Aalborg University Esbjerg
Department for Software, Media and Electronics
Niels Bohrs Vej 8
DK-6700 Esbjerg
Denmark

Summer 2008

Master's Thesis
for
Master of Science in Computer Engineering

Extraction of 3D Structure using Texture Flow

Ole Thomsen Buus

May 30th 2008

supervisor: Jens Arnspang

Abstract

This report investigates the application of texture flow for direct estimation of local three-dimensional structure of a rigid textured surface moving relative to a fixed perspective camera. Texture flow is defined here as a spatiotemporal gradient, based on the estimation of local image texture densities and its derivatives. A major part of the work deals primarily with computational experiments. A simulation framework was developed for exact simulation of local surface and image texture density. An experimental setup was also built and experimentation with a digital camera and a movable textured surface was attempted.

The conclusion is two-fold: (1) In the computational experiments, it was indeed found possible to directly estimate local surface orientation and depth using the estimated texture flow derivatives. Sometimes also for velocities (ego-motion). (2) The camera experiments need further work. The estimation of texture density in real digital images is difficult and the methods chosen for texel segmentation showed to be highly sensitive to the lighting conditions.

Informative

Project title:	*Extraction of 3D Structure using Texture Flow*
Project period:	November 2007 – June 2008
Report hand-in date:	May 30th 2008
Author:	Ole Thomsen Buus
Supervisor:	Jens Arnspang
Enclosed material:	1 CD-ROM

Signature

Ole Thomsen Buus

Table of Contents

List of Figures

List of Tables

Preface

This report is my Master's Thesis. It deals with a specific problem within the general area of 3D computer vision. This problem seeks to estimate three-dimensional shape and structure using the "*texture density*" in two-dimensional digital image sequences.

A project website has been created to support this written report:

`http://hardbytes.web.surftown.dk/textureflow/`

The projects present a new method for estimating surface shape using monocular view only – known in the report as the "*primary method*"[1]. The primary method is a combination of methods from shape from texture and structure from motion. The report is written toward students and scientists at my own level of education and experience.

With an emphasis on texture and texture density, the project investigates the primary method by applying two kinds of experimentation: 1) synthetic *computational* experiments, and 2) non-synthetic *camera* experiments.

Overview

The report consists of a total of *five* chapters. Each chapter has its own primary area of focus.

Chapter 1 is an introduction to the problem area and the primary method. The notions of texture and texture density are introduced and discussed. The chapter also takes a look at some previous work in the area of shape from texture.

Chapter 2 takes a detailed look at the the primary method. It introduces and derives the basic formalisms behind it and discusses shortly, how one is supposed to make sense of it all. A short chapter which primary agenda is is to prepare the reader for the next two chapters.

[1] Other names would be suitable in more general settings; for instance, "the texture density method" or "the texture flow method" etc.

Chapter 3 focuses on much of the work carried out in relevance to the computational experiments. It introduces the structure and data flow of a mathematical framework built using the Mathematica computer algebra system. The concept of abstract texture density is also introduced. At the end the chapter presents the actual results from the experiments.

Chapter 4 presents the methods applied in the camera experiments. The experimental setup built for the camera experiments is introduced and challenges faced in the texture analysis of real digital images are also discussed. A possible method for estimating image texture density in synthetic texture is presented.

Chapter 5 This chapter concludes the report and summarizes the major results achieved in this work. Some final remarks are made on the possible limitations of the primary method.

Acknowledgments

When writing the acknowledgments — something which I have not had the pleasure to do in other previous work — you know that you are almost done. The main text has been written, corrected, and corrected again. A cup of coffee is now waiting. At this points, you sit down and acknowledge that you certainly did not get through this without the help of others. Also, I might say this right away, should I forget to mention you here, I am sorry.

A good project can seem difficult to find nowadays. My supervisor, Jens Arnspang, was kind enough to propose that I continue work of his own. It was also a trilling learning experience to have the support a supervisor which had an, almost intimate, understanding of the thoughts behind the general formalisms.

Other staff at the university certainly also deserves to be mentioned. Britta Jensen was always ready to help, when in need of information or material. Other staff were also pillars of support in the beginning and throughout the project, especially in relation to designing and building the experimental setup. Carsten Andreasen, head of the mechanics engineering department, was an extremely helpful support. He was never short of good ideas. He also did much of the work in assembling the experimental setup.

Also, Jørgen Houe Pedersen and Jens Madsen Houlrik were both very important in the early stages of the camera experiments. It was no problem to borrow equipment, even of the most delicate kind. Finally, Jens Jørgen Jensen, university caretaker, was always ready to help, even with the most unconventional requests.

Ole Thomsen Buus
Aalborg University Esbjerg, May 2008

1. Introduction

This chapter is a gentle introduction into the problem area of this report. Section 1.1 takes a look at the problem at hand and introduces new method for estimating 3D structure from a 2D image sequence. Section 1.2 introduces some fundamental concepts and cites some related previous work from the computer vision literature – both classic and modern. Section 1.3 describes the formal project methodology, it argues how experimentation is used and presents the goals for the project.

1.1. Subject and Motivation

It seems remarkable how easy it can be to navigate in a public building where you have never set foot before. You know by experience that most public buildings are equipped with some type of navigation-aid in the form of signs. Finding room A332 is not really a problem if a map is available in the lobby and corridors and rooms are marked accordingly. Of course you can make mistakes, you can have a bad day, or the signs can be wrong. But generally humans are good at quickly interpreting and understanding a new unknown environment.

The ability of humans beings to quickly *interpret*, *understand*, and *use* their immediate environment is a very sophisticated and highly useful "active sensing" ability. This ability is made possible by many different senses of the human body, for instance, *vision*, *hearing*, *smell*, *touch*, *locomotion* and also *memory*. Human vision is especially important in this context since this is the sense which makes *visual perception* possible. An automated system, say a robot, would be quite useful could it *mimic* visual perception – and thus the goal becomes *automated visual perception*, also known as *computational* vision, which is the general topic of this report.

A functional automated visual perception system would very useful for many situations where decisions can be made on the basis of *visual inspection*. A simple example is visual inspection of products on a conveyor belt in a factory. In such an industrial setting one has a chance to control many aspects of the visual scene, even down to the lighting. Automated industrial visual inspection is thus perhaps a more realistic goal (though not a trivial one) and some solutions already exist today.

Another example is *automated navigation* where the ability to automatically construct a 3D model of the surroundings becomes very important. This problem is considerably

more complex than inspection in a controlled setting. In the last few years though, many advances have been made and the research have matured, even to the point of commercialization. Today it seem possible that autonomous vehicles could be part of the near future.

In a way they already are. The DARPA[1] Urban Challenge 2007 took place in November 2007 at a closed air force base in California, United States. It was a professional *autonomous vehicle* contest among eleven teams. The goal was to finish a 96 kilometer long *urban area* course, using nothing but on board visual and/or range sensors, and real time computer processing of the surrounding environment.

The 2 million dollar prize winner was the Tartan Racing team from the Carnegie Mellon University, Pennsylvania, United States. Their vehicle, the "Boss", were able to, fully autonomously, complete the full course in a little over 4 hours at an average speed of 23 km/h [22]. This is just one example of what could be considered the "peak" of current research in computer vision. Traffic safety is a concern which car manufacturing companies hope can be improved by equipping cars with "smart" capabilities. On board computer systems might be able to react faster than humans when needed, or, if necessary, be able to drive the car if the driver, for instance, suddenly looses consciousness.

The overall goal of this report is *not* to address any specific *end usage*. Instead it deals with solving a specific computer scientific problem which is crucial to the implementation of automated visual perception systems. This report presents a new method for solving the problem of deriving, from a 2D image sequence, the 3D structure of surfaces of 3D objects within the imaged scene. Such information are cues believed to be crucial to humans for correct interpretation of *3D space*. The formalisms behind this *primary method* were originally published by Arnspang [2] in 1991. Since then, many sophisticated and useful methods for estimating 3D world structure from 2D images such as velocities and shape, have appeared.

The primary method is yet another method for doing just that, but the main formalism behind it, a geometric relationship between *image* and *surface texture densities*, seems to have been overlooked in work published since then. What Arnspang showed in 1991, and what the results of this report show, is that texture density has the potential for being a useful texture gradient. Extracting this gradient is not easy though and Section 1.2 will have more to say about this.

Chapter 3 and 4 will each introduce, explain, and present results from computational and real camera experiments of moving textures. The goal with such experiments is to see whether or not the primary method is useful or not, and where further work should be focused.

[1] *Defense Advanced Research Projects Agency*.

1.1.1. The Problem

At least two areas of research are directly related to the primary method: *Structure from motion* and *shape from texture*. Structure from motion seeks to estimate 3D *structure* including surface-shape but also *translational* velocities and the *rotational* velocity of 3D points, lines, curves, and/or entire surfaces. To say anything about velocity the concept of motion is necessary and thus a sequence of digital images is needed. The sampling interval of the sequence should generally be short enough, such that the apparent movement of objects cover the span of only a few pixels in each frame.

The problem of shape from texture is very much related to structure from motion and it is natural to combine methods from both problems. But there are also some major differences. Shape from texture seeks to determine only surface shape (no velocities) relying only on a single image (as opposed to an image sequence) of a static 3D scene which includes one or more textured surfaces.

The basic idea behind shape from texture, is to take advantage of the visual distortion in the geometry of the texture when observed in a perspective image. This particular form of distortion is also known as *projective distortion*. Given that the distortion in the image of the texture is due to actual perspective projection, i.e. that some projective distortion is not mimicked by the texture itself (or even canceled), it can be assumed that the *variation* of distortion, the concept of a *texture gradient*, is what provides direct information about the shape of the underlying textured surface.

