This dataset has been obtained from the same source and in the same way as the cactus dataset. It consists of 16 images of a gargoyle, each having a resolution of 719×485 . As seen in figure 11.5 the gargoyle is almost in gray, contains a texture and many small details. The background is alternately black and white paper, which should make it easier for an algorithm not to use the background. An interesting detail is that the gargoyle has two holes under the arm, which is clearly visible in some of the input images.

11.4 The Pot

The Pot dataset is a nice painting work by one of my supervisors, Henrik Aanæs. It is a pot with a radius of about 30 cm, painted in a black and white camouflage pattern. It consists of 5 images with a resolution of 512×512 . The intrinsic and extrinsic parameters of the camera are given in camera matrices suited for the Matlab image package. These have been converted into the domain used in

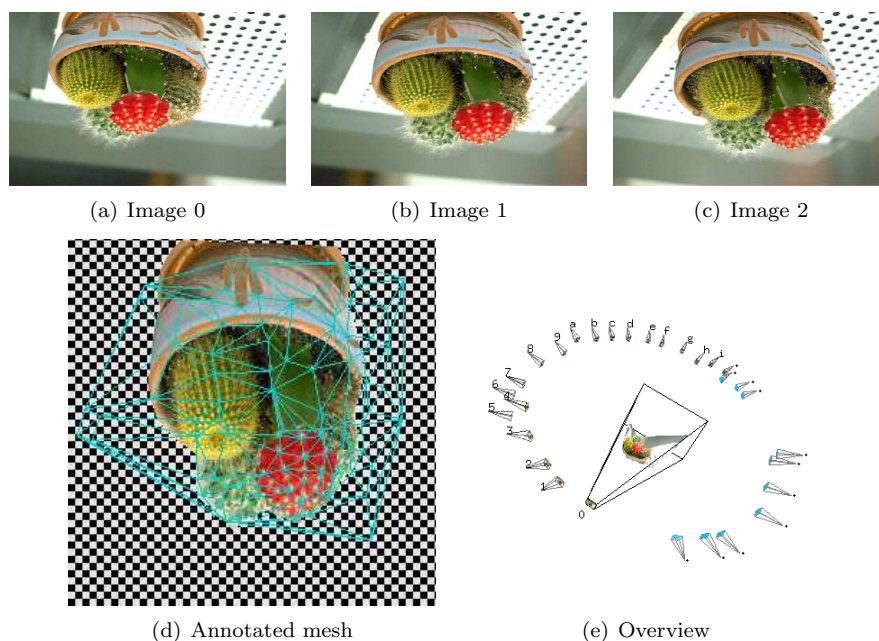


Figure 11.4: Image 0, 1 and 2 of the cactus dataset, the annotated mesh and an overview of the distribution of the Camera's.

the program. The object in this dataset is fairly simple, see figure 11.6, however the illumination level is quite different, and the checkerboard pattern on the baseplate can present a challenge to a surface estimation algorithm, since the pattern can be shifted.

11.5 The Face

The face dataset was generously donated by Kenneth Møller, taken with the setup used in his master thesis, see [44]. The images are taken in pairs, and later rectified to prepare for a stereo vision algorithm, see figure 11.7. The dataset contains 13 such pairs, each with a resolution of 1024×768 , however they are only calibrated in pairs and thus only a single pair can be used simultaneously. As will be discussed later, living tissue presents a large challenge when estimating the surface.

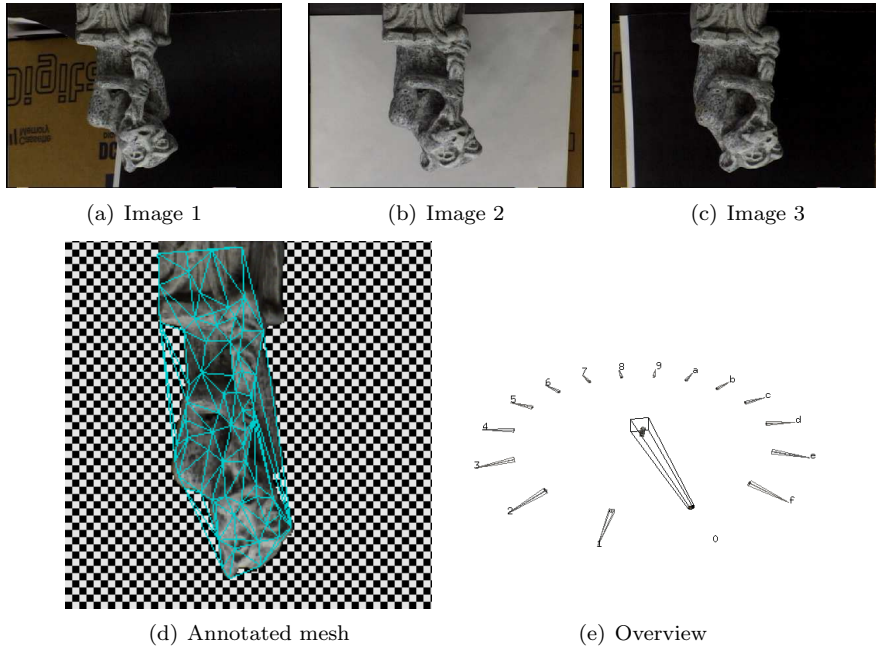


Figure 11.5: Image 1, 2 and 3 of the gargoyle dataset together with the hand annotated mesh and an screen shot from the program showing the setup of the scene.

11.6 Masters Lodge

This dataset is of architectural nature, like the one used in Vogiatzis' evaluation of his implementation. It was obtained from [6], and is popular in the literature as a dataset. The scene contains an inner corner of a building with a tower and an extruding lodge (see figure 11.8), hence the name. Vogiatzis claims that the algorithm is well suited for architectural datasets, as they are largely planar and can be represented using relatively few vertices.

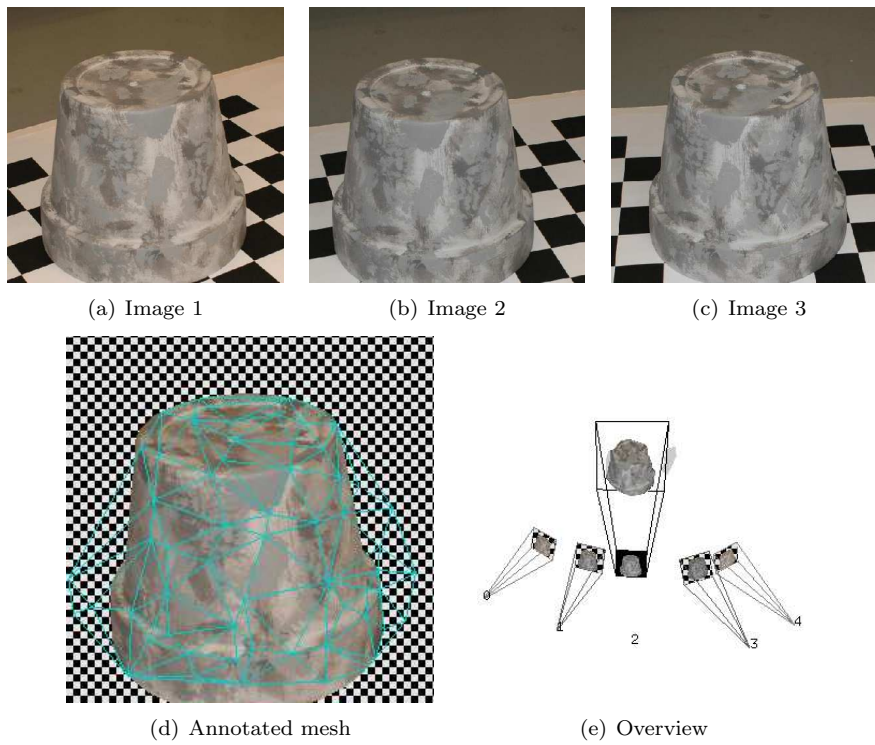


Figure 11.6: Image 1, 2 and 3 of the pot dataset, an overview of the composition of the scene and the annotated mesh.

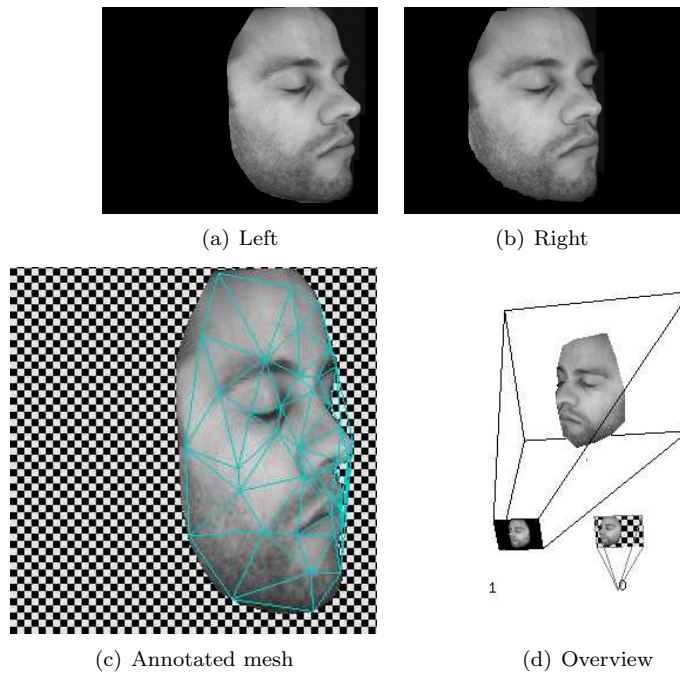


Figure 11.7: The left and right image of the face dataset. Below the annotated mesh are shown together with a screen shot showing the setup.

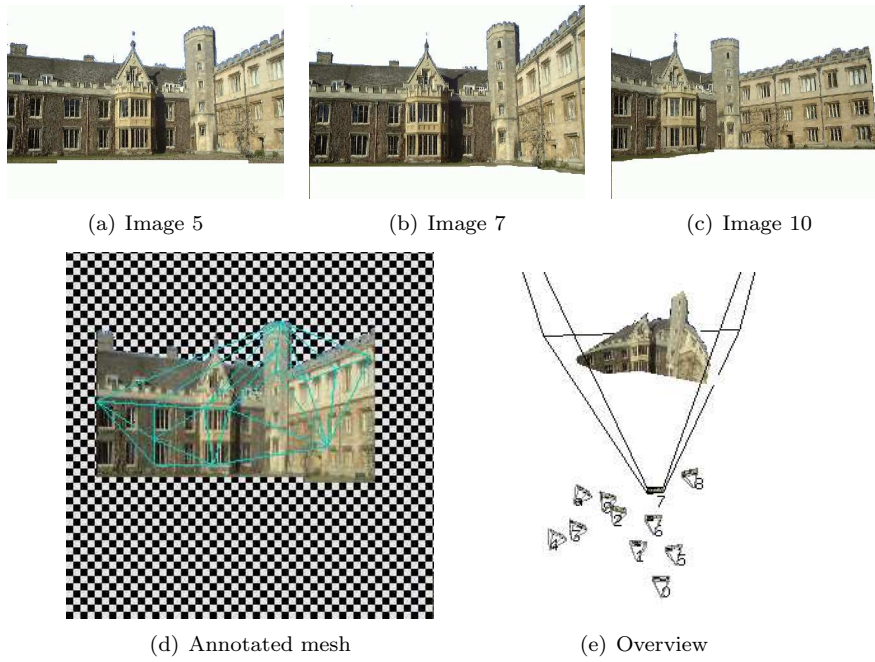


Figure 11.8: Image 5, 7 and 10 of the Masters Lodge dataset, the setup and the annotated mesh.

Results

The purpose of this chapter is to present the results of the implementation run with the datasets presented in the last chapter. As it is difficult to visualize the resulting 3D models on paper, the reader is encouraged to supplement the images provided here with the corresponding X3D models stored on the CD accompanying this document. References to the relevant models will be given in the context, while more information about the CD, and how to open X3D files can be found in Appendix B.

The intention of this testing is the following:

- To clarify that the basic algorithm operates as expected.
- To evaluate on the performance, both convergence wise and of the resulting quality of the 3D models.
- To compare some of the possible implementation choices described in Chapter 7-10, and where adequate, use the results of Vogiatzis as a reference.