Texture is a complex concept not easily defined. In this context "texture" is informally defined to be some kind of visual *pattern* that lies on the surface of some object. This pattern is said to consist of smaller *texture elements* or *"texels"*. A more formal definition would distinguish between *surface texture* and *image texture* (this definition will be formally introduced in Section 1.2). A surface texture is the texture on the surface – with no distortion. The image texture is the image of the surface texture with projective distortion.

Figure 1.1 [2] shows a textured surface, or perhaps just a surface texture with some form of distortion. How to know? The important fact remains: We humans perceptually recognize that the image texture seems to be "painted" on an underlying smoothly curved surface. A question also remains: How are we able to interpret shape from distortions of texture elements? In this case the only elements are black circles. What cues do the human brain extract from Figure 1.1 and how does it process these cues and gain the sensation of an underlying surface?

Such questions are very often discussed, not only in a purely computer scientific setting, but also within the fields of experimental psychology – in particular the field of *psychophysics*. The work by Gibson [6] is probably the earliest work which argues

[2] Reprinted from [19], with permission from Elsevier Limited.

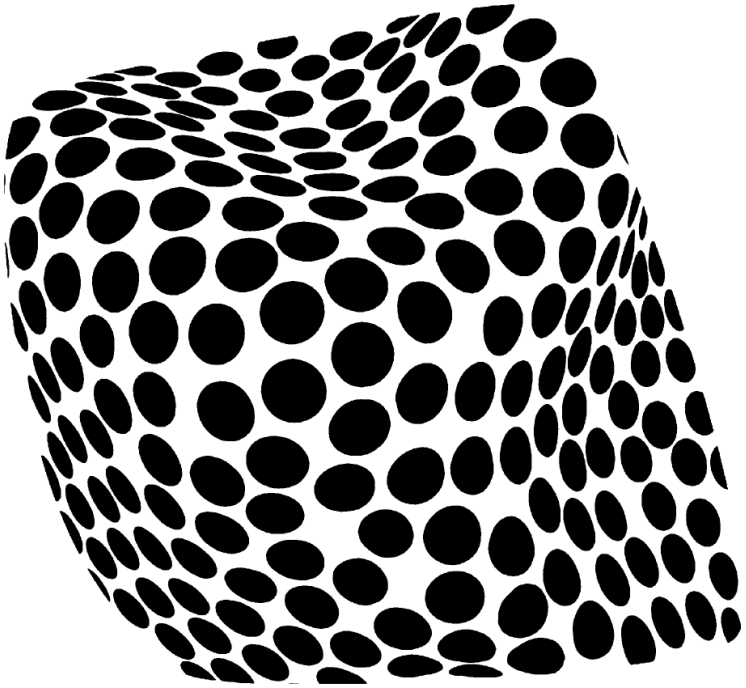


Figure 1.1.: A texture with distortion which gives the perception of a smooth curved surface.

that perception of *space* is comparable to the perception of visual surfaces. He also argued that the *sensation* of space (or shape) are due to the impressions[3] of surface and edge.

Gibson believed that humans visually senses shape and space due to an automatic "reading" of the variations of texture, when seen from different angles. The measures of variation, though usally only formally defined, are generally known as *texture gradients*. One definition of a texture gradient is the *texture density*, which is also used in the primary method. Texture density is (informally) defined as the number of texels per surface area. Again, according to Gibson, texture density is an important visual cue for perception of depth. The concept of texture and texture density will be revisited in Section 1.2.

1.1.2. A Solution

A more detailed explanation of the primary method will be postponed until Chapter 2. Instead this chapter will introduce the basic fundamentals and assumptions used in it. Section 1.2 will elaborate on the concepts of texture, texture density, and shape from texture as well as refer to some important previous work.

[3] Psychological concepts such as "sensation" and "impressions" shall remain undefined here.

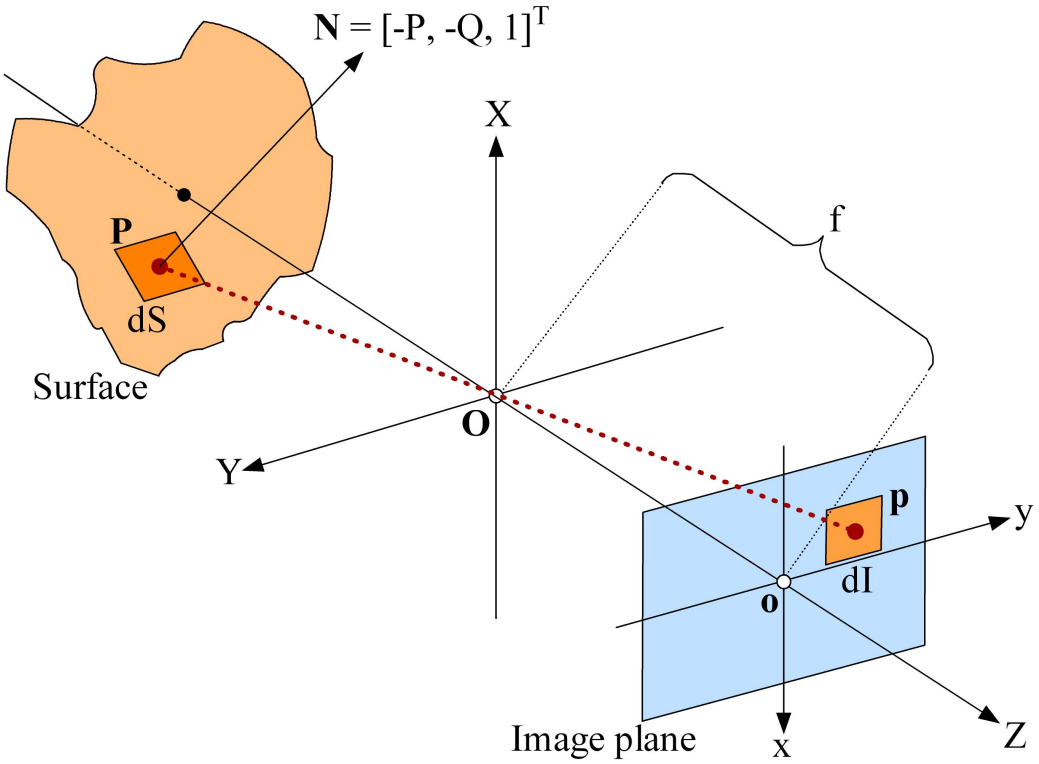


Figure 1.2.: A simple model of perspective projection – the pinhole camera-model.

The primary method is defined using a simple model of perspective projection – the *pinhole camera model*. Figure 1.2 shows this model in three dimensions. This model simplifies the relationships between *world coordinates* (on the surface) and *image coordinates* (on the image plane). Two coordinate systems are used: A *camera reference* (X, Y, Z)-system (with origin in $\mathbf{O} = [0, 0, 0]^T$) and an *image reference* (x, y)-system (with origin in $\mathbf{o} = [0, 0]^T$). Coordinates of the image reference system can always be represented relative to $\mathbf{O}$ by setting the third Z-component to f, the *focal length* of the camera (for instance $\mathbf{o} = [0, 0, f]^T$).

The pinhole camera-model (or pinhole-model) offers a simplified look at the physical image acquisition process. It only contains the geometrical aspects of image formation – which is why it is sometimes known as *geometric image formation*.

A simple geometric relationship between coordinates in the camera reference system and the image reference system is derived using the concept of *similar triangles*:

$$fX + xZ = 0, \qquad fY + yZ = 0 \tag{1.1}$$

which leads to the relations

$$x = -f\frac{X}{Z}, \qquad y = -f\frac{Y}{Z} \tag{1.2}$$

To just partially explain the physical process of image formation, concepts from *radiometry* are needed. Radiometry is the essential part of image formation concerned with the relation among the amounts of *light energy* emitted form light sources, reflected from surfaces, and registered by sensors (for instance a CCD sensor in a digital camera).

In the pinhole-model, rays of light are represented as lines in space and there is no concept of light energy involved (all light goes through a very small hole – a pinhole). To incorporate such physical quantities, one can use the simplest model of optical image formation, known as the *thin lens* model. Two important concepts from radiometry used in the thin lens model, are the *image irradiance* and *scene radiance*. Image irradiance is the power of the light, per *unit area* and at each point **p** of the image plane. The scene radiance is the power of the light, per unit area, emitted by each point **P** on the surface (see, for instance, Trucco and Verri [20]).

The pinhole-model, as it is used in the current context, actually incorporates some of these concepts. Two small unit areas, dS and dI, are shown in Figure 1.2 on the surface and the image plane, respectively. The idea comes from radiometry where both scene radiance and image irradiance are localized to *unit area* dS and dI, respectively. These unit areas are considered locally planar and they have a *normal vector* $\mathbf{N} = [-P, -Q, 1]^T$. P and Q are *gradient space* components and they define the orientation of the local plane of each surface point **P** (or the entire surface should it be a plane).

The primary method takes advantage of a fundamental geometric relationship between these two unit areas. This relationship is the ratio dS/dI and it can be derived using the concept of *solid angles* i.e. angles in three dimensions (Chapter 2 shows the full derivation).