This will be achieved, by first using synthetical datasets to evaluate the foundation, and later, in increasing order of difficulty, the real world datasets. For

clearness, the test results are arranged using the datasets as a basis. To show the spectra of objects to which this algorithm can be applied, the implementation choices are evaluated using different datasets. Where appropriate, datasets enhancing the differences in these choices are used. To summarize, these are:

- Image metric using Correlation or SSE.
- Deformation Strategies, both the 6 proposed by Vogiatzis, the normal deformation strategy and the triangle divide strategy.
- Simulated annealing / Static algorithm.
- Bayes selection / Best choice selection.
- Capturing using Blending or Pairing.
- Annealing schedules.
- Using auto-adjusting constants or not.

12.1 Synthetic results

Using a synthetical dataset, one can test how well the implemented algorithm performs under perfect conditions. Two different models are used to prepare datasets, a simple box, and the famous Stanford bunny.

12.1.1 The Box

To test the basic workings of the algorithm, two 256×256 images of a simple textured box is captured. The initial mesh is set to be a distorted version of 384 vertices uniformly distributed over the ground truth box, see figure 12.1 for a screen shot from the program showing the initial composition of the scene. The input parameters are set as closely as possible to that of the same test by Vogiatzis, however an occlusion penalty has been added to avoid a model collapse as described in 7.3.1. It should be mentioned that such a test without occlusion penalty has been done, however as the result is simply empty buffers it has not been included here. Figure 12.3 shows the buffers as seen before and after convergence, which took approximately 1000 iterations, 4000 proposals and less than a minute. It clearly shows that the final model resembles the input images much better than the initial, and how the mesh is greatly simplified. The SSE error buffer still shows errors, however as we shall see later this is unavoidable.

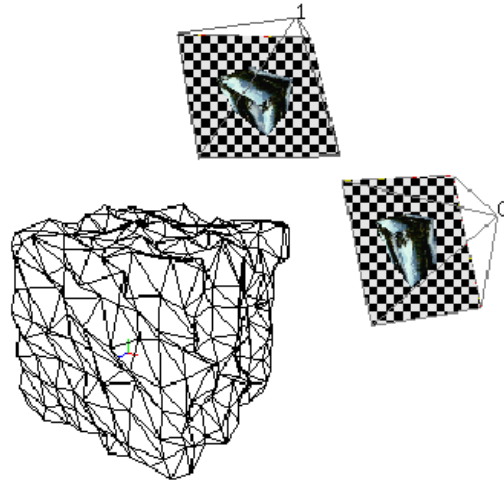


Figure 12.1: Screen shot of the initial composition of scene in the simple box test.

In figure 12.2, the convergence of the objective function is shown as a plot of the deformations distinguished using different colors for the different strategies. As expected the first part of the convergence is dominated by edge collapse deformations, while the second part has more spatially adjusting deformations.

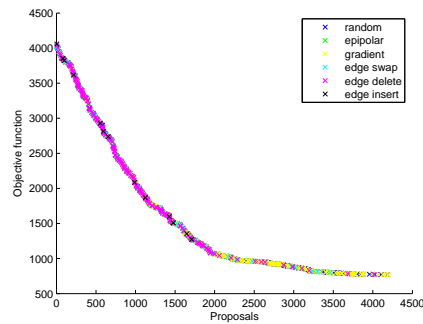


Figure 12.2: The objective function versus the number of iteration used for the Box test.

The resulting model contains 20 vertices and 22 triangles, which is more than

the 7 vertices and 6 triangles needed to represent the surface as viewed from the two cameras. This is partly a product of the fact that collapsing more vertices together would move one of the border vertices producing a large error in the images as mentioned in section 9.3.5, and partly because of the projective error discussed in section 10.3.2.

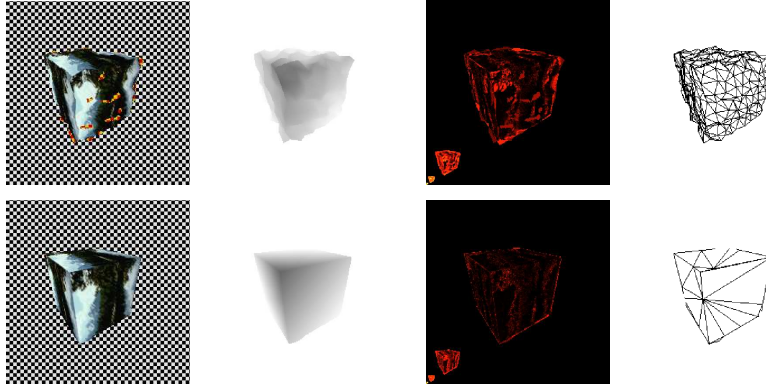


Figure 12.3: The first row shows the buffers of the initial situation. The second row shows the result after convergence. The resulting model can be found in `Box/test1/model1.x3d`.

To ensure that the later of these error sources are in fact influencing the result, the image metric of the ground truth box has been measured in different mesh resolutions starting from the minimum of 6 triangles. The result is shown in figure 12.4, which strengthens the projective error theory. Subdivision of the mesh greatly reduces the image cost, until some limit is reached. The dashed red line shows the metric when using the adaptive subdivision technique proposed in section 10.3.2. The result of this technique is very close to what can be achieved by continuously subdividing the triangles. Ideally it should thus be added to the standard algorithm for better results, however it is rather slow, and should thus only be used as a possible polishing of the final solution. When applied to this result, the size of the mesh is allowed to go further down to 16 vertices and 16 triangles, before the cost collapsing an edge gets too high.

Vogiatis achieved a mesh with only 8 vertices, however he does not mention the re-projective error, and can thus have used some unknown method to avoid it. The minimum solution however, can easily be obtained by enlarging the penalty for having vertices to a sufficiently high level out ruling the image errors caused by the edge collapses. In this case it has been sufficient to raise the vertex penalty to about 300, which brings the size of the mesh down to 7 vertices. A vertex penalty at this size is however highly unrealistic in real models. This final

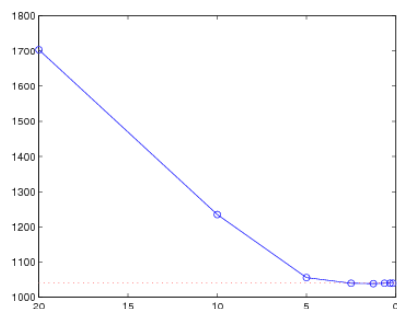


Figure 12.4: The the size of triangles versus the image cost for the box.

result has a mean euclidian error of 0.3 compared to box size of $10 \times 10 \times 10$. Most of this error comes from the back most vertex, which is natural, since least information is available there.

12.1.2 The Bunny

The box is a very simple object, that can easily be represented using only a few vertices. To test the system using a more difficult object, while still using synthetical data, the Stanford bunny has been chosen. 3 512×512 images are captured using different viewpoints, to show that the algorithm is not limited to the stereo case. The scene with the 3 images and the annotated initial mesh can be seen in figure 12.5.

As with the Box, the initial situation and the result have been included, see figure 12.6. This time the amount of vertices is less important and thus the vertex penalty has been set substantially lower. It should be noted that since we have included an occlusion penalty, the convergence is not only determined by a search for the correct model with respect to the images and the smoothness term. The initial mesh contains a substantial amount of occluded and alpha pixels, as can be seen in the snap shot buffer. The cost of these pixels are relatively high, which in turn means that the first part of convergence is mainly concentrated on limiting this cost, in some situation on behalf of the photorealism of the model.

The result shows that the general appearance of the bunny has been improved and some of the errors due to the triangulation has been corrected. The result is a smooth low resolution mesh that re-projects the input images fairly well.

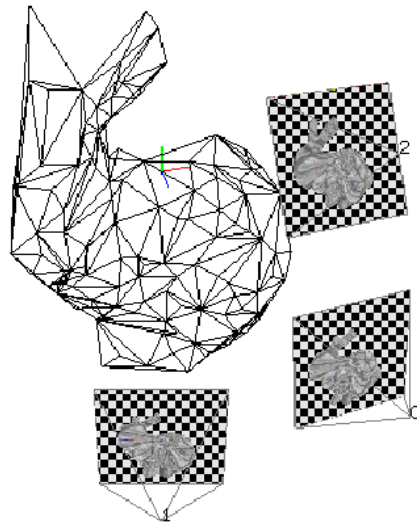


Figure 12.5: The initial composition of the scene using the Stanford bunny as an object

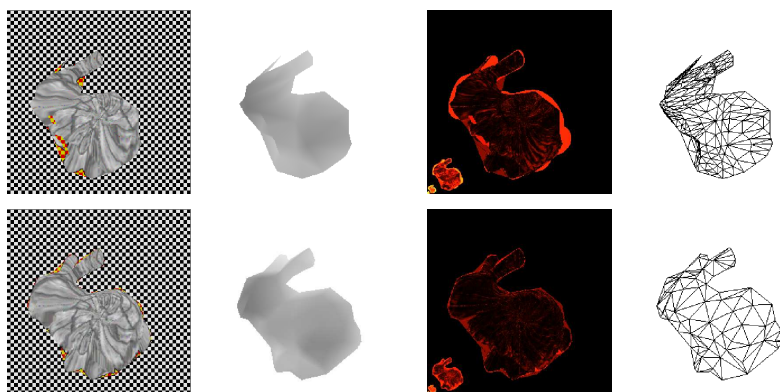


Figure 12.6: The buffers of the initial situation, and the result after convergence of the Bunny test. The resulting model can be found in `Bunny/test2/model11.x3d`.

Many of the smaller details are not captured, for example the ear is represented as a flat shape, while the ground truth clearly is concave. The curls in the wool are far too detailed to be captured at this mesh resolution, however the hips, the rear ear and the tail have the correct form even though the initial mesh lacked these details. The snapshot in the result contains occluded pixels, which is also expected since the input images are captured from different views. The differences between the two error buffers show a large improvement, however there is still a considerable amount of information not captured by the model.

It would be interesting to hold the resulting model up against the ground truth Bunny, measuring the MSE of the vertices. Such an evaluation however has not been implemented, as the focus has been on improving the algorithm.

Deformation Strategies

The actual distribution of the deformations in this test is shown together with their acceptance ratio and their relative gain in Table 12.1. As can be seen, the distribution of the deformations is not completely uniform. The reason for this can be found in the deformation selection process. If a deformation is chosen, but can for some reason not be performed, then another deformation is chosen at random. Thus if a deformation type has a high drop ratio, then it will have a lower overall usage. This is the case with the swap deformation. An edge can only be swapped if it has two neighboring triangles, thus it fails at every border

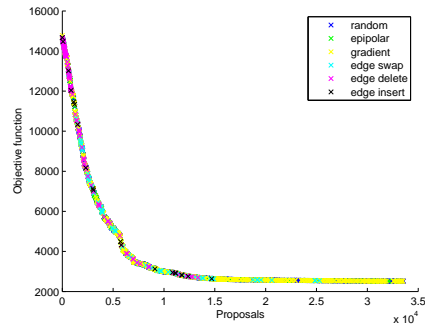


Figure 12.7: The objective function versus the number of iterations used in the bunny test.

edge.

Deformation	Acceptance	Usage	Gain
random	23.21%	13.14%	15.94%
epipolar	15.60%	11.46%	23.83%
edge swap	5.47%	9.80%	6.35%
edge collapse	2.40%	13.18%	11.73%
vertex split	0.98%	12.99%	9.77%
gradient	13.13%	13.12%	15.88%
normal	25.06%	13.19%	13.08%
triangle divide	3.60%	13.12%	3.40%

Table 12.1: Bunny test run without auto adjusted constants.

Surprisingly the normal and the random deformations are most successful on average. This can have several reasons. First of all, these deformations are uniformly distributed in space about the old vertex. Thus some of them will not change the mesh very much, which gives a high chance of acceptance using the simulated annealing approach. Second, the initial mesh does not fit the bunny's outline in the images. This requires movement perpendicular to the normal at the border, which may explain why the random deformations are relatively well accepted. When refining the mesh, the proposed triangle split deformations performs significantly better than the vertex split as expected.

Vogiatis claims to have obtained acceptance ratios of 20.2% for random, 30.5% for gradient and 47.2% for epipolar deformations. This is similar to our result for the random deformation, however he achieves much better acceptance for gradient and epipolar deformations. The source of this difference can either be

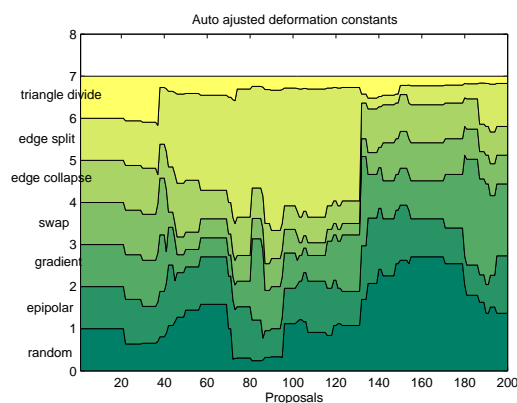


Figure 12.8: The distribution of the deformations in the first 200 proposals of the Bunny test.

differences in the implementations and or the nature of the data. As such, no single reason can be identified.