Figure 1.3 shows another view of the pinhole-model with some more detail. The symbol Ω defines the angle between the normal vector and the line-of-sight[4] and β is the angle between the line-of-sight and the *optical axis*[5]. By using these two fundamental angles, the following ratio between dS and dI can be defined:

$$\frac{dI\cos\beta}{\left(\frac{f}{\cos\beta}\right)^2} = \frac{dS\cos\Omega}{\left(\frac{Z}{\cos\beta}\right)^2} \Leftrightarrow \frac{dS}{dI} = \frac{\cos\beta}{\cos\Omega}\left(\frac{Z}{f}\right)^2$$

As mentioned earlier, the primary method is based on a relationship between surface and image texture densities. Surface texture density $\mu_S \in \mathbb{R}$ is the number $\epsilon_S \in \mathbb{N}$ of texels on the surface per area dS: $\mu_S = \epsilon_S/dS$. Image texture density $\mu_I \in \mathbb{R}$ is the number $\epsilon_I \in \mathbb{N}$ of texels in the image per area dI: $\mu_I = \epsilon_I/dI$.

[4]The line between **P** and **p**.
[5]The Z-axis.

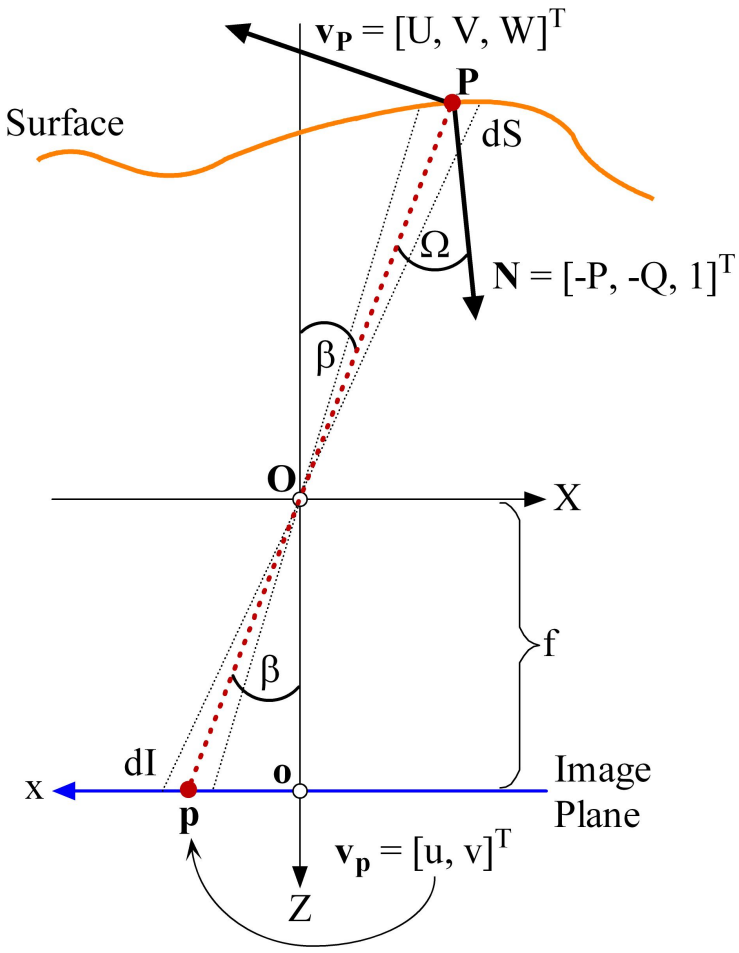


Figure 1.3.: Some important sizes and angles needed in the definition of the primary method.

A fundamental assumption about texture densities is made in the primary method (and also in other methods). If a point **P** in a locally planar area dS on surface Φ projects to a point **p** in a locally planar area dI on image plane Γ, one can assume that the two texel *cardinalities* ϵ_S and ϵ_I are equal: $\epsilon_S = \epsilon_I = \sigma$.

It is not hard to imagine circumstances where this assumption could be wrong, due to focusing errors, errors in the digital image acquisition, or simply dirt on the front lens. But the assumption *does* make sense in general. If the focus is correct and the texels themselves are visible and distinguishable in the image and on the surface it is a viable assumption.

Based on this assumption, it is now possible to formulate a relation between texture densities:

$$\frac{\sigma}{dI} = \frac{\sigma}{dS}\frac{dS}{dI} \Leftrightarrow \mu_I = \mu_S \frac{dS}{dI} = \mu_S \frac{Z^2}{f}\frac{\sqrt{P^2+Q^2+1}}{Px+Qy+f} \tag{1.3}$$

Equation 1.3 is derived using dot vector products in the pinhole model geometry. Related to this equation is also the *Image Irradiance Equation*: $E = kL$, where E is image irradiance, L is surface radiance, and k is a constant which depends only on current camera parameters (see Horn [8] Chapter 10). One motivation behind the primary method was to derive an equation which relates texture densities μ_S and μ_I instead of *photometric* densities E and L. Equation 1.3 is such a relation and it is known as the "Texture Density Equation". Importantly, this *texture density equation* contains P, Q, and Z; none of which are directly[6] present in the *image irradiance equation*.

It is possible to be a bit more formal about the texture densities which in fact are functions. Equation 1.3 can be rewritten as a relation between functions:

$$\frac{dS}{dI} = \frac{\mu_I(x(t), y(t), t)}{\mu_S(m, n)} = \tau(x, y, P, Q, Z) = \frac{Z^2}{f} \frac{\sqrt{P^2 + Q^2 + 1}}{Px + Qy + f} \tag{1.4}$$

Here m and n are *surface coordinates*, which means that they define locations *on* a surface, regardless of the orientation of the surface itself. It makes sense to define the surface texture density to be a function $\mu_S : \mathbb{R}^2 \mapsto \mathbb{R}$.

Note that μ_S is independent of time t. This is an important fact, and it does limit the range of situations with moving surfaces. For instance, the rotational velocities P' and Q' are 0. Such details are revisited later in Chapter 2.

In the computational experiments in chapter 3 things have been simplified such that $(m, n) \in \mathbb{Z}^2$ and $\mu_S : \mathbb{Z}^2 \mapsto \mathbb{R}$. The function $\tau(x, y, P, Q, Z)$ is also important and chapter 2 will have more to say about it.

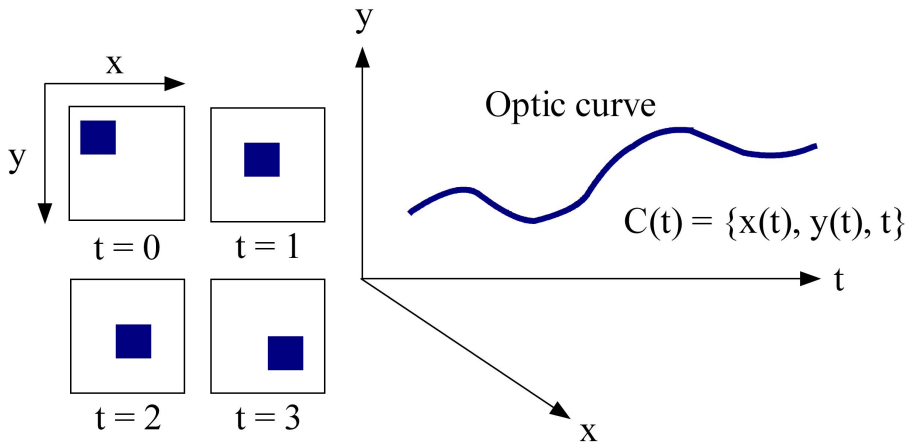


Figure 1.4.: An optical $C(x) = \{x(t), y(t), t\}$ curve in space.

[6] It is possible to introduce a *reflectance model*: $E = k\pi R(P, Q)$. But the *image irradiance equation* is not directly "born" with 3d structure variables present.

The primary method combines the cues from both texture and motion. As seen in Figure 1.3 a surface point **P** has a velocity vector $\mathbf{v_p} = [U, V, W]^T$. Figure 1.4 shows a curve $C(x) = \{x(t), y(t), t\}$ in space. It represents the "*image*" of a single surface "patch" over time. If an object moves over the span of a few images, it is fair to assume that groups of pixels (surface patches) with the same color moves with the object. The *optical curve* C follows such a surface patch trough (x, y, t)-space. The idea is also shown to the left in Figure 1.4 where four images at time t from 0 to 3 depict the movement of a group of black pixels.

By taking the total time derivative of the image texture density $\mu_I(x(t), y(t), t)$ along the optical curve C, using the texture density equation, it is possible to formulate nonlinear *constraint equations* which includes the variables for translational velocities (U,V,W), rotational velocities (P',Q'), surface orientation (P,Q), and depth Z (such equations are called "Texture Flow Equations"). Extraction of 3D structure using the primary method boils down to at least three subproblems which poses some fundamental algorithmic challenges: 1) Counting texels, 2) estimating derivatives in (x, y, t)-space and 3) solving the constraint equations. Chapter 3 and 4 will each have more say about how such challenges were attempted solved.

The concept of texture density and its time derivative along a space curve is very much related to optical flow and the problem of estimating it. Just as optical flow is a 2d vector field defining the *velocity* of *image intensities*, the primary method seeks to determine "velocities" of image texture densities - a 3d vector field. One major difference is that optical flow is not directly born with an analytical relationship between image and surface densities and what comes closest is the very general image irradiance equation mentioned earlier ($E = kL$). On the other hand the primary method is explicitly born with an analytical counterpart: The texture density equation (Equation 1.3).

Due to such similarities with the optical flow; Arnspang referred to the current method of determining *spatiotemporal* texture density measurements as "Texture Flow". In the meantime though, that exact term has received an entirely different meaning in more recent literature. Subsection 1.2.1 will have more to say about this new definition of texture flow.

This section took an introductory look at the problem and a proposed solution. The next section will present a short look at of some of the related fundamentals and some previous work.