A more fair basis for comparison is the relative gain of the deformation types. Using this the epipolar strategy performs best, while the 3 other vertex deformations are similar. Interesting is that the 3 original connectivity deformations has a high gain compared to the acceptance ratio. This however is understandable as they move the vertices in space perpendicular to the normal of the mesh. The new triangle divide was chosen for not doing this, which explains its low gain.

Auto-adjusting Constants

To evaluate the impact of using an adaptive distribution of the deformations, the same test is run again. The resulting model is very close to what is already obtained, however as table 12.2 shows, the acceptance ratio has increased. In total the first test accepted 11.3% while using a non-uniform distribution gave 15.3%. Especially the deformation types having low acceptance before is now accepted a little more often. This shows that the convergence needs fewer connectivity deformations than is gained from a uniform distribution.

The result on the distribution of using auto-adjusted constants is illustrated, for the first 200 proposals, in figure 12.8. All constants start out using the

Deformation	Acceptance	Usage	Gain
random	23.95%	21.62%	23.00%
epipolar	14.80%	13.12%	19.94%
edge swap	6.47%	6.29%	7.15%
edge collapse	3.55%	8.29%	8.57%
vertex split	1.56%	7.24%	3.89%
gradient	14.09%	13.08%	11.41%
normal	23.70%	21.37%	13.25%
triangle divide	5.23%	8.99%	3.02%

Table 12.2: Test run with auto adjusted constants

uniform distribution. The vertex split deformation takes a long time before enough information is gathered to base a constant on, which is why it has such a large part of the deformations to start with. After this, the distribution finds a more natural bearing.

12.2 Architectural results

The next set of tests of the implementation is done using architectural scenes from the real world. Architectural scenes are chosen next, as they have some nice properties for 3D reconstruction. They mostly consists of large planar or smoothly curved surfaces, that can easily be represented by relatively few vertices.

12.2.1 The Pot

The Pot dataset would probably not be described as a piece of architecture by most people, but it has been included here because it is simple and has many of the same properties as architecture. Image 1, 2 and 3 are chosen as input for the algorithm as they present a challenge because of their difference in light distribution, as will be discussed later. First a simple reconstruction using SSE is performed, as is documented in figure 12.9. As can be seen, the result resembles the pot much better than the initial mesh. The occluded areas has been minimized and the mesh simplified to a balance between the image error and the mesh cost. The general shape of the pot is reconstructed, however the extruding edge in the bottom of the pot has not. When comparing the error buffers, it is difficult to identify the edge as a significant error, which may explain why the model has not adopted to it. In figure 12.10 the convergence is plotted.

Like the Box, it shows a large amount of edge deletes to start with and spatial adjusting towards the end. Interesting is that there also is a notable amount of edge inserts used to adjust the mesh throughout most of the convergence.

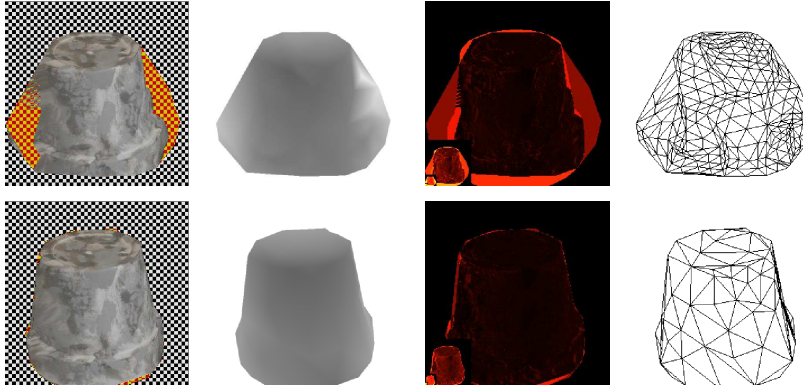


Figure 12.9: The first row shows the initial situation. The second row shows the result after convergence. The resulting model can be found in `Pot/test1/model11.x3d`.

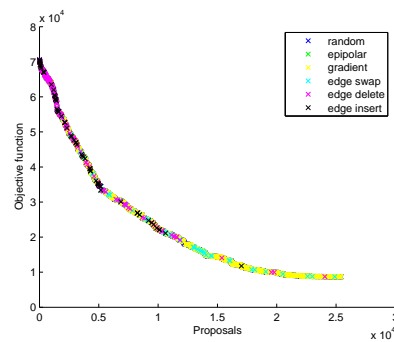


Figure 12.10: The objective function versus the number of proposals used.

Image metric - SSE versus Correlation

The images from this dataset are real world images having visible differences in the color distribution, which would thus make a good foundation to evaluate the differences between the two proposed image metrics, correlation and SSE. The algorithm is based on a randomized search and it would thus provide an

inaccurately comparison if evaluating the two image metrics separately. Therefore both have been calculated and stored for each iteration, while only one of them has been the controlling metric, ie. the one used in the proposal selection. To avoid ambiguities, two tests are conducted to let both metrics control a test. Further more the same tests are performed on normalized images from the same dataset, to reveal the possible differences. In theory the random nature of the algorithm could favor one of the image metrics, however because of the vast number of proposals this should not be significant. Figure 12.11 shows the convergence of these 4 tests each with the metric either itself controlling the convergence, or not. To account for the difference in the actual cost, the cost have been normalized.

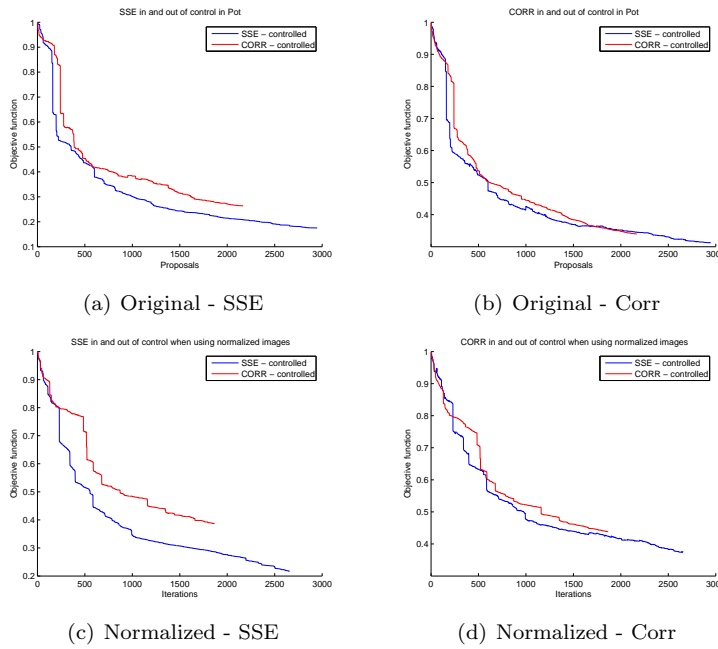


Figure 12.11: The convergence using SSE and CORR, both controlled by itself and the other. The upper row shows the result from the original Pot dataset, while the lower shows the result on a normalized version of the same dataset.

As can be seen, the two metrics follow each other in the convergence. In general it seems like the convergence is slightly better when SSE is in control. In the normalized images, the gap between the two metrics are enlarged, which is natural since the advantage using CORR lies in the 1 order statistical differences between the input images. It however is surprising that SSE performs better than CORR in the non-normalized case. In the upper left graph, when SSE

is in control it is expected that the convergence is better than when CORR is controlling, which is the case. In the upper right, however it is unclear if the corresponding is true for CORR. It is expected that CORR would perform notable better in control than when not, however only a little difference is present. The test has been performed 3 times, to exclude the possibility of the this result being extra ordinary. All 3 tests showed the same with a significance similar to this. One of the possibly explanation is that SSE uses all 3 color channels in its comparison, while CORR is limited to use a gray scale evaluation. If not, then it comes down to an implementation error, which is highly unlikely the relatively small difference taken into consideration. The last explanation is that the data could favor one of the metrics. Fx the initial model could have areas being wrong, but giving a lower cost to CORR than SSE. Thus it would make SSE seem better, however also this is unlikely to be the case in the amount seen here.

12.2.2 The Masters Lodge

The Masters Lodge shows a real piece of architecture, which is what Vogiatzis proposed as subject for this method to do surface estimation. The surface has a simple overall structure, however it contains a number of small details, that can be difficult to capture. A number of these in increasing level of detail are the actual masters lodge extending from the surface, the roof extending backwards, the windows intruding a little inwards, the embrasures in the walls at the roof etc. For this test, the three images presented in the dataset are used.

The result of a simple test is shown in figure 12.12. The initial mesh only contains 14 vertices, and thus requires many deformations to reach an optimum. The result shows a much finer grained mesh which encapsulates some of the details mentioned above. The structure of the masters lodge and the tower is captured and the roof extends backwards. The walls in general contains more vertices than seems necessary at first sight. This however is due to the windows in the walls as can be seen in the 3D model. The error buffer shows that there are many details not captured. Fx the structure in the windows are finer than were possible to achieve with the extra vertices. The convergence took approximately 20 minutes.

Annealing Schedules

This datasets is very similar to the second dataset used in Vogiatzis article. As mentioned in 8.2.2, Vogiatzis uses an annealing schedule, that very rapidly

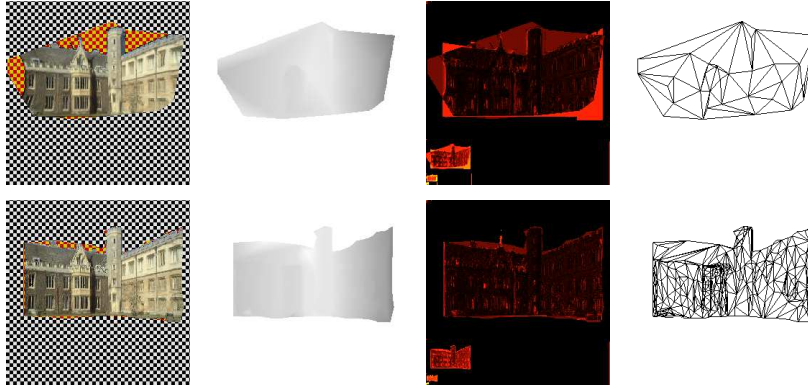


Figure 12.12: The first row shows the initial situation. The second row shows the result after convergence. The resulting model can be found in `MastersLodge/test2/model18.x3d`.

descends to an almost constant temperature. Two other proposed schedules having a more slow temperature descent were discussed. This together with the choice of having a constant temperature gives 4 different choices. These are tested using the masters lodge dataset, to be able to compare the results to Vogiatzis's result on the similar dataset.

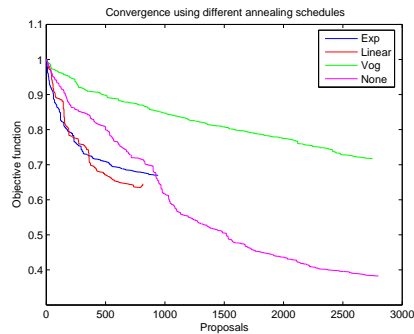


Figure 12.13:

The result can be seen in figure 12.13, which shows a clear difference between the annealing schedules. The exponential schedule starts in the lead however it is closely followed by the linear method. Both of them converge at close to 1000 iterations since a too small temperature for progress is reached. The schedule proposed by Vogiatzis never gets the chance to use the high temperature which

explains the slow, but steady descent. It does not however reach the same level in the objective function despite the 2500 iterations. As a reference a static schedule is included. Because of the too fast descent of the linear and the exponential schedule, it reaches the same cost at the same time they stop. However, it is static and thus continues. It somehow shows the potential of when using a good annealing schedule, as it should be able to reach this level of convergence and by lowering the temperature go even lower. It however also converges since the model has reached a level, where the temperature are too high for progress. This clearly shows the importance of the annealing schedule, and that great improvement count be achieved choosing the right one.

12.3 Difficult objects

To test if the method can be used on more difficult objects, it will be tested on the Cactus, the Gargoyle and the Face.

12.3.1 The Gargoyle

The Gargoyle is an object, that contains very little color information. It has a general humanoid shape, with many small details, because of small bumps in the material. The bumps are perceived as dark areas in the input images, which should help capturing the general shape, but unlikely to be captured as small deviations in the surface. Image 1, 2 and 3 has been chosen as input images sub-sampled to 360×243 pixels.

The result is shown in figure 12.14. It shows that the complexity of the object is causing problems for the algorithm. Most of the overall shape has been improved, and the error buffer shows that many details have been corrected. The bottom of the gargoyle (the top in the upside down images), the arms and the shoulder has been captured fairly well. Almost all small details are not visible in the resulting model. Especially the right side of the model presents a problem. Some of the texture below the head has been stretched to cover occluded areas, which shows that the penalty for occlusion has been set too high.

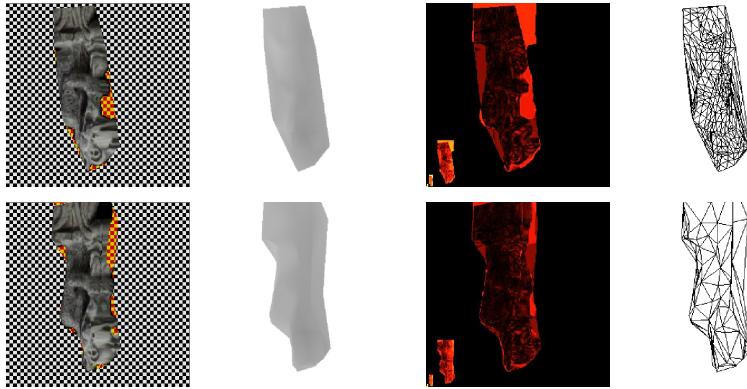


Figure 12.14: The first row shows the initial situation of the Gargoyles test. The second row shows the result after convergence. The resulting model can be found in `Gargoyles/test2/model1.x3d`.

Bayes selection / best choice

The gargoyles dataset has been chosen to test the difference between an algorithm setup in the Bayesian framework and an implementation based on the best choice. Therefore the test is run again, however this time the algorithm has been made greedy using the best choice selection rule. As can be seen in figure 12.15, the difference between the selection methods are relatively small to start with. It seems like the greedy approach has a small advantage over Bayes selection, which is naturally since no bad choices are made. This situation, however, changes when nearing convergence. While the greedy approach has difficulties finding good deformations, Bayes selections continues downwards. The convergence is reached after approximately the same amount of proposals. When studying the amount of deformations accepted, it is clear that Bayes selection allows for a wider range of deformations to be taken. Most of the time, almost twice the amount of when being greedy. All these small step backwards is clearly weighed up by the advantage of the ability to tunnel through small bumps in the objective function. This is not the singlehanded conclusion from this test, but have been shown in multiple other tests. This test using the gargoyles is merely an example of the effect.

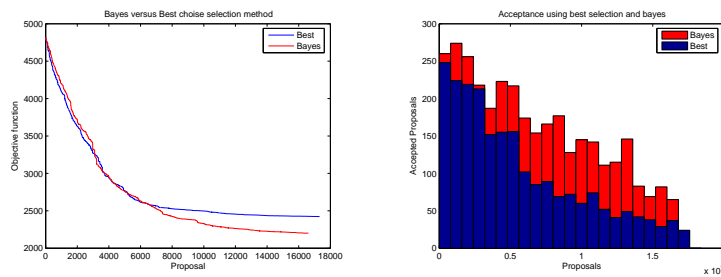


Figure 12.15: Left is the objective function when using either the best proposals of Bayes selection. Right is a histogram showing the acceptance in the same test.

12.3.2 The Cactus

The cactus is a rather difficult object. Each plant has a distinct color, which should make the overall surface easy to estimate. The texture of each plant, however is repetitive, and contains spikes, that appear as a blurry cloud, difficult to model. Image 1, 2 and 3 has been used in a simple test setup, like in the previous tests. The result is shown in figure 12.16. The error buffer shows an improvement, however admittedly, it is rather small. The rim of the pot and some of the figures on it has been improved, however the spikes of the cactus are visible, which shows that there is unused information. This however can also be due to the spikes making these areas appear different from different angles, which is supported by the fact that only one set of spikes can be identified in the error buffer. Like in the Gargoyle, the texture of the red cactus has been stretched to cover occluded areas. In the wireframe, the rim of the pot is not visible at all, which may be due to the small angular difference the input images has. It however shows the adaptiveness of the mesh, as the red cactus requires more vertices to present its structure than the pot. Studying the 3D model will show that the reconstructed pot is not very smooth and round. Especially the rim presents a problem. The cause of this can be that the texture at the rim is almost uniform going round the pot. Thus it gives the same visual appearance when some of the texture is 'copied' to other areas of the rim by erroneous structure. The result obtained here is far away from what other algorithms have shown possible. They however achieve this using conceptually different approaches.

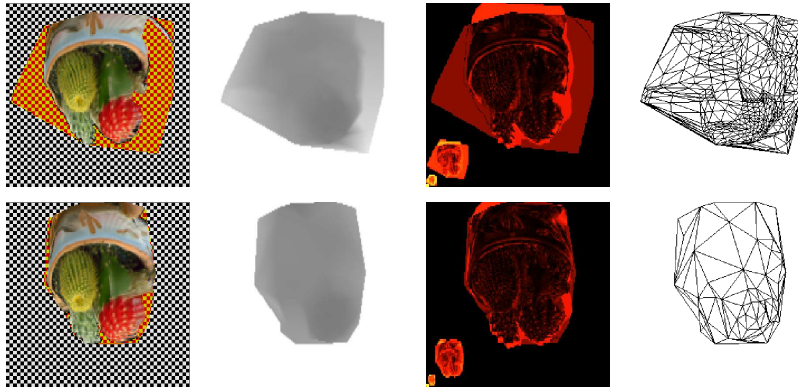


Figure 12.16: The first row shows the initial situation. The second row shows the result after convergence. The resulting model can be found in `Cactus/test1/model11.x3d`.

Capture Method - Blending or Pairing

The Cactus dataset contains images from all around the cactus. Therefore most of the cactus are visible in more than one image, or in other words only a few pixels in each `Camera` are occluded. This is only when taking all cameras into consideration, like when using *blending* to capture the snap shots. If using *pairing* many pixels in each image will be occluded, giving rise to large errors. To show this difference 5 images have been chosen covering half of the cactus. These are image 0, 2, 4, 7 and 9. The convergence of the two tests have been recorded and can be seen in 12.17. As expected, the blending method copes best with the multiple images. The pairing method has difficulties finding a good minima, as it is too affected by the occlusion cost. The result is a strange mix of trying to remove occlusion and fitting the images. This single test is provided here to illustrate the differences, however many other tests has shown the same.

12.3.3 The Face

The Face datasets presents a large challenge, since human skin can not be modelled in a convincing way using the standard light model, see [23]. The algorithm implemented assumes Lambertian objects, however it is interesting to see how the algorithm behaves under these circumstances. A single set of the rectified images are used for the test. Figure 12.19 shows the resulting buffers. The overall structure of the face has been captured, that is the chin, the cheek and

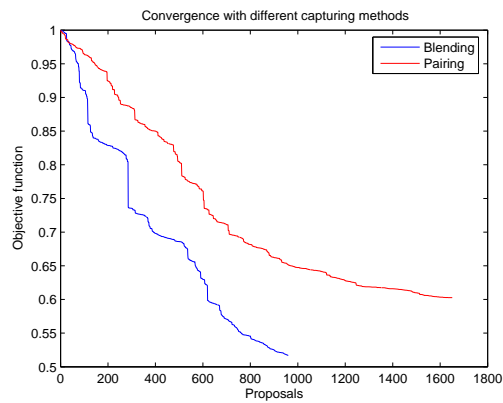


Figure 12.17:

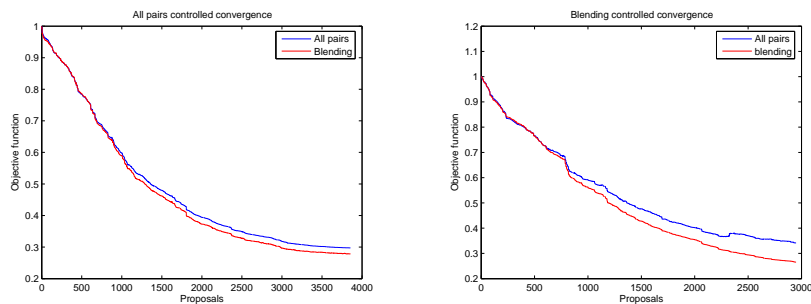


Figure 12.18: Two plots of the image metric, when capturing using blending or all pairs. The first plot is controlled by the image metric resulting from using pairs of images, while the other is controlled by blending

the forehead. The eyes are showing as inwards intrusions, but not very well. The nose is curled up in a complete unrealistic manner. If studying the input images, one will find, that the nose has a specular highlight at different places on the two images. The curling up of the nose can be an attempt to match this highlight without changing the texture of the rest of the nose. The error buffers shows almost no improvement, except that the final model fits the outline of the input images better. This can either be because the initial mesh represents the surface well, or because of the lack of details in the texture of the face.

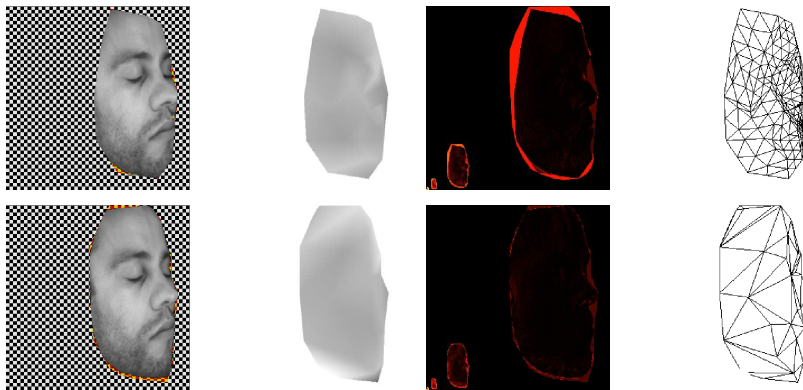


Figure 12.19: The initial buffers and the result from the Face test. The resulting model can be found in `FaceDataKHM/test2/model1.x3d`.

12.4 Summary of Test

To summarize the main results from the test are

- The algorithm is good at capturing the general structure.
- The algorithm is bad at capturing details.
- The algorithm has problems adapting the mesh resolution.
- Using the SSE image metric has a slight advantage over using correlation, because of using colors in SSE.
- Using an auto-adjusted proposal distribution gives better convergence.
- The annealing schedules used descents too fast and requires more work to become optimal.
- Using pairing for input is only advisable for small angular differences in the input images.
- The Bayes selection has a positive influence on the convergence.
- The implementation is not good for facial reconstruction.

Discussion

This section discusses and concludes on the workings of our implementation of the basic algorithm.

13.1 Identified Flaws and Issues

The largest problem identified in almost all tests is the general reluctance to adapt to lesser details in the input images. The overall structure of the model rarely presented a problem to the algorithm, however when refining this result, the model either adapted using some unrealistic surface or not at all. An example having both successfully adopted and erroneous areas can be seen in figure 13.1, which was obtained manually adjusting the constants and run the algorithm multiple times with increasing mesh resolutions. The problem relies in the smoothing term. Setting it too strongly cripples the algorithm and setting it too weak allows a distorted surface.

The relative low resolution in the results presented in the last chapter was chosen to avoid too long running times. Each proposal only deforms a few vertices, making the convergence very slow for large meshes. Some long runs, for example the one in figure 7.1, took over 16 hours, because of the large amounts of vertices involved.

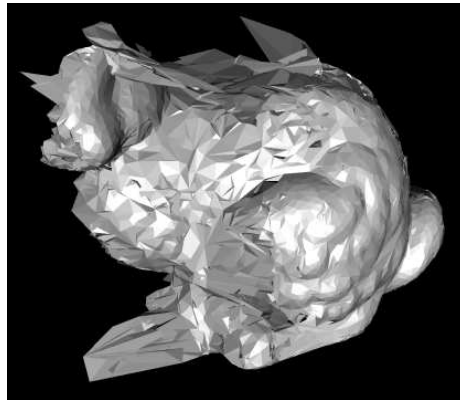


Figure 13.1: A screen shot of a model the Bunny achieved using the algorithm multiple times. The screen shot shows areas where the model has captured the details nicely and areas, where the surface is distorted. The X3D model can be found in `Bunny/curly.x3d`

13.2 Conclusion

As can be seen from the results in the last chapter, the implementation of the algorithm functions as expected. Models deform into more visually correct surfaces under the constraints of the smoothness term. The algorithm however has problems adapting to details making it weak, when details are needed. It however is good at reducing the resolution in the input mesh in an adaptive way.