1.2. Some Fundamentals and Previous Work

This section will introduce the reader to some fundamental concepts and while doing that, some previous work will be cited and some of it, if relevant, will be discussed

shortly. The concept of texture and texture density has not been explained to any full extent yet. Subsection 1.2.1 and 1.2.2 will do just that.

After this, Subsection 1.2.3 will take a brief look at shape from texture and cite some important previous work. Important in this context is more modern research related to shape from texture in general. A good example is the promising work by Loh [12].

1.2.1. Texture

It seems that is has been difficult to define the concept of optical texture in any general way. Though many have tried, it usually is enough to define texture in the context it is used. Texture is abundant in nature and they generally seem to consist of smaller elements, texture elements, or texels. These texels are seemingly *repeated* in a some form of structural pattern which is not always possible to define in any general way.

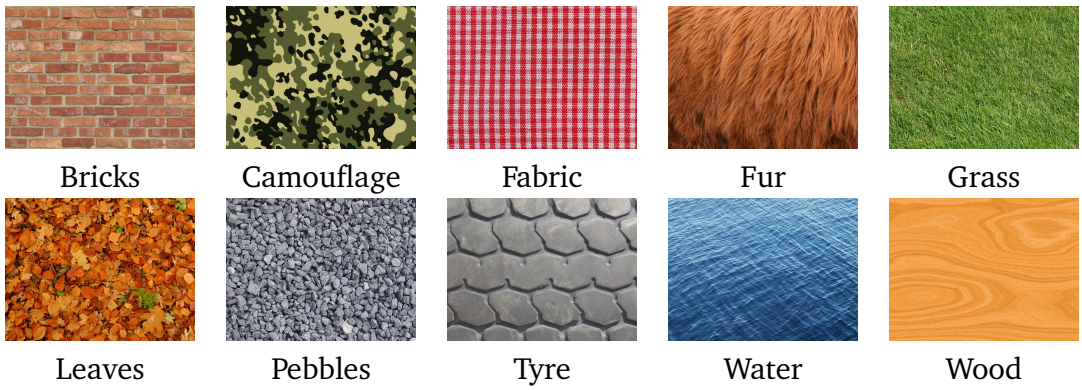


Figure 1.5.: Some examples of natural and man-made textures.

The images in Figure 1.5 represents some examples of texture; both *natural* texture (fur, grass, leaves, pebbles, water, wood) and *artificial* texture (bricks, camouflage, tyre) are shown. The difference between the two is seen in the way the texels are repeated. Natural textures are generally random, whereas artificial textures are often *deterministic* or *periodic*.

The field of *texture analysis* deals primarily with recognizing/classifying textured image regions using various measures of texture description. When seeking to extract certain information about a texture, at least three principle approaches can be used: Statistical, structural, and spectral. Any such measures of texture contents will be introduced in later chapters when and if applicable.

It seems crucial to find a useful definition of texture. Tuceryan and Jain [21] makes a compelling argument on the difficulty that exists in defining texture. They also come with a few examples on texture definitions created by various researchers. Generally

such definition incorporate various *discriptors* such as *smoothness*, *coarseness*, and *regularity*.

One useful definition of texture (and texels) could be the following definition (borrowed from Trucco and Verri [20]):

Texture and texel : A *surface texture* is created by the regular repetition of an element or pattern, called surface texel, on a surface. An *image texture* is the image of a surface texture, itself a repetition of *image texels*, the shape of which is distorted by the projection across the image. The idea of distinguishing between a surface and an in image texture was mentioned shortly in Subsection 1.1.1.

Such concepts need further explanation:

Surface texture : A surface texture exists only on a surface and it's most conformal view is when the surface is a plane. In this work a surface texture is not *self-occluding* which in the simplest case means that no texel cover parts of other texels. Each texel is perfectly visible and adjacent to other texels. This simplifying assumption makes the computational and camera experiments easier.

Image texture : Even if the surface texture is not self-occluding, the physical features of the surface could have that effect that texels seen in the image, are partially or fully occluded by other texels. This can be avoided in the experiments by limiting the surfaces to planes.

A crucial part of what we call texture is the apparent continuation of a spatial pattern in nature: The cats fur, a zebra's stripes, the leaves on a tree; all extremely difficult to quantify. In more recent research this visual phenomenon has become known as *texture flow*. This more modern concept of texture flow is very important to problems like texture description, segmentation, and synthesis (creation of artificial textures).

Though methods from these areas are of considerable importance to the extraction of information from texture, this new definition of texture flow seems to be very different from the definitions of the same term "texture flow", used in the primary method, as mentioned in Subsection 1.1.2. Though this is not attempt of an etymological discussion it seems prudent to take a look at the new use of texture flow.

The work by Ben-Shahar et al. [3] describe texture flow as a two-dimensional structure:

> Informally, texture flows are defined by their orientation content – a dense visual percept characterized by local parallelism and slowly varying dominant local orientation (almost everywhere). This class of patterns is common in both natural and man-made objects and, for centuries, it's been used by artists as a tool to convey both the shape and shading of smoothly varying surfaces and their discontinuities [3].

Ben-Shahar et al. formally define texture flow as an *orientation function* or a *vector field*, among others. In any case the work represent one way to mathematically address the complexity of texture flow, and it focuses on two main phenomenons: *local parallelism* and *slowly varying* dominant local orientation. In another related work, Tai et al. [18] are successful in estimating the texture flow, and as an example they synthesize a zebra's stripes in a natural image. Tai et al. define texture flow informally:

> Texture flow estimation is often targeted towards images of textured 3D surfaces in natural scene, where texture flow arises due to the geometric variation of non-planar surfaces or imaging under perspective. It is assumed that the underlying texture has some repeating structure that is varying in the image (i.e. distorted) in terms of scale and orientation. We called this distortion texture flow [18].

These informal descriptions lead to 2d vector field representations of texture flow, which are then used to either describe or synthesize textures, or both. The original meaning, as presented by Arnspang [2] is very different. Just as optical flow is a 2d vector field, the texture flow is a 3d vector field representing, for each (x, y, t)-point $\mathbf{p}$, the gradient $\nabla(\mu_I) = [\frac{\partial \mu_I}{\partial x}, \frac{\partial \mu_I}{\partial y}, \frac{\partial \mu_I}{\partial t}]^T$. This does not correspond with the new understanding of texture flow, and it is only fair that this report use the original definition of texture flow as applied in the primary method.

1.2.2. Texture Density

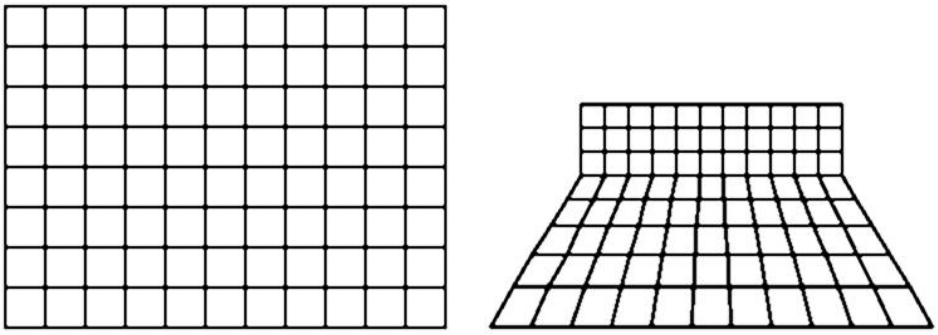
As mentioned earlier, Gibson argued that the perception of space, i.e. scene depth, was due to a increase of the density in texture elements. This concept has been used, for instance, in paintings for many years and thus it was a well known fact before Gibson wrote about it in the 1950's.

Figure 1.6 shows the the painting "*Paris Street; Rainy Day*" from 1877, painted by French impressionist painter Gustave Caillebotte (1848-1894). This is an example of the use of perspective concepts such as foreshortening and scaling in paintings. Objects in the foreground are slightly out of focus. In the mid-distance objects are sharpest (the carriage and some pedestrians) while object far away are painted very indistinct. This is also a good example on the use of texture density for the depiction of natural scenes.

What Gibson [7, 6] suggested, other than already mentioned, was that texels are *uniformly* distributed, in the sense that each unit area on a surface in the world contains close to the same number of texels. Such an assumption does not seem realistic when looking at natural texture; as mentioned earlier the texture density has a *gradient*. Gibson, also aware of this, proposed that humans perceive the *orientation* of naturally textured surfaces from a combination of concepts of *sameness* and *difference* which correspond to uniform density and the gradient of texture, respectively. When

Figure 1.6.: 1877 painting "*Paris Street; Rainy Day*" by Gustave Caillebotte. The painting shows a scene from Paris in late 19th century (image is in the public domain).

applying such assumptions today, it is said that one uses the "*Gibsonian approach*" [1].



No depth Perceptual depth

Figure 1.7.: An example of perceptual depth. Relative to each other the two images show how the perception of depth arises when *foreshortening* and *scaling* is applied (see text).

The graphic to the left in Figure 1.7 shows a conformal grid with no apparent depth

information (uniform density). The graphic to the right contains depth information due to how the grid-boxes are smaller and thus the density per. image area has increased. The *foreshortening* effect on the grid in the foreground results from the projective distortion that the right graphic contains. This creates the illusion of a *ground plane* and this can also be considered a cue for depth perception.

Perspective projection has two major effects on the resulting image. The first one is the foreshortening effect seen in Figure 1.7. This distortion depends on the angle between the (local) surface normal vector and the optical axis (the Z-axis). The other distortion is *scaling* and it depends on the distance between the surface and the image plane, i.e. depth. Increasing depth makes objects appear smaller. These effect are seen in Figure 1.7 (right graphic) where the "ground plane" grid-boxes are both foreshortened (angles between orthogonal lines are distorted) and scaled (grid-boxes get smaller with increasing depth).