It is believed that the performance of the algorithm could be improved considerably using the correct constants, however as can be seen finding these constants is a time consuming task, not completed yet.

Part III

Improvements

Discussion of Possible Improvements

As concluded in the last part, the basic algorithm does converge, however it has several drawbacks, that makes it reach a non global minima of the objective function. This part is a presentation of a number of proposals to better this. Some deviations to the algorithm presented by Vogiatzis was already discussed in the last part, however these were merely different implementation choices, which either were necessary or deserved to be evaluated. The proposals presented here are new ideas, that would change the algorithm in a way that would not fit well into last part. Some of these proposals are implemented as an extension to the algorithm, and has thus been tested and evaluated. Others are theoretical ideas, that has not been implemented because of lack of time.

Presenting, implementing and evaluating such proposals would normally require a substantially work, however as the basic algorithm already provides the needed framework, this can done fairly simple. The rest of this chapter introduces the proposals presented, and the background for doing so.



Figure 14.1: Figure of a group of vertices trapped in a local minima. The red arrows shows a possible, but improbable escape from the minima.

14.1 Coarse to Fine Technique

The algorithm, as presented until now, requires a good initialization. It would be interesting to investigate the possibilities of using this surface estimation technique in a full scale algorithm, that starts of a simple guess, fx a cube. This will be described in Chapter 15, where a coarse to fine technique will be used to stepwise increase the information available, i.e. the resolution of the input images. This technique is therefore denoted as the *zooming approach*.

14.2 Improving Convergence

When improving such an algorithm one must ask one self: *what is going wrong*. It is known that the scene has some (possibly unknown) ground truth surface, that can be achieved by deforming the model using the current deformations, as discussed in [57]. It may however be that achieving this is very improbable, and thus very unlikely to happen when running the algorithm. Thus the question can be reformulated into: *what makes the real surface improbable to reach*. This question has many interesting answers, some of which has already been discussed. The following discusses some of the answer to this question.

One of the key elements of the algorithm is the deformation of the vertices. Using the random deformation, all possible surfaces can be reached if the accompanying mesh has sufficient resolution. The non random deformations makes it more probable that a vertex is moved so that it will fit the image information better, however sometimes vertices can be trapped in local minima, simply because it is too improbable that it would deform into the right shape. An example viewed in 2D is a spiky surface estimated with a plane, please refer to figure 14.1. Moving any single vertex will make the current model deviate more from the real surface than gained, and thus make the cost higher. Moving all vertices along the red arrows will make it fit the surface and make the cost lower. Thus to escape the minima, series of improbable steps must be chosen. To improve this, it could be penalized not being on an edge, and thus making a single step towards the real

surface more probable. One useful observation of such sharp edges, is that they usually appear as a feature in the images. Thus using the features in the input images to adjust the vertices may improve the convergence. This is discussed further in Chapter 16.

14.2.1 Error Based Selection

One of the main problems relating to the speed of convergence is that the algorithm blindly picks a triangle or vertex and deforms it. It can thus in theory always pick the same deformation, and thus never converge. As the algorithm is defined now, the algorithm is image based, ie. the cost is evaluated in image space. This makes the result photorealistic, however it also makes the choice of the deformations a blind search. In most image based algorithm the evaluation of the estimated surface is used directly in the deformations, see fx [26] and [60]. Similar primitive selected as base for a deformation could be chosen using the cost the primitives infer on the model. This will be discussed further in Chapter 17.

The Zooming Approach

This chapter will describe an attempt to use the basic algorithm without an initialized mesh obtained using another surface estimation method. As described in Part I, making such an initial guess is a thoroughly researched area, and robust methods for doing so have been developed for most cases. It is thus not out of necessity, but more curiosity that this is investigated here. First the idea is introduced, then the algorithm build on these ideas is presented. It is tested and conclusions upon it is made.

15.1 Introduction

Using the algorithm without proper initialization is straight forward, fx a simple cube could be used as a starting guess. The result doing so, however is rather poor and the convergence is very slow. Especially areas of the surface with repeating patterns is subject to errors, as no initial information of the 3D structure is given, and thus the algorithm may find a strong local minima at a wrong 'shift' in the pattern. An example of this can be seen in figure 15.1, where the regularity of the spikes in the cactus dataset fooled the algorithm.

A simple method to both speed up the algorithm and improve the result is to make a coarse solution first using a down sampling of the input images. This



Figure 15.1: Two screenshots of a piece of the cactus dataset, where the patterns on one of the cactus' has been shifted. The left shows the cactus from an angle visualizing the shift, while the right is from an angle similar to the one of the Camera's.

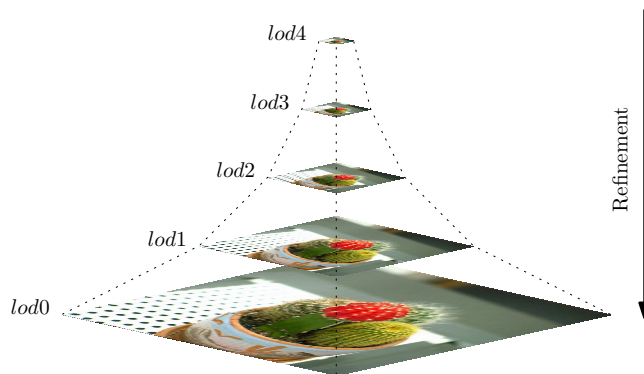


Figure 15.2: A pyramid from the Cactus dataset. For every level of detail, the model is refined.

argument can be used again on the algorithm running on the down sampled images until single pixel images are reached. This would correspond to run the algorithm on an *image pyramid* (see figure 15.2), jumping down the steps as a final solution is reached, or to iteratively zoom into the object, for better detail, hence the term, *zooming approach*. It should be noted that it does not correspond to real zoom, as it would result in the field of view getting narrower for each step. In addition to the faster convergence with respect to the number of proposals used, it is also much faster to operate on smaller images.

By running the algorithm on down sampled input images, some of the errors originating from repetitiveness in the texture is avoided, as the down sampled images have smoothed out the patterns. Thus the problematic cactus from before would start out as a green blob, from which the general structure of the cactus could be achieved before the algorithm is introduced to the repeating spikes.

15.2 The Zooming Algorithm

To describe the *zooming*, we define **lod** as the *level of detail* in the same spirit as OpenGL. **lod**₀ is thus the highest resolution available in the input data. As very little can be induced from very low resolution images, a minimum initial resolution is defined to be 32×32 pixels. The algorithm assumes no uniformity in the size of input images, however best results are expected when the images have approximately the same δ -value. If not the convergence may concentrate on reducing the error in the high resolution images, while low resolution images are neglected.

There exists a number of different methods for downsampling images, fx gaussian pyramids, wavelet pyramids, nearest neighbor pyramids etc. For simplicities sake, and to exploit the build in software in OpenGL a simple averaging method is chosen, where each pixel at a level k is the average of the 4 corresponding pixels in level $k - 1$.

The formulation for the algorithm using the zooming approach is straight forward. We simply add an extra outer while loop, that increases the detail level for each run of the basic algorithm. The algorithm can be seen in algorithm 15.2. To account for some of the problems regarding mesh refinement, each loop subdivides the mesh. Thus each run of the basic algorithm will be able to remove the unnecessary vertices, while the others can be adjusted to fit the new information in the data.

Algorithm 4 The Zooming Algorithm

```

Set lod to lodstart
while lod ≥ 0 do
  while converging do
    deform the surface
    evaluate objective function
    if better than not deformed surface then
      keep the deformed surface
    else
      discard the new surface
    end if
  end while
  increase the lod
end while

```

15.3 Results

The new algorithm was tested using both the Cactus dataset and the Pot datasets. To visualize the refinement steps of the Cactus dataset, the resulting buffers of each level of detail are shown in figure 15.3. The algorithm took about a half hour to converge. As can be seen, the simple cube is deformed into an object of the correct size and position in the highest level. From there each level refines the mesh, and adjusts the vertices. The final mesh resembles the cactus well, which can also be seen in the 3D model. Not much is happening in the buffers from level 2 to level 0, which is mainly because of the limited resolution of the printed buffers. The mesh is subdivided at each refinement step which makes the mesh resolution explode. It seems like the algorithm does not completely exhaust the information given in each step well enough. Thus not all unnecessary vertices are removed, which causes even more new vertices in the next subdivision. The result of this test is as good, if not better, than the result obtained using the basic algorithm using an initial mesh.

Figure 15.4 shows the same test using the pot dataset. As can be seen also this has succeeded fairly well. The structure of the pot has been reconstructed with the exception of an artifact in the top. This example shows that using the standard algorithm as a base for the zooming approach makes it inherit the problems already discussed in Chapter 12. These problems however grow larger as they are used at a lower level of detail, and next multiplied. As can be seen from the Cactus results most of the errors done in an earlier step are taken care of, when sufficient information is available to correct it. The Pot however shows that this is not always the case.

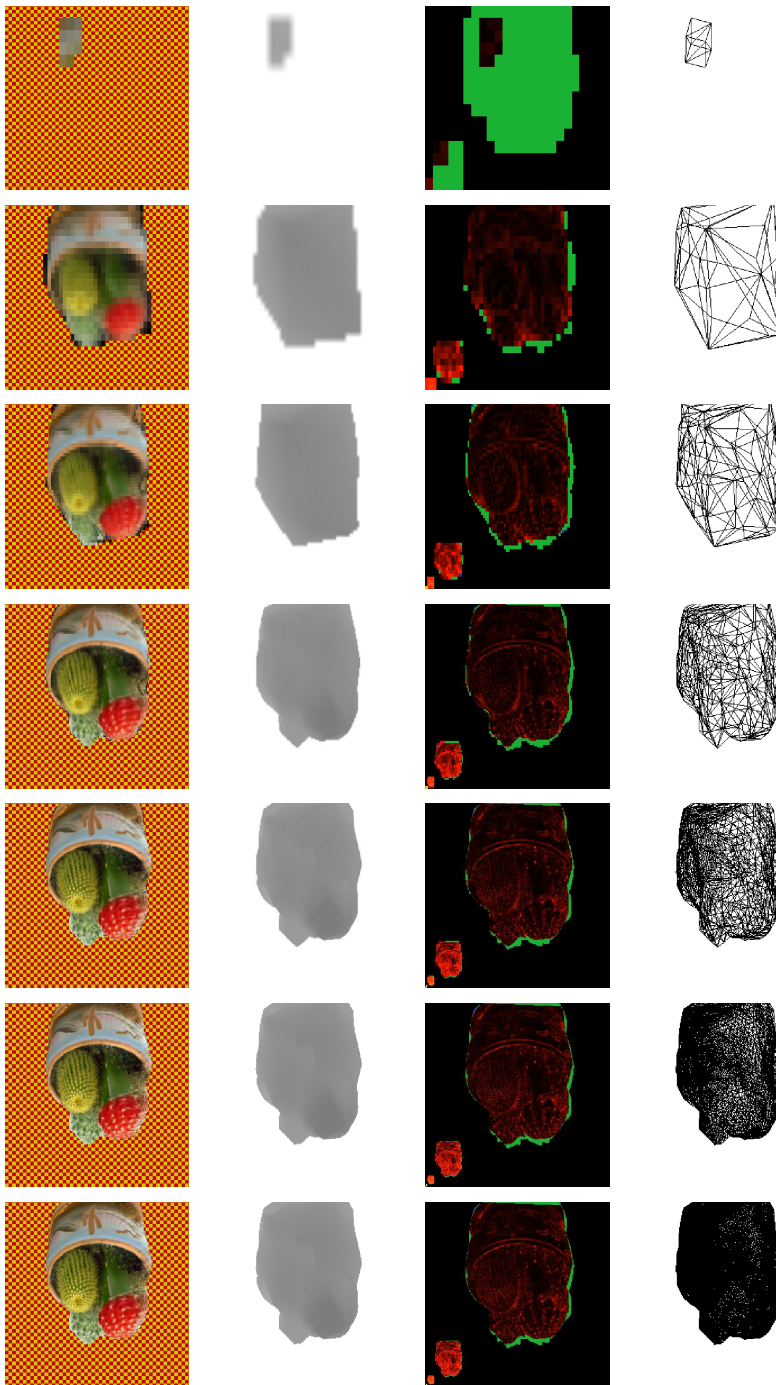


Figure 15.3: The result of using the zooming approach on the Cactus dataset. The buffers are shown in chronological order, i.e. from lod_6 to lod_0 . The resulting model can be found in `Cactus/zoom2/zoom0.x3d`.

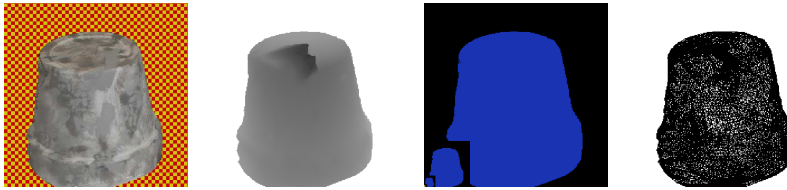


Figure 15.4: The result of using the zooming approach on the Pot dataset. The error buffer is blue because CORR is used, which does not visualize well. The resulting model can be found in `Pot/zoom1/zoom0.x3d`.

15.4 Conclusion

The tests clearly shows that an initial mesh is not a necessity for estimating the surface using the deformable models method. The model takes longer time to converge, and the result can contain major defects, if the individual steps can not cope with them before they become to large. Besides that the quality of the models obtained are good.

Use of Feature Maps

In this chapter the use of extracted *feature maps* will be discussed. As mentioned one of the problems reaching the global minima of the objective function (the correct surface) is that there is no control of the distribution of the vertices on the surface, as long as their projections resembles the images to the current quality of the reconstruction. Thus when deforming the model, the vertices may be positioned in an unfortunate way that makes a successful refinement improbable.

16.1 Introduction

One proposal that may better this, is to extract feature information from the input images, like in feature matching, and use it in the positioning of the vertices. The argument for doing so is twofold. Firstly, features provide good correspondences, and thus, if a point in space both match a strong feature in all images and gives low error when the local mesh is evaluated, then it is even more certain that the position of the point is correct. Secondly many difficult parts of a surface gives strong features in the images thereof. Examples of this could be corners or sharp edges, that due to lighting conditions or occlusion of underlying objects with different texture, is a stronger feature than the surrounding surface.

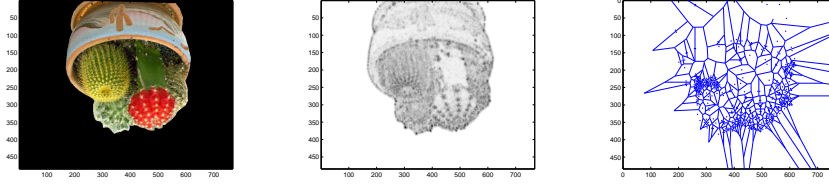


Figure 16.1: Figure of one of the images of the cactus dataset and the corresponding Harris Field (cubical root of it to improve visualization), and a Voronoi diagram of 288 of the strongest features.

Figure 16.1 shows what features extracted from images may look like. There exists a number of feature detection algorithm that can produce this kind of data. The feature field in the figure is the result of *Harris Corner Detector* as discussed in Chapter 4.

16.2 Implementation

Aside from the normal objective function, one could add a lookup for every vertex into the Harris field of every image and favor vertices being at strong features while penalizing when low feature. The cost for not being at a strong feature should be big enough to encourage the positioning there, but small enough to secure that vertices avoids being positioned at positions with weak features, if these are correct regarding the input images. Thus it should make vertices attracted to nearby features, but not force the model to remove all vertices in uniform areas with no features. This can be achieved by removing a part of the penalty for having a vertex in the mesh, discussed in section 9.2, based on how strong the features are in the projections of the vertices into the images. As the cost for having a vertex is now, it is not high enough to make the model remove all vertices. Thus it will only be an extra reduction in the cost, when a vertex is placed at a strong feature in space.

To be able to adjust the significance of the feature cost in the results a weighing constant, w , is introduced. This gives the final feature term

$$\mathcal{F}(M) = \frac{1}{|\text{cams}|} \sum_{c \in \text{cams}} \sum_{v \in [M]} \max(0, \text{vertexpenalty} \times (1 - wf(v, c))) \quad (16.1)$$

, where $f(v, c)$ is the feature strength of v in camera c . The max term are used to ensure that the cost is positive.

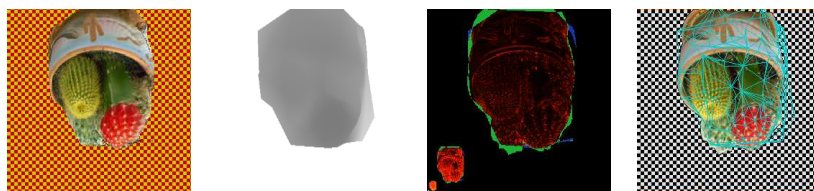


Figure 16.2: The result of one of the tests using feature cost, where $w = 30$. The resulting model can be found in `Cactus/feature30/model1.x3d`.

16.3 Results

One of the datasets that presented large problems because of uncontrolled vertex placement are the Cactus dataset. As can be seen in figure 16.1, the rim of the pot and many of the spikes are clearly visible in the feature field. It may thus be worthwhile to add this feature penalty, to make it beneficial for the vertices to use these points in space. This has been done for 5 different weighings of the feature cost, where a high weight gives large bonuses for using feature points. In general w is chosen relatively high, as only the strongest features would otherwise be useful. The resulting convergences can be seen in figure 16.3. Recall that adding a feature cost actually removes some of the cost with respect to the basic implementation. As can be seen in the right plot, the amount is much less than what is actually gained in the objective function using the feature cost.

As an example the result using a weighing of the feature cost of 30 can be seen in figure 16.2. If studying the 3D models of the result it can be seen that many of the vertices are placed on or close to strong features. In general less errors in the models occurred when using the feature cost, which is natural, since erroneous surfaces that appear correct with respect to the images, seldom match the features as well.

16.4 Conclusion

The feature cost adds a new constraint to the mesh, making it more willingly to use good feature points in space. The result is that some errors are avoided,

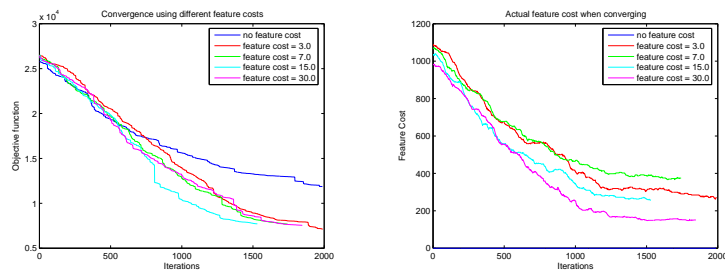


Figure 16.3: The convergence of 5 different tests weighing the feature cost different. To the right the actual feature cost is shown.

and that the result in general fits better. It has a small but significant impact on the objective function satisfying the objective for introducing it.

Error Based Selection

This chapter will describe a novel idea, where the randomized selection of a primitive for deformation, is substituted with a more sophisticated selection based errors. The idea has not been implemented fully, and this is thus only a presentation of it.

The algorithm as described makes use of some of the image data in its deformation of the model, however the selection of the primitive to be deformed are completely random. This makes the algorithm awfully slow for large meshes, as the algorithm may 'guess' wrong many times before finding a primitive, that actually needs a deformation.

17.1 Using Errors

One intuitive way of improving this is to store the cost of each triangle in the evaluation, and use it to select a *bad* triangle or one of its corners. This however is not straight forward when using an image based algorithm. It would require projecting the triangles into the **Camera**'s and sum the error at the pixels that they cover. This is both slow and difficult as some pixels are shared between triangles.

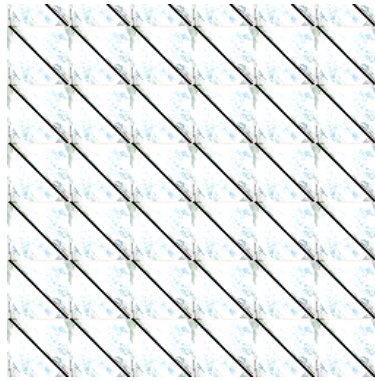


Figure 17.1: The automatic sampling and evaluation of triangles arranged in a texture. The actual triangles and their internal error can be made out.

On the other hand an object based approach could be used as in a correlation deformation. Sampling on the triangles and projecting them into the images to compare their colors are relatively slow using the straight forward approach on the CPU. The task however can be formulated as a problem solvable on a graphics card. Figure 17.1 shows how arbitrary triangles could be sent to the graphics hardware. Sampling in the triangles could be done using the automatic interpolation, lookup of the color values could be done using texture matrices like in section 10.2.4, and comparison of the colors using an image metric. The result for each sample, stored in the corresponding pixel, can even be summed into scalar values of each triangle using a modified summing algorithm.

As the algorithm only deforms a limited number of primitives each iteration, all triangle errors need not be updated each time. A deformation may however render a distant triangle occluded or un-occluded, thus changing its cost. Therefore to avoid too large errors the cost would have to be fully reevaluated regularly.

The triangle cost can not be used directly as a vertex cost, when searching for an expensive vertex to deform. A vertex's cost could however be defined as the average cost of its owners. Thus if a vertex causes its nearby triangles to have a large cost, it will itself have a large cost, while the neighboring vertices $[v]$ would only be effected by a few of the high cost triangles.

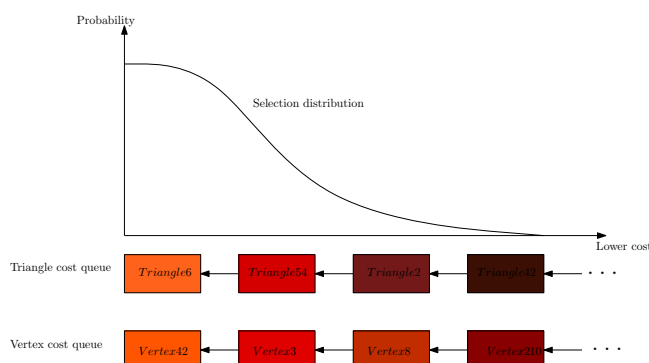


Figure 17.2: Illustration of the selection using cost queues.

17.2 Setup Using Queues

Using one of the methods described for obtaining individual costs for the triangles, a primitive must be selected. A simple solution would be to either take the worst triangle or the worst vertex by searching through all the error costs. Two issues advocates not using this directly. First, the search through all vertices and triangles are slow. It would be wiser to implement a queuing data structure, where the triangles and vertices are sorted by their cost. Inserting into such a queue only requires $n \log(n)$, where n is the number of triangles or vertices. Second, always selecting the worst primitive can block the algorithm. Fx if a primitive for some reason causes a large cost, but is very unlikely to be improved without other deformations as well. In stead it is proposed to select from the cost queue using some selection distribution. Fx this could be the positive normal distribution. The cost queues and the selection are illustrated in figure 17.2.

17.3 Discussion

This method, if implemented correctly would provide a more error based algorithm. It is expected that it would help the speed of convergence, however, it may be that it will make the algorithm get trapped at local minimas easier, because of the constraints in the selection process. The proposal can be seen as a bridge between the image based and the object based algorithm, as the advantages of both are used. Unfortunately the time to implement this fully has not been available, and it must therefore remain as an idea.

Part IV

Discussion

Future Work

As described in Part I, the field of surface estimation is in rapid development. Complete novel ideas are spawned, old ideas are improved on and ideas are fusioned together. This research area is still to be discovered by the heavy investments that are seen in other areas with great potential. In this chapter some proposals to future work is presented together with a discussion of the most important catch-ups of the current implementation. This should be seen as an extension to the ideas already proposed in the last part of this thesis.

18.1 General improvements

As has already been discussed, the algorithm has a greater potential than has been achieved here using a limited amount of time. Thus the following issues could be addressed in order to improve the implementation:

- A study of the annealing schedule, to find a more optimal schedule than used.
- Some means of reducing the computational cost when evaluating a single deformation. Fx locality of the involved primitives in the input images could be exploited.

- Better tuned constants.
- Implement a feature matching and the 8-point algorithm to achieve a full framework for surface estimation as described in section 3.2.

18.2 Model Texture Creation

Many applications that could have use of a surface estimation method requires textured surface models. The X3D models resulting from a surface fit using the implementation are texturized, however for simplicity this is done using a single input image and the coordinates projected herein. Thus for a complete surface reconstruction, the 'backside' would be textured with the same texture as the 'front side'. In [13] multiple textures are used to create a super-resolution texture of a model. Such a method could be used to create a full texture of the model, improving the use of the algorithm.