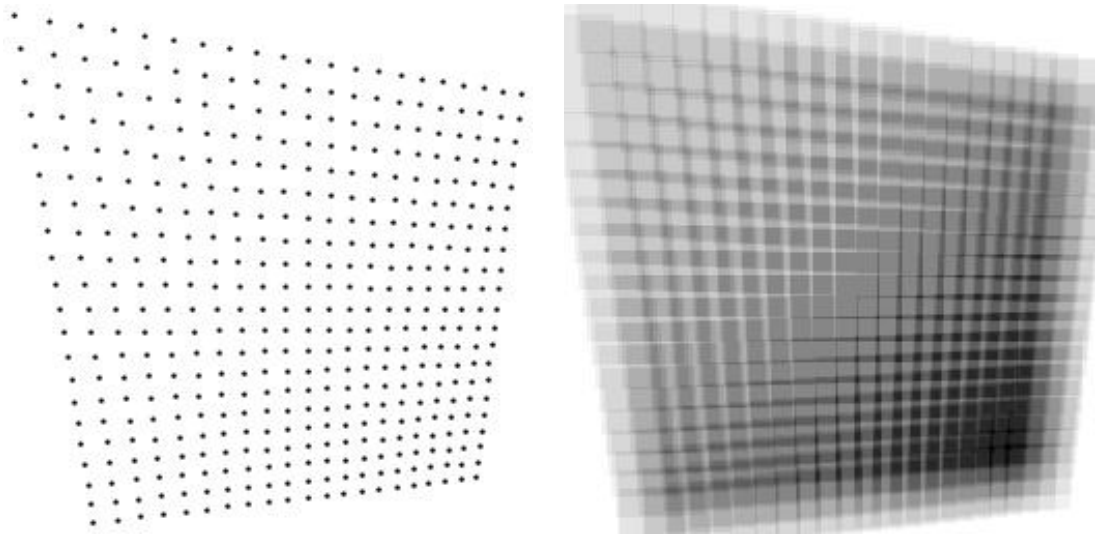
Another form of projection, orthographic, does not include the scaling distortion. This projection is used in, for instance, technical drawings and it can be seen as the perspective projection of points very far away, such that world points are projected parallel to the optical axis.

But how to *measure* texture density? How to represent it mathematically, perhaps as a function with domain and codomain? Kanatani and Chou [11] made and attempt in a formal definition of texture density $\rho(x, y)$ with *delta-function-like* singularities. The value of $\rho(x, y)$, which describes the amount of texture (σ) divided by the area it occupies (dS), is ∞ as texture elements (the exact area of 1 texture element at (x, y) is larger than dS) and 0 at features such as dots and lines. The treatment ends up in a rigorous mathematical definition of *dot* and *line* texture densities as *functionals*.

This work (the experiments in particular) will not attempt such an elaborate treatment of texture density. Texture density is in its most basic form a *binary* image operator. The foreground of the binary image corresponds to texels and the counting is done purely by connected-component-analysis using either 4- or 8-connectedness[7]. An even more simple approach, though perhaps naive in but the most simple synthetic textures, would be to create histograms by counting pixels of a certain grayscale intensity or color in a window.

Figure 1.8 shows an example of how texture density measures are to be interpreted. The image to the left in Figure 1.8 is a synthetic *dot texture*, 256×256 pixels in size. It is an image of a plane with gradient space parameters $P = 1$ and $Q = -1$, which means that the plane is "tilted" 45° angle with both the X- and Y-axis. The plane is made out of dots (441 of them) with the same distance to each other. But due to the orientation of the plane and perspective projection, the density of the dots in the image varies – it has a gradient.

[7] This can, for instance, easily be applied using the *bwlabel()*-function in the image processing toolkit in Matlab.



Synthetic dot texture Dot density plot

Figure 1.8.: Example of texture density measures on an image with dots (see text).

The right image of Figure 1.8 shows the result of applying a neighborhood operation to a binary version of the left image. This operation counts the number of connected components in a neighborhood of size 32×32 pixels, using 8-connectedness. White corresponds to the lowest density and black the highest density, which corresponds nicely with the left image.

1.2.3. Shape from Texture

Much have been said already about the shape from texture problem and its goals. Most approaches deals primarily with an analysis of the image texels or assumes some distribution of them.

Usually when dealing with texture as a source of information at least three basic assumptions can be made about that texture: *homogeneity*, *stationarity*, and *isotropy*. Homogeneity means even distribution of texels. The term refers to the *location* of texels, without considering the rotation of each texel. Stationarity means that texels differ by translation but not a rotation. Isotropy refers to the distribution of edge segments over all orientations in each texel. An isotropic texture is a texture with almost circular texels. An example of a highly non-isotropic texture would be a texture with *line-segments* as texels.

Important classic work in shape from texture and related problems, is those by Stevens [17, 16, 15], Witkin [23], Aloimonos [1], Kanatani and Chou [11], and Gaarding [5, 4], among others. Examples of more modern work which shows promising results are Loh [12], Loh and Hartley [13], and Loh and Kovesi [14].

Witkin's [23] and Steven's [16] work are important when compared to the primary method and its reliance on finding texels and counting them. Aloimonos [1] comments on their findings. A general algorithm for finding and counting texels in natural color textures, has yet to be developed.

Witkin's [23] proposed a statistical approach without assuming spatial homogeneity. Isotropy was expected and he was able to estimate surface orientation using orthographic projection. Witkin had two arguments on why not to continue with the Gibsonian approach: 1) is was unsure whether or not the uniform density assumption could be used as a basis for general surface structure extraction, and 2) that even if that were possible, the method (algorithm) would need to known about the texels, and finding texels in textures is not at all obvious.

Steven's [16] talks about the kind of information that one can expect to extract from texture gradients, and whether or not one is able to determine *what* exactly has been extracted. He found that texture density depends on both scaling and foreshortening, and because of this non-linear dependence he concluded that texture density is not a good measure for computing surface orientation.

Aloimonos [1] comments on both of these two works and argues how the uniform density assumption is useful after all. Texture density is also used and the approach is to partially separate the foreshortening and distance effects. The uniform density assumption is generalized to another form which is said to capture a very large subset of natural and man-made textures. Instead of finding texels, referred to as "strong segmentation", he is able to use the edges of the texels ("weak segmentation"). The work is primarily based on an approximation of perspective projection, known as *paraspective* projection.

This work by Aloimonos is interesting because he is able to estimate surface orientation by estimating texture density and minimizing for P and Q using a constraint for uniform density assumption. It is important to note that the method does not exactly find and count texels. Instead it assumes that texels edges can be detected and that the sum of the lengths of these edges is uniformly distributed. The results using paraspective projection are slightly more accurate than those using perspective projection.

Fast forwarding to our time we have Loh's work [12]. The work presents algorithms for different scenarios. One for homogeneous and stationary texture using orthographic projection, and another algorithm which is more general and does not make any assumptions about the texture (also published in [13]). Part of the work is *spectral* based, i.e. is makes assumptions and extract image information by using features from the fourier amplitude spectrum. The more general algorithm focuses on independent texels and searches for the affine transformation between a "frontal" texel and the rest. The results seems promising.

This section took a look at some important fundamentals and previous work. The next section will shortly explain how and why the experiments were carried out as they were. The goals of the project will also be presented.

1.3. Project Methodology

It seem like a good idea to give a brief explanation on how the work was carried out and what tools were used, though much of this will also be evident from Chapters 3 and 4.

The main goal of this project is to investigate whether or not the formalisms presented by Arnspang [2] can be used for the purpose for which they were published in 1991. More explicit goals are specified within the problem domains of the two kinds of experiments.

This project applies two methods of experimentation: 1) Computational experiments and 2) camera experiments. It turned out, as the project progressed, that the majority of the time (close to 80%) used, had been spent on designing, testing, and debugging the computational experiments. These experiments were carried out using Wolfram's Mathematica 6 – an advanced symbolic computer algebra system (CAS). It allows more expressive freedom in the computations than, for instance, MATLAB does. It should be noted that MATLAB does also have symbolic manipulation capabilities borrowed from Maple. But this functionality was not discovered until work in Mathematica had begun.

The camera experiments had a much broader application focus than the computational experiments. Instead of "only" proving the formalisms behind the primary method computationally, the camera experiments sought to extract 3D surface structure from *real* image sequences. The camera experiments involved much different aspects of computer vision engineering than the computational experiments did. It required the design and manufacturing of a physical installation (the "experimental setup") which makes it possible to create actual image sequences of a moving plane textured surface using a *real* digital camera.

1.3.1. Computational Experimentation

The main idea was to create a computational "framework", based on the mathematical assumptions in the primary method. Using the built-in numerical optimization features of Mathematica, as it is shown in Chapter 3, it is indeed possible to solve for the six variables P, Q, Z, U, V, W (though it seems that certain numerical singularities do occur). All experiments were carried out using simulated *plane* surfaces.

The goals of the computational experiments:

- Build a mathematical framework in Mathematica which computationally proves that the formalisms behind the primary method are correct.
- Simulate perspective projection of surface points on a plane surface, for a subset of plane orientations, depths, and velocities.
- Simulate synthetic *surface* and *image* texture density using the formalisms in the primary method.
- Estimate first partial derivatives of the image texture density.
- Build and solve constraint equations, for single and multiple image points, for the variables U, V, W, P, Q, Z.

With these goals successfully achieved, it will have been computationally proven that the formalism behind the primary method indeed are correct. This could be seen as important groundwork for further research into the mathematical formalism behind the primary method.