18.3 Using Frontier Points

As proposed in [29] and discussed in 5.6, frontier points can be used to estimate materials and lighting conditions in a scene. The graphics hardware used in the implementation could easily be programmed to take lighting and material into consideration when producing the reprojections. Thus the Lambertian assumption could be relaxed, which would enable the use of the method in a new range of more difficult images.

18.4 Higher Order Topologies

As the mesh and its deformations are designed now, it can not change its initial topology. In other words, no deformations exists to break the mesh and glue it together again. Many real world objects, as fx the gargoyle in figure 18.1 has holes in them. Therefore it would be necessary to extend the algorithm with gluing and breaking deformations in order to be able to reconstruct this class of objects.



Figure 18.1: One of the images of the gargoyle dataset showing that there is a hole in the statuette under the arm.

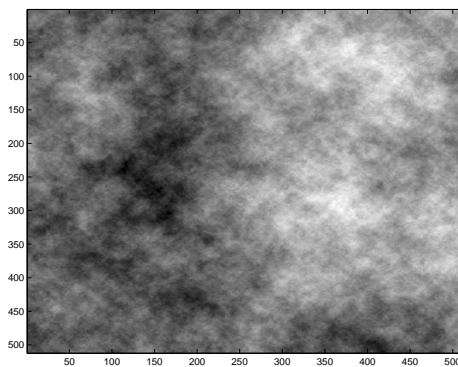


Figure 18.2: An example of Perlin noise.

18.5 Study of the Textures Impact on the Algorithm

One very interesting future challenge, that could unfortunately not fit into the time frame of this work, is the study of the impact the input image properties has on the quality of the result. As mentioned, intuition tells, that objects textured with many different frequencies of details would be easier to match. Thus it could be that the perfect texture would be so-called Perlin noise, a mix of random textures at different frequencies as seen in figure 18.2. A thorough research of the dependency between the statistical properties of the texture and the algorithm is thus called for.

Discussion

This Chapter will be used to discuss and conclude on the work done in this thesis. First the main contributions are listed and discussed. This is followed by a final conclusion.

19.1 Main Contributions

19.1.1 A Survey of the Field of Surface Estimation

A research in the area of surface estimation has been done. The major methods and concepts encountered has been described, both in a historical context and technically in a classification.

19.1.2 Preparation of Datasets

5 datasets acquired elsewhere has been prepared and standardized for the use in this thesis. Where not available a simple 3D structure has been annotated

and created in Matlab. Further more a program for creating new synthetical datasets from a wide range of 3D models has been implemented.

19.1.3 A GPGPU Implementation

An algorithm proposed by G. Vogiatzis has been implemented using GPGPU. The implementation has been thoroughly tested together with a set of small deviations to the proposed algorithm.

19.1.4 Presentation and Implementation of Improvements

3 major changes to the basic algorithm has been proposed. Two of which are implemented using the framework of the basic algorithm and evaluated. These are

- **The Zooming Approach** for using the algorithm without initialization. Implemented and evaluated.
- **Feature Cost**, where a feature map is used to add a feature cost to the objective function. Implemented and evaluated.
- **Error Based Selection**, where selection of primitives to deform are based on the error the primitives infer on the global cost. Described.

19.2 Conclusion

The objectives of this thesis were: 1) to conduct a research of the current state of surface estimation, 2) to describe and implement an algorithm using deformable models and GPGPU, 3) to evaluate and conclude upon it and finally 4) to propose, implement and test improvements to such an algorithm.

As described in the previous section, all of these objectives have been fulfilled. Chapter 2-6 presents the current state of development, both historically and technically, thus satisfying the first objective. The second objective is fulfilled in Chapter 3-10, especially in Chapter 10 where the use of GPGPU is described. The evaluation and the testing framework presented in Chapter 11-13 meets the third objective and finally, the fourth objective is achieved in Chapter 14-17,

where 3 conceptually new ideas are presented, two of which are implemented and evaluated.

The main focus in the work documented in this thesis has been on designing and implementing a program for surface estimation using GPGPU. Using graphics hardware can both be a fruitful and treacherous task. Programming GPGPU can sometimes feel like groping in the darkness. However, the implementation has largely been a success, though it has also shown that a great deal of the effort, to make such an algorithm perform well, should be put into tuning all the bits together. The result has proved that the underlying idea holds. It however has also shown and that it has a larger potential if future work is conducted.

To be able to make a computer 'see' is a dream. It is something that scientists will strive for, like they strived for putting a man on the moon in the sixties. The road to the current state of development has been tough, however it is my belief that with the current progress, we will soon begin to see small applications using surface estimation for various purposes. The work presented in this thesis has, at least for me, been a small contribution to the great goal of making this possible.

Bibliography

- [1] Devil - an open image library.
- [2] Stanford encyclopedia of philosophy: Bayes's theorem.
- [3] Vrm1 - virtual reality modelling language.
- [4] Wikipedia - the free encyclopedia.
- [5] X3d - xml based 3d world standard.
- [6] T. Drummond A. Broadhurst and R. Cipolla. A probabilistic framework for space carving. pages 388–393, 2001.
- [7] H. Stern A. Gelman, J. Carlin and D. Rubin. *Bayesian Data Analysis*. Chapman and Hall, 1995.
- [8] Henrik Aanæs. Methods for structure from motion. Technical report, September 2003.
- [9] Isaac Asimov. *I, Robot*. 1950.
- [10] Marco Attene and Michela Spagnuolo. Automatic surface reconstruction from point sets in space, June 03 2000.
- [11] Gregory Barattoff and et al. The encyclopedia of virtual environments (eve).
- [12] T Bayes. *An Essay Toward Solving a Problem in the Doctrine of Chances*. Philosophical Transactions of the Royal Society of London 53, 1764.

- [13] Alexander Bornik, Konrad Karner, Joachim Bauer, Franz Leberl, and Heinz Mayer. High-quality texture reconstruction from multiple views. In Roman Durikovic and Andrej Ferko, editors, *The Journal of Visualization and Computer Animation*, volume 12(5), pages 263–276. John Wiley and Sons, Ltd., 2001.
- [14] Matthew Brown, Richard Szeliski, and Simon Winder. Multi-image matching using multi-scale oriented patches. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 510–517, Washington, DC, USA, 2005. IEEE Computer Society.
- [15] Karel Capek. *R.U.R.* 1921.
- [16] David Capel. *Image Mosaicing and Super-Resolution (Cphc/Bcs Distinguished Dissertations.)*. SpringerVerlag, 2004.
- [17] RODRIGO L. CARCERONI and KIRIAKOS N. KUTULAKOS. Multi-view scene capture by surfel sampling: From video streams to non-rigid 3d motion, shape and reflectance. 2001.
- [18] Jens Michael Carstensen. *Image analysis, vision and computer graphics*. IMM, DTU, DK, 2002.
- [19] Jens Michael Carstensen. *Image analysis, vision and computer graphics*. IMM, DTU, DK, 2002.
- [20] D. J. Cavicchio. Adaptive search using simulated evolution, 1970.
- [21] chUmbaLum sOfT. Milkshape 3d.
- [22] Graph Cuts, Ramin Zabih, and Vladimir Kolmogorov. What energy functions can be minimized via graph cuts?, November 28 2002.
- [23] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Trans. Graph.*, 24(3):1032–1039, 2005.
- [24] Olivier Faugeras. Variational principles, surface evolution, pde's, level set methods, and the stereo...(compendex). 1998.
- [25] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.
- [26] P. Fua and Y. Leclerc. Object-centered surface reconstruction: combining multi-image stereo shading. In *Image Understanding Workshop*, pages 1097–1120. Defense Advanced Research Projects Agency, Software and Intelligent Systems Office, April 1993.

- [27] Andrea Fusiello. Elements of geometric computer vision, 2006.
- [28] Bounds D. G. New optimization methods from physics and biology. 1987.
- [29] P. H. S. Torr G. Vogiatzis and R. Cipolla. Multi-view stereo via volumetric graph-cuts. April 2005.
- [30] Paolo Favaro George Vogiatzis and Roberto Cipolla. Using frontier points to recover shape, reflectance and illumination. 2005.
- [31] Marsha Jo Hannah. *Computer matching of areas in stereo images*. PhD thesis, 1974.
- [32] C. J. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. 4th Alvey Vision Conf.*, pages 147–151, Manchester, 1988.
- [33] R. Hartley. In defence of the 8-point algorithm. In *In Proceedings of the 5th International Conference on Computer Vision*, pages 1064–1070, Cambridge, Massachusetts, USA, 1995.
- [34] A. Hertzmann and S.M. Seitz. Shape and materials by example: a photometric stereo approach, 2003.
- [35] B.K.P. Horn. *Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View*. PhD thesis, 1970.
- [36] J. N. Thomson J. G. White, E. Southgate and S. Brenner. The structure of the nervous system of the nematode *caenorhabditis elegans*. 1986.
- [37] Hong kai Stanley and Osher Ronald Fedkiw. Fast surface reconstruction using the level set method, May 02 2001.
- [38] S. Kirkpatrick and E. Stoll. A very fast shift-register sequence random number generator. *Journal of Computational Physics*, 40:517–526, 1981.
- [39] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving, September 25 1998.
- [40] Kyros Kutulakos. Multi-view stereo datasets.
- [41] P. J. M. van Laarhoven and E. H. L. Aarts. Simulated annealing: Theory and applications, 1987.
- [42] Kurt Akeley Mark segal. The opengl graphics system: A specification. October 2004.
- [43] Max McGuire. The half-edge data structure, 2000.

- [44] K. H. Møller. 3D object modelling via registration of stereo range data. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2006. Supervised Jens Michael Carstensen, and co-supervisor Henrik Aanæs, IMM.
- [45] D. Morris and T. Kanade. Image-consistent surface triangulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-00)*, pages 332–338, Los Alamitos, June 13–15 2000. IEEE.
- [46] S.R. Lockery N.a. Dunn, J.S. Conery. Simulating neural networks for spatial orientation in c.elegans. 2002.
- [47] Jean-Philippe Pons, Renaud Keriven, and Olivier Faugeras. Modelling dynamic scenes by registering multi-view image sequences. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 822–827, Washington, DC, USA, 2005. IEEE Computer Society.
- [48] Art Pope. Vista.
- [49] M. Ruo Zhang; Ping-Sing Tsai; Cryer, J.E.; Shah. Shape-from-shading: a survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21:690 – 706, 1999.
- [50] Dimitris Samaras, Hong Qin, Liu Yang, and Ye Duan. Shape reconstruction from 3D and 2D data, June 15 2004.
- [51] Daniel Scharstein. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, November 06 2001.
- [52] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring, 1997.
- [53] P. Sturm and B. Triggs. A factorization based algorithm for multi-image projective structure and motion. In *In Proc.. of the European Conference on Computer Vision*, pages 709–720, 1996.
- [54] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography - a factorization method. *International Journal of computer Vision*, 9(2), pages 137–154, 1992.
- [55] Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000.

- [56] Greg Turk and Peter Lindstrom. Image-driven simplification, January 04 2000.
- [57] George Vogiatzis, Philip Torr, and Roberto Cipolla. Bayesian stochastic mesh optimisation for 3D. Technical report, November 23 2003.
- [58] R. Yakimovsky, Y.; Cunningham. A system for extracting three-dimensional measurements from a stereo pair of tv cameras.
- [59] O. Faugeras Z. Zhang, R. Deriche and Q.-T. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence Journal*, Vol.78:pp.87–119, 1995.
- [60] Li Zhang and Steven M. Seitz. Image-based multiresolution shape recovery by surface deformation. volume 4309, pages 51–61. SPIE, 2000.