1.3.2. Camera Experimentations

The camera experiments, with their much broader focus, introduced certain difficulties from day 1. As Chapter 4 will describe in detail, mistakes were made when designing and building the experimental setup. Due to such setbacks the camera experiments where simplified considerably. For instance, the textured surface was frontal only ($P = 0$ and $Q = 0$). The line-of-travel of the surface was also toward the camera only, and set to be the actual Z-axis ($U = 0$, $V = 0$, and $W \neq 0$). This means that the camera experiments can only *model* situations which involves *frontal* planes traveling in a direct line toward the camera, parallel to the Z-axis.

The implementation of digital image processing algorithms (for instance, connected component analysis) was carried out in MATLAB, due to its strengths in digital image processing.

The goals of the camera experiments:

- Design and build an *experimental setup* that makes it possible, by using a digital camera, to create image sequences of a movable plane textured surface. It should be possible to change the texture, orientation, and *line-of-travel* of the plane surface.
- Create one or more image sequences using the experimental setup and a digital camera. Do this for a frontal plane surface with a line-of-travel toward the camera, parallel to *or* on the Z-axis.
- Implement functionality for estimating texture density in simple *known* textures with separable texels (connected component search techniques).

- Build and solve constraint equations, for single and multiple image points, for the variables U, V, W, P, Q, Z.

Should the goals be met, it would imply that the formalisms of *texture flow* as a cue for direct extraction of 3D structure, can indeed be applied to a real planar surface, traveling in a direct line toward the camera (parallel to the Z-axis), with *known* texture.

Later steps for estimating image texture density and eventually the partial derivatives (the texture flow), turned out to be unsuccessful. See Chapter 4 for further conclusions.

2. The Primary Method

This chapter will explain in details the formalisms behind the primary method – a new method for automatic estimation of surface structure and velocity. Subsection 1.1.2 in Chapter 1 explained the basic ideas. This chapter goes into further detail about some of the underlying assumptions and theory. Section 2.1 derives the Texture Density Equation and Section 2.2 does the same for a set of Texture Flow Equations.

2.1. The Texture Density Equation

Chapter 1 introduced the primary method, but only in overall detail. This Section will derive the ratio between the two "*image patches*" dS and dI using the pinhole model and *solid angles*. Finally the Texture Density Equation will be derived.

2.1.1. Solid Angles

This subsection will take a look into the basics of *solids angles*. Solid angles are a concept of angles in three dimensions and are measured in *steradians*.

In two dimensions, the *central* angle θ in radians, *subtended* at the center of a circle with radius r, is related to the *arc length* s it cuts out (*subtends*) on the circle's circumference: $\theta = s/r$. In three dimensions, the solid angle Ω in steradians, subtended at the center of a *sphere* with radius r, is related to the *spherical cap* area A it cuts out (subtends) on the sphere's surface: $\Omega = A/r^2$. Figure 2.1 shows the relation between angles in 2d and solid angles in 3d.

Confusion can arise when using the words *subtended* and *subtends*. A central (or solid) angle *subtends* a spherical arc (or cap), and thus a spherical arc (or cap) is *subtended* on a circumference (or surface) by a central (or solid) angle. On the other hand, one can also say that a spherical arc (or cap) *subtends* a central (or solid) angle. Thus a central (or solid) angle is *subtended*, by a spherical arc (or cap), at a single point known as the angle's *vertex*; usually defined as the center of a circle (or sphere) with radius r. Distinguishing between the two methods of description is usually not necessary and both are used interchangeably in this report.

The measure of steradian in a solid angle at a sphere's center, corresponds exactly to the measure of the subtended spherical cap area at unit distance from the sphere's

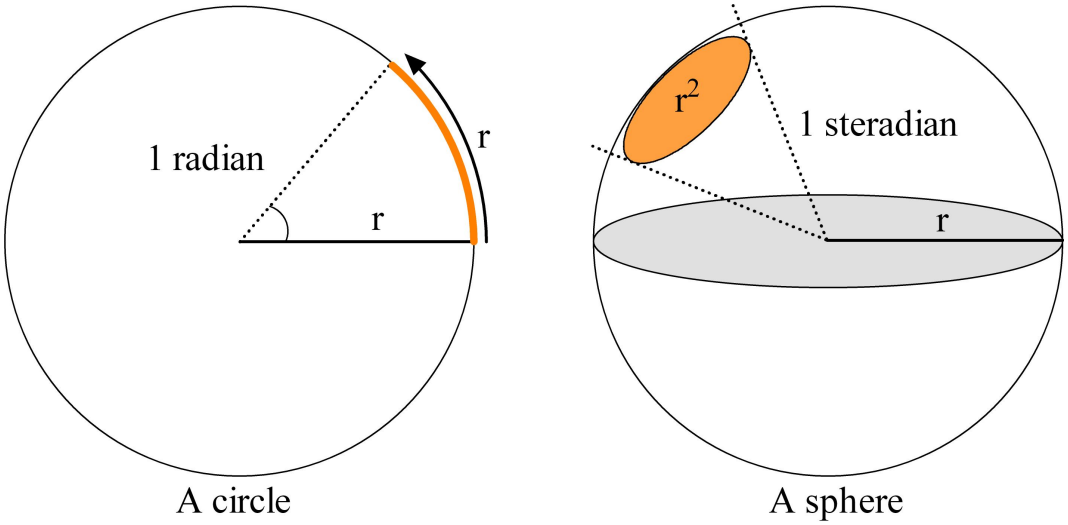


Figure 2.1.: Figure shows the relation between 2d angles measured in *radians*, and 3d *solid angles* measured in *steradians*.

center. For instance, consider a sphere with a radius of 1 meter. A solid angle of 1 *steradian*, at the center of the sphere, subtends exactly 1 *square meter* of the sphere's surface. A solid angle of 3.7 steradian would subtend exactly 3.7 square meter.

Back in two dimension, we also know that an angle of 2π *radians* corresponds to the entire circle (since the full circumference has an arch length of $2\pi r$). Since the surface of a sphere is equal to $4\pi r^2$, a solid angle of 4π *steradians* subtends the entire surface of a sphere.

A central angle subtends a spherical arc on the circle's circumference. The part of the circle "covered" by the angle is known as a *circular sector*. The corresponding geometric shape in solid angles is a *right circular cone* with the *apex* at the center of the sphere. The perimeter of the *base* of the cone intersects with the sphere's surface. The geometry involved is seen depicted with considerable detail in Figure 2.2.

Solid angles are useful in the context of the simple imaging geometry used in the pinhole model. A measure of solid angle is a measure of how large and object appears to be at a certain distance. It is possible, for instance, that a nearby small object is subtended by the same solid angle as a larger object far away. Solid angles are used here to model the size of *perceived area* of an objects surface, which might be at at an angle to the direction of line–of–sight.

2.1.2. Derivation

This subsection will derive the ratio dS/dI using solid angles. This relation between dI and dS is rather trivial and can be derived using both the thin lens model and the pinhole model. The pinhole model is used here.

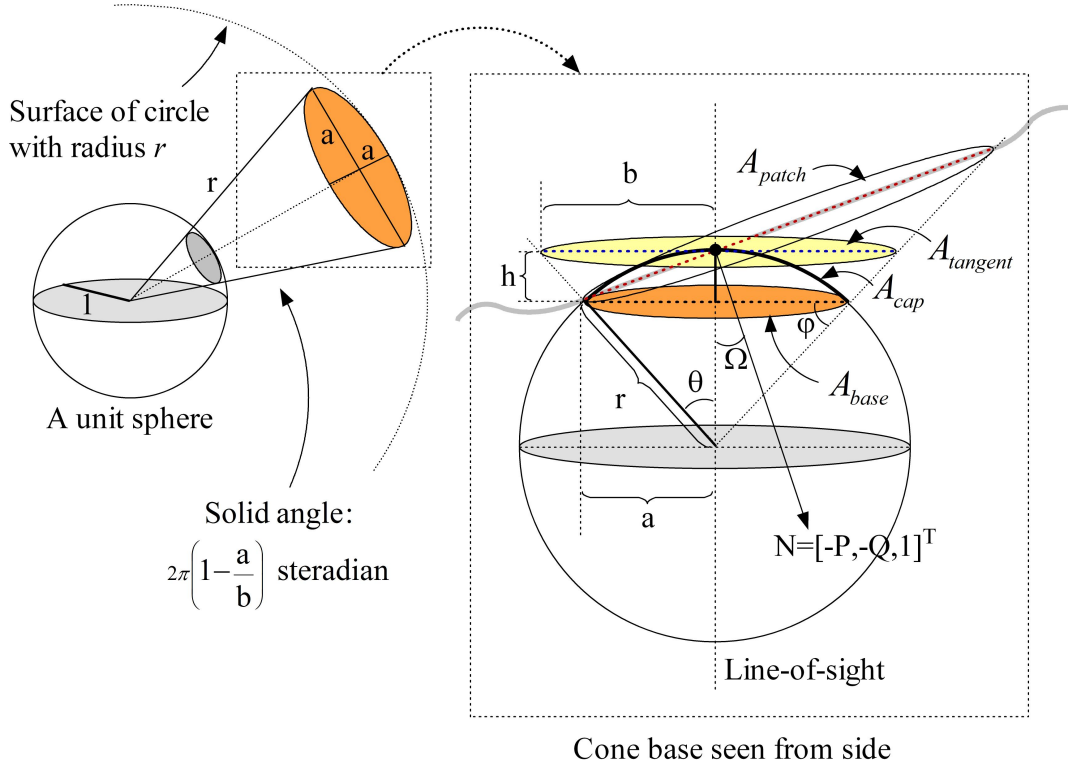


Figure 2.2.: Figure shows most of the geometry related to the use of solid angles in this report.

Figure 2.2 shows the geometry in the solid angle cone. The graphic to the left shows the general situation involving a sphere of radius r. It is important to understand that the area A_{base} of the base of the cone, which in Figure 2.2 equates to $A_{base} = a^2\pi$, is *not* identical to the area A_{cap} of the spherical cap (which equates to $A_{cap} = 2\pi rh$, where $h = r \cdot \frac{b-a}{b} = r \cdot [1 - \frac{a}{b}]$). The graphic to the right in Figure 2.2 views the cone orthogonal to the line-of-sight. The graphic also shows a circle *tangent* to the sphere surface and is thus at distance r to the center (origin). This circle has area $A_{tangent} = b^2\pi$.

The circle with area A_{patch} (which is also tangent to the sphere) corresponds to an *image patch*, or what in Chapter 1 was referred to as a "unit area". It is rotated at an angle Ω between its surface normal N and the line-of-sight. The area $A_{tangent}$ is a "foreshortened" version of A_{patch}, and can thus be expressed as $A_{tangent} = A_{patch} \cdot \cos\Omega = b^2\pi$ (the cosine can here be considered a *scaling* factor).

Importantly, we now know that

$$b = \frac{1}{\sqrt{\pi}} \cdot \sqrt{A_{patch} \cdot \cos\Omega}$$

and we also know that $a = r \sin\theta$ and $\tan\theta = \frac{b}{r}$. One can now derive A_{cap} expressed

using A_{patch} and $\cos\Omega$ by further use of simple geometry:

$$\frac{a}{b} = \frac{r}{b}\sin\theta = \frac{r}{b}\sin\left[\arctan\left(\frac{b}{r}\right)\right] = \sqrt{\frac{r^2}{r^2+b^2}} = \sqrt{\frac{r^2\pi}{r^2\pi + A_{patch}\cdot\cos\Omega}}$$

And, since we know that $A_{cap} = 2\pi rh = 2\pi r^2(1-\frac{a}{b})$, we can now express the spherical cap area A_{cap} as

$$A_{cap} = 2\pi r^2\left(1 - \sqrt{\frac{r^2\pi}{r^2\pi + A_{patch}\cdot\cos\Omega}}\right) \tag{2.1}$$

The general expression of A_{cap} has now been derived and there would be no problem in continuing with it. In the computer vision literature though, a solid angle S; as a ratio between surface area and squared distance; is often defined as $S = A_{tangent}/r^2$, and not as $S = A_{cap}/r^2$.

This is not quite as forbidden as it might look at first. All the involved quantities are consequences of the physical image acquisition process. When surface area is considered as a *radiometric* quantity, it is an *infinitesimally* small area – it is almost zero. Any actual difference between the areas A_{cap} and $A_{tangent}$ can thus be considered negligible, and stating $A_{tangent} \approxeq A_{cap}$ becomes allowed. From this point on, this text will use $A_{tangent} = A_{patch}\cos\Omega$ as the model of "perceived area".

Figure 2.3 shows the pinhole model including two solid angles. A fundamental assumption, which is valid in perspective projection, is that these two solid angles are equal, which would mean that the two cone bases at unit distance from **O** are equal in size.

This equality can be written as:

$$\frac{dI\cos\beta}{(r_{image})^2} = \frac{dS\cos\Omega}{(r_{surface})^2} \Leftrightarrow \frac{dI\cos\beta}{\left(\frac{f}{\cos\beta}\right)^2} = \frac{dS\cos\Omega}{\left(\frac{Z}{\cos\beta}\right)^2}$$

where r_{image} and $r_{surface}$ represents the distance from **O** to point **p** and **P**, respectively.

And thus we have the ratio[1] dS/dI as also shown earlier en Chapter 1:

$$\frac{dS}{dI} = \frac{\cos\beta}{\cos\Omega}\left(\frac{Z}{f}\right)^2 \tag{2.2}$$

The point **p** is also a vector: $\mathbf{p} = [-x, -y, f]^T$, expressed in the camera reference system ($\mathbf{O} = [0,0,0]^T$). Using **p** and another vector, $\mathbf{e} = [0,0,1]^T$, as shown in

[1] The same ratio would naturally have resulted if the "perceived area" had been modeled using the area A_{cap} as defined in Equation2.1.

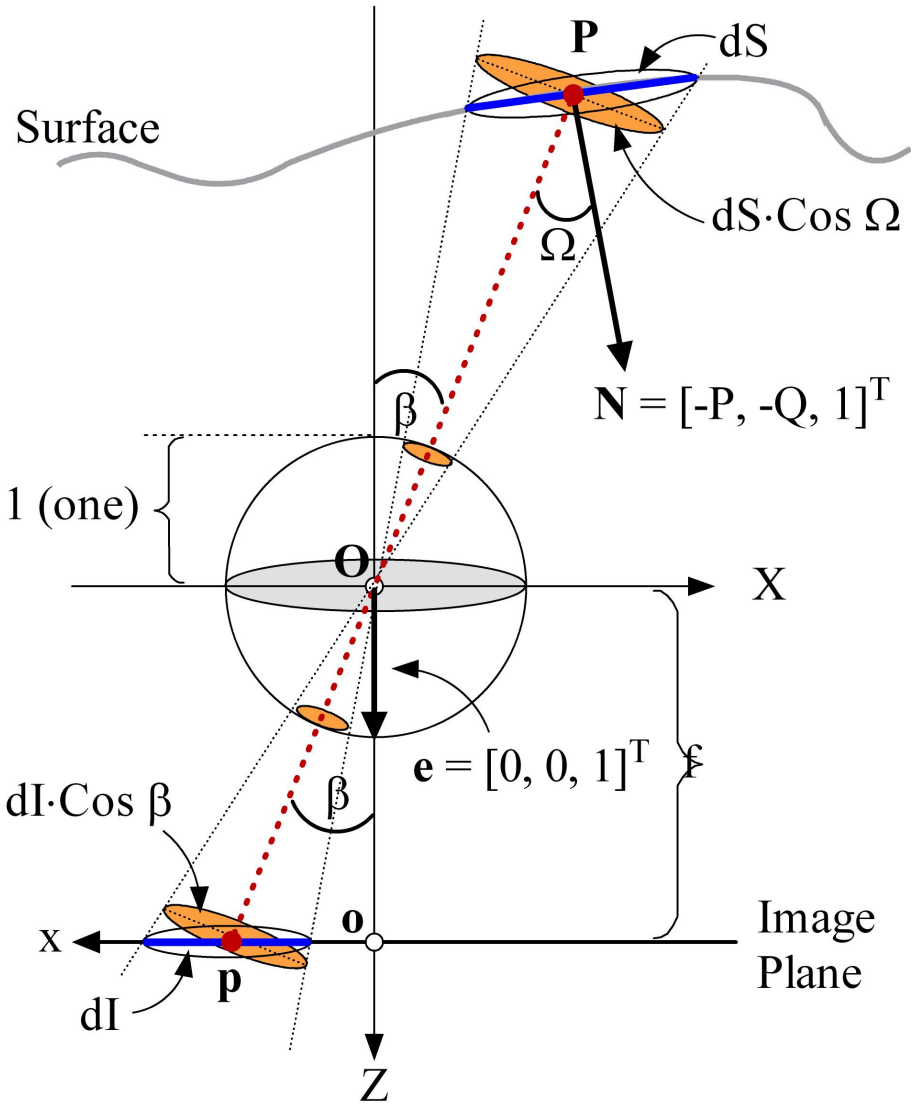


Figure 2.3.: The pinhole model with the solid angles created by the perspective projection geometry.

Figure 2.3, one can rewrite the cosines of the two angles Ω and β in Equation 2.2 using dot products:

$$\cos\Omega = \frac{\mathbf{p}^T\mathbf{N}}{\|\mathbf{p}\|\,\|\mathbf{N}\|}, \quad \cos\beta = \frac{\mathbf{p}^T\mathbf{e}}{\|\mathbf{p}\|\,\|\mathbf{e}\|}$$

and the ratio dS/dI thus equates to:

$$\frac{dS}{dI} = \frac{Z^2}{f}\left[\frac{\|\mathbf{N}\|}{f}\frac{\mathbf{p}^T\mathbf{e}}{\mathbf{p}^T\mathbf{N}}\right] = \frac{Z^2}{f}\frac{\sqrt{P^2+Q^2+1}}{Px+Qy+f}$$

which leads us to the *Texture Density Equation* in its most basic form:

$$\frac{\sigma}{dI} = \frac{\sigma}{dS}\frac{dS}{dI} \Leftrightarrow \mu_I = \mu_S \frac{dS}{dI} \Leftrightarrow \mu_I = \mu_S \frac{Z^2}{f}\frac{\sqrt{P^2+Q^2+1}}{Px+Qy+f} \tag{2.3}$$

Equation 1.4 in Chapter 1 introduced the involved quantities as functions. One of these, the *image texture density* function μ_I, is used in Section 2.2 to derive two *Texture Flow Equations*:

$$\mu_I(x(t), y(t), t) = \mu_S(m,n) \cdot \tau(x, y, P, Q, Z)$$

where

$$\tau(x, y, P, Q, Z) = \frac{Z^2}{f}\frac{\sqrt{P^2+Q^2+1}}{Px+Qy+f}$$

The function τ is interesting and it seems highly relevant for the purpose at hand. It involves both depth (Z) as well as orientation (P and Q) and image position (x and y). It seems only natural to exploit this expression even further. This will happen in the next section which will derive a set of texture flow equations.