List of Figures

2.1	A so-called binocular view of a Manhattan scene from approximately 1910. The image is taken from [4].	8
4.1	The epipolar geometry in a stereo setup	16
4.2	The Harris corner detector used on a cactus being upside down for reasons being clarified in Chapter 11.	17
4.3	Illustration of the baseline and depth in a stereo setup. setup. . .	20
5.1	One of the results of space carving by Broadhurst, Drummond and Cipolla in [6].	26
5.2	One of the original (left) images and the result (right) using frontier points, taken from [30].	28
7.1	Figure showing the famous Stanford bunny textured with a brick texture, and three cameras from different viewpoints. The rightmost camera is projecting its image onto the bunny, and the two other cameras show their perception of the scene. The breast and part of the head is not visible from the rightmost viewpoint, and thus the reprojections sees these areas as occluded (red and yellow).	35

7.2	A figure of the workings of a lenticular print describing a problem that can occur in the algorithm. All the blue surfaces can only be seen from the blue camera and visa versa.	41
7.3	Figure showing the different possible compositions when comparing an image I_i to a reprojection I_k . n means that the area is not defined for that image, a means that it is defined and o means that it contains an object, but is not defined.	42
8.1	Example of a model trapped at a local minima. To continue the descent the local maxima must be bypassed.	44
8.2	Figure of the relationship between $p(\mathbf{M}' \mathbf{I})$, $p(\mathbf{M} \mathbf{I})$ and μ . \mathbf{M}' is chosen over \mathbf{M} with probability μ	46
8.3	The three annealing schedules	50
9.1	Figure showing a triangle connecting three vertices. Notice that because of the righthand rule, the triangle is outwards facing. . .	53
9.2	Figure showing how the angular smoothness term are invariant to the division of triangles and edges.	55
9.3	Illustration showing the swap deformation.	59
9.4	Illustration of an edge collapse deformation.	60
9.5	Illustration of a split deformation.	62
9.6	Figure of the flow, when rolling back the deformation buffer. Every deformation is matched by its inverse.	63
10.1	Overview of the program structure showing the major elements. .	67
10.2	A screen shot from the program showing 8 narrow angled Camera 's displaying their input image, and an initial mesh.	69
10.3	Overview of the memory design of the mesh.	76
10.4	A sketch of the evaluation of the three proposed methods for evaluating a view.	78

10.5	Figure of the summing algorithm using shaders.	79
10.6	Example of the buffers used evaluating a view. The black and white checkerboard pattern in the snapshot means that the alpha channel is 0, while the red and yellow checkerboard pattern is occluded areas. The small figure of the bunny in the lower left corner of the comparison buffer is the effect of the summation, that reuses this texture to save space.	79
10.7	Figure showing the projective error due to the interpolation in texture space and not in world space. The red curve shows how the line would be perceived after the projection.	81
10.8	An example of the projective error. From left, is the original texture, a square made of 2 triangles on which the texture is projected from a Camera at a skew angle, and right the same square subdivided sufficiently to minimize the projective error. The bad quality of the projections is an unavoidable effect of the reprojection.	82
10.9	Figure illustrating the depth error in 10.9(a) in 2 dimensions, 10.9(b) 3 dimensions and 10.9(c) an example from the program, where the light gray is occluded pixels.	83
11.1	Simple example of a dataset of the Box.	89
11.2	A real image of the bunny, and two rendered images of the the model.	89
11.3	The left image is the texture used in this text, and the right is textured Stanford bunny.	90
11.4	Image 0, 1 and 2 of the cactus dataset, the annotated mesh and an overview of the distribution of the Camera 's.	91
11.5	Image 1, 2 and 3 of the gargoyle dataset together with the hand annotated mesh and an screen shot from the program showing the setup of the scene.	92
11.6	Image 1, 2 and 3 of the pot dataset, an overview of the composition of the scene and the annotated mesh.	93

11.7	The left and right image of the face dataset. Below the annotated mesh are shown together with a screen shot showing the setup. .	94
11.8	Image 5, 7 and 10 of the Masters Lodge dataset, the setup and the annotated mesh.	95
12.1	Screen shot of the initial composition of scene in the simple box test.	99
12.2	The objective function versus the number of iteration used for the Box test.	99
12.3	The first row shows the buffers of the initial situation. The second row shows the result after convergence. The resulting model can be found in <code>Box/test1/model1.x3d</code>	100
12.4	The the size of triangles versus the image cost for the box.	101
12.5	The initial composition of the scene using the Stanford bunny as an object	102
12.6	The buffers of the initial situation, and the result after convergence of the Bunny test. The resulting model can be found in <code>Bunny/test2/model1.x3d</code>	103
12.7	The objective function versus the number of iterations used in the bunny test.	104
12.8	The distribution of the deformations in the first 200 proposals of the Bunny test.	105
12.9	The first row shows the initial situation. The second row shows the result after convergence. The resulting model can be found in <code>Pot/test1/model1.x3d</code>	107
12.10	The objective function versus the number of proposals used.	107
12.11	The convergence using SSE and CORR, both controlled by itself and the other. The upper row shows the result from the original Pot dataset, while the lower shows the result on a normalized version of the same dataset.	108

12.12	The first row shows the initial situation. The second row shows the result after convergence. The resulting model can be found in <code>MastersLodge/test2/model18.x3d</code>	110
12.13	110
12.14	The first row shows the initial situation of the Gargoyle test. The second row shows the result after convergence. The resulting model can be found in <code>Gargoyle/test2/model11.x3d</code>	112
12.15	Left is the objective function when using either the best proposals of Bayes selection. Right is a histogram showing the acceptance in the same test.	113
12.16	The first row shows the initial situation. The second row shows the result after convergence. The resulting model can be found in <code>Cactus/test1/model11.x3d</code>	114
12.17	115
12.18	Two plots of the image metric, when capturing using blending or all pairs. The first plot is controlled by the image metric resulting from using pairs of images, while the other is controlled by blending	115
12.19	The initial buffers and the result from the Face test. The resulting model can be found in <code>FaceDataKHM/test2/model11.x3d</code>	116
13.1	A screen shot of a model the Bunny achieved using the algorithm multiple times. The screen shot shows areas where the model has captured the details nicely and areas, where the surface is distorted. The X3D model can be found in <code>Bunny/curly.x3d</code> . .	118
14.1	Figure of a group of vertices trapped in a local minima. The red arrows shows a possible, but improbable escape from the minima.	122
15.1	Two screenshots of a piece of the cactus dataset, where the patterns on one of the cactus' has been shifted. The left shows the cactus from an angle visualizing the shift, while the right is from an angle similar to the one of the <code>Camera</code> 's.	126

15.2	A pyramid from the Cactus dataset. For every level of detail, the model is refined.	126
15.3	The result of using the zooming approach on the Cactus dataset. The buffers are shown in chronological order, i.e. from lod_6 to lod_0 . The resulting model can be found in <code>Cactus/zoom2/zoom0.x3d</code> .	129
15.4	The result of using the zooming approach on the Pot dataset. The error buffer is blue because CORR is used, which does not visualize well. The resulting model can be found in <code>Pot/zoom1/zoom0.x3d</code> .	130
16.1	Features in a Cactus Image	132
16.2	The result of one of the tests using feature cost, where $w = 30$. The resulting model can be found in <code>Cactus/feature30/model11.x3d</code> .	133
16.3	The convergence of 5 different tests weighing the feature cost different. To the right the actual feature cost is shown.	134
17.1	The automatic sampling and evaluation of triangles arranged in a texture. The actual triangles and their internal error can be made out.	136
17.2	Illustration of the selection using cost queues.	137
18.1	One of the images of the gargoyles dataset showing that there is a hole in the statuette under the arm.	143
18.2	An example of Perlin noise.	143
B.1	The file structure of the CD	170

List of Tables

10.1	The extension used in the implementation.	71
10.2	The constants used in the implementation together with a small description and the default value.	85
12.1	Bunny test run without auto adjusted constants.	104
12.2	Test run with auto adjusted constants	106
A.1	The shortcuts for <code>capture.exe</code>	166
A.2	The shortcuts for <code>main.exe</code>	167
A.3	Options and parameters in a job description file.	168

Part V

Appendix

A P P E N D I X A

User guide

In this chapter the installation and use of the two program files will be explained. All files mentioned here can be found on the CD in the **Install** directory. As the implementation is now, it only works under Microsoft windows. It is a requirement that the system has a relatively new graphics card and that new drivers for OpenGL is installed.

A.1 Installation

To install the program, all files must be copied to a user directory. From there the two bat files can be run to test the installation.

A.2 `capture.exe`

`capture.exe` is a small program for capturing datasets. It only takes a single argument which is a path to a model to load. This must be in either `.txt`, `.x3d` or `.ms3d` format. When started the arrows can be used for navigation around the center and 'page up' and 'page down' can be used to zoom. Further more

short-cut	Description
'c'	Captures an image of the current view, and adds the corresponding camera parameters to the job file.
'.'	Decreases the resolution of the output images.
'_'	Increases the resolution of the output images.

Table A.1: The shortcuts for `capture.exe`.

the view can be tilted using 'Home' and 'End'. The images captured are taken from the current viewing angle, thus it is similar to what can be seen on the screen. Please refer to table A.1 for the available shortcuts. Some information are displayed in the window.

A.3 main.exe

`main.exe` is the main program for doing surface estimation. The input is a job description as will be described later. If successfully loaded, the scene with the initial mesh is displayed. This can be navigated like in `capture.exe`. A large amount of shortcuts can be used to control the actual convergence, which can be started there. These shortcuts are listed in table A.2. Information are displayed onscreen.

A.3.1 Job description file

The job description file describes the job to be done. It can be fully automatic, thus it starts the algorithm right away, without using resources on displaying anything and saves the result. In the job description file, '#' in the start of a line ignores the full line. If an error is made, the program will tell and stop. Pathes in the file is either relative to the `main.exe` or full paths.

The main purpose of the job description file is to give the paths for images and the camera parameters. The images are described together with all the other options and parameters in table A.3, however the camera parameters are a little special. They must follow this example form:

short-cut	Description
<i>Main functionality</i>	
's'	Starts the algorithm with the current options.
'z'	Decreases the level of detail used.
'x'	Increases the level of detail used.
'w'	Jumps to window form.
'f'	Goes to full screen.
ctrl-'s'	save the current buffers.
ctrl-'z'	Starts a zoom deform.
0-9	Sets the projecting cameras.
shift-0-9	Shows the buffers of the cameras.
'c'	Evaluates the current view.
'd'	Divides all triangles into 4 triangles.
'p'	Render point cloud.
'l'	Render wireframe.
't'	Render triangles.
<i>Options and parameters</i>	
F2	Enables option menu. Can change parameters by moving up and down and back and forth. The menu is closed if nothing is done for 2 seconds.
F4	Enables/disables auto-divide.
F5	Enables/disables update of the screen during algorithm.
F6	Enables/disables the rendering of the cameras.
F8	Changes capture mode between pairing and blending.
F9	Switches between the contents viewed in a camera. This can be one of the buffers or nothing.
<i>Test functionality</i>	
'q'	Deforms all vertices at random.
'e'	Random epipolar deform.
'r'	Random edge collapse.
'h'	Random edge split.
'g'	Random Gradient descent deform.
'j'	Random Swap.
'k'	Random random vertex deform.
'n'	Random normal deform.
'o'	Selects a deform at random, and discards if worse than last.
'v'	View vertex information. Very slow on large meshes.
space	Select a random vertex.
backspace	Rollback a single deformation.
'a'	Rollback all deformations.
Enter	Clear rollback buffer.
F1	Run mesh check test.
F3	Start memory monitoring.
F6	Prints the mesh to the console.

Table A.2: The shortcuts for main.exe.

short-cut	Description
'c'	Captures an image of the current view, and adds the corresponding camera parameters to the job file.
'.'	Decreases the resolution of the output images.
'_'	Increases the resolution of the output images.

Table A.3: Options and parameters in a job description file.

```
cam[id].matrix=    [0.912936 0.000000 0.766044 0.000000 ]
                   [0.262003 1.119882 -0.312243 0.000000 ]
                   [0.604023 -0.342020 -0.719846 12.582236 ]
```

, where [id] should be substituted with an id number.

A P P E N D I X B

CD - Thesis Data

The file structure of the accompanying CD can be seen in figure B.1. The CD contains the following main elements:

- Source code for the program
- Source text for the report
- Datasets used
- Models
- Installation
- This report in `.ps` and `.pdf` format

The X3D models stored on the CD, can be opened using any X3D compliant browser. An example of a free X3D browser is *Octaga Player* which can be downloaded from <http://www.octaga.com/>.

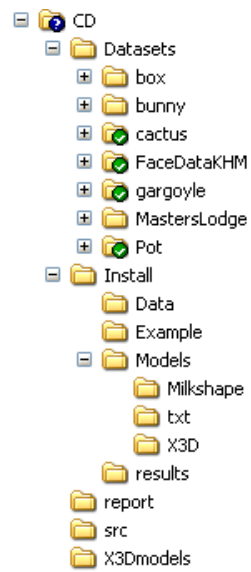


Figure B.1: The file structure of the CD