2.2. The Texture Flow Equations

As this section will show, the problem with surface shape and velocity estimation can be solved by solving for the appropriate variables in a set of *Texture Flow Equations*. This section will derive such equations by taking the total time derivative, of $\mu_I(x(t), y(t), t)$ over an *optic* curve C in (x, y, t)-space.

2.2.1. Optic and Density Curves

Consider a sequence of images. Perhaps it is due to a moving surface. It might be a translating surface, a rotating surface, or a deforming surface. Or perhaps a static surface with a time varying surface texture density (for example, cast shadows of waving leaves).

Consider a point $\mathbf{P}$ on this surface, with *spatial* velocity $\mathbf{V_P} = [U, V, W]^T$. Point $\mathbf{P}$ will be at distance Z, and the image patch at $\mathbf{P}$ has the normal vector $\mathbf{N} = [-P, -Q, 1]^T$ (see Figure 1.3).

Consider also that point $\mathbf{P}$ projects to image point $\mathbf{p}$ and the light intensity (irradiance) in the images $E(x(t), y(t), t)$ can be measured in the image sequence. In the problem of *optical flow*, one seeks to derive the *image motion* of all points $\mathbf{p}$: $\mathbf{v_p} = [u, v]^T = [dx/dt, dy/dt]^T$.

The assumption is that the image intensity at a certain image patch remains constant at "small" variations in x, y, and t ([9, 10]):

$$E(x, y, t) = E(x + \delta x, y + \delta y, t + \delta t)$$

One can now derive a *Motion Constraint Equation* using a first order Taylor expansion or by assuming that the image intensity is conserved and using the chain rule for differention:

$$\frac{dE(x(t), y(t), t)}{dt} = 0 \Leftrightarrow \frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = E_x u + E_y v + E_t = 0$$

The motion constraint equation describes a line in (u, v)-space and further constraints are needed to to solve for both u and v, uniquely specifying a point one that line. One classic example is Horn and Schunck's [9] *smoothing constraint*, which assumes that the image motion field varies smoothly almost everywhere in the image.

It can be difficult to imagine a function in three variables. Consider, for instance, the density in a cloud, where the function $d(x, y, z)$ defines the cloud density at a certain place in the cloud. The same analogy can be used for image intensity $E(x, y, t)$ and image texture density $\mu_I(x, y, t)$.

In Chapter 1 the notion of an *optic curve* was introduced. In optical flow estimation the assumption was that the *local* image intensity $E(x, y, t)$ remains close to constant, even if there is motion in the overall image. It is not hard to imagine that a moving neighborhood of image points (or perhaps just a single point in the limit), with the same image intensity, "carves out" a three-dimensional curve $C = \{x(t), y(t), t\}$ in (x, y, t)-space. It is this curve which is known here as an *optic curve*. The image intensity $E(x, y, t)$ can be considered as a *bundle* of such curves in (x, y, t)-space.

Returning to the image sequence and the texture densities, the image texture density $\mu_I(x, y, t)$ can also be considered as such a curve. Instead of calling it an *optic* curve though, a more accurate description would perhaps be a "density" curve. Such a density curve models the *response* of a high-level spatiotemporal image texel counting *operator* or *filter*, in (x, y, t)-space. Also, contrary to image intensity, one *cannot* in general expect the local measure of texture density in an image neighborhood to be constant.

One can thus state the following general truth about image texture density:

$$\mu_I(x, y, t) \neq \mu_I(x + \delta x, y + \delta y, t + \delta t).$$

which means that

$$\frac{d}{dt}\mu_I(x(t), y(t), t) \neq 0$$

Degenerate cases, contrary to this general rule, can of course exist. For instance, consider a *frontal plane* surface ($P = 0$, $Q = 0$) moving parallel to the camera ($W = 0$). No movement occurs on the Z-axis, hence no depth variations occur. The frontal plane surface has a homogeneous *surface texture* which means that it has the same *surface* texture density everywhere (uniform texture density). Since the plane is exactly *frontal*, the *image* texture density is also uniform over space and time.

Image texture density is naturally dependent on scale, and since scale itself is dependent on variations in depth – of which there is none at the moment – it would be possible for such density curves in (x, y, t)-space to, have the same image texture density value from end to end. But in general we cannot expect scale to be constant, as well as we cannot expect all surface textures to be homogeneous.

Now that the concept of optic and density curves has been defined, the next subsection will move on to deriving a set of texture flow equations.

2.2.2. Derivation

The texture flow equation seen in Equation 2.3 can be rewritten as the following:

$$\mu_I f\,(Px + Qy + f) - \mu_S Z^2 \sqrt{P^2 + Q^2 + 1} = 0 \tag{2.4}$$

This expression states that a certain combination of surface and image texture density, orientation, and depth, at on specific image point, equates to 0. In regards to this and later expressions, it should be noted that all quantities, except f, naturally are directly (or indirectly) functions of time t: $x \equiv x(t)$, $y \equiv y(t)$, $P \equiv P(t)$, $P' \equiv P'(t)$, $Q \equiv Q(t)$, $Q' \equiv Q'(t)$, $Z \equiv Z(x(t), y(t))$, $U \equiv U(t)$, $V \equiv V(t)$, and $W \equiv W(t)$.

It is important to note that Equation 2.4 defines a local relation in the image (at point (x, y)). To have something to say about other important time-dependent quantities (as, for instance, the translational velocity $[U, V, W]^T$ and the rotational velocity P' and Q'), it would be natural to take the total time derivative of function $\mu_I(x(t), y(t), t)$. This is said to be done over an optic curve $C = \{x(t), y(t), t\}$ which exists in (x, y, t)-space.

This is a natural consequence of the nature of function μ_I. As mentioned earlier, the image texture density, measured locally in each image in a sequence of images, is modeled as a density curve C in (x, y, t)-space. Taking the derivative *along* this density curve, we arrive at useful expressions which involves the needed partial derivatives to model the spatiotemporal 3D gradient, understood here as as *texture flow*.

The *total* time derivative of $\mu_I(x(t), y(t), t)$ is derived using the chain rule:

$$\mu'_I(x(t), y(t), t) = \mu_{Ix} u + \mu_{Iy} v + \mu_{It}$$

where μ_{Ix}, μ_{Iy}, and μ_{It} are the partial derivatives of μ_I in x, y, and t, respectively.

The *surface* texture density μ_S is always considered a constant. Thus $\mu'_S = 0$. Complex surfaces on non-rigid surfaces, e.g. the surface of water, with time dependent *surface* texture density, can thus not be modeled. Also, for reasons of simplification, the two rotational velocities are, at the moment, both considered to be *zero*: $P' = 0$ and $Q' = 0$.

When knowing this, taking the time derivative of the left-hand expression in Equation 2.4 results in the following:

$$[\mu_{Ix}u + \mu_{Iy}v + \mu_{It}]f(Px + Qy + f) + \mu_I f(Pu + Qv) - 2\mu_S ZW\sqrt{P^2 + Q^2 + 1} = 0$$

where $u = dx/dt$, $v = dy/dt$, and $W = dZ/dt$. Later on, two other velocities are also used: $U = dX/dt$, and $V = dY/dt$.

The surface texture density μ_S is now substituted with the expression in 2.3 and the first *Texture Flow Equation* for a *translating* texture with constant *surface* texture density (fixed texture) appears:

$$\left[\mu_{Ix}u + \mu_{Iy}v + \mu_{It} - 2\mu_I\frac{1}{Z}W)\right] \times (Px + Qy + f) + (Pu + Qv)\mu_I = 0 \quad (2.5a)$$

When $P' \neq 0$ and $Q' \neq 0$, the differentiation and substitution results in a second and sligtly different *Texture Flow Equation* for a surface with constant *surface* texture density (fixed texture):

$$\left[\mu_{Ix}u + \mu_{Iy}v + \mu_{It} - 2\mu_I\frac{1}{Z}W)\right] \times (Px + Qy + f) + \mu_I\left[Pu + Qv + P'x + Q'y - \frac{(PP'QQ')(Px + Qy + f)}{(P^2 + Q^2 + 1)}\right] = 0 \quad (2.5b)$$

Equation 2.5a considers P', and Q' to be *zero*. This is not the case with Equation 2.5b which includes those quantities. Of course when setting P', and Q' to *zero* in Equation 2.5b, you end up with Equation 2.5a.

In Chapter 1 the surface texture density was defined as a function $\mu_S \equiv \mu_S(m, n)$, where m and n are *surface* parameters[2]. This corresponds to fixed or constant surface texture density, because the function is defined as being independent of time and thus $\mu'_S = 0$.

The two texture flow equations in Equation 2.5a and 2.5b involves u and v. It is possible to find other more useful expressions of these two *optic flow* quantities. Consider

[2]We will meet these two parameters again in Chapter 3, where they are used explicitly in the computational experiments.

the relations in the pinhole model, between space coordinates ($[X,Y,Z]^T$) and image coordinates ($[x,y]^T$):

$$fX + xZ = 0, \qquad fY + yZ = 0$$

By taking the time derivative of both left-hand expressions we get the following:

$$fU + uZ + xW = 0, \qquad fV + vZ + yW = 0$$

We now have a useful expression of the two optic flow velocities u and v:

$$[u,v]^T = \left[-\frac{fU + xW}{Z}, -\frac{fV + yW}{Z} \right]^T$$

By substituting these expressions for u and v in Equation 2.5a and 2.5b, we obtain the "direct" versions of the two texture flow equations:

$$\begin{aligned} &\Big[\mu_{Ix}(fU + xW) + \mu_{Iy}(fV + yW) - \mu_{It}Z + 2\mu_I W\Big] \\ &\qquad \times (Px + Qy + f) + \mu_I \left[P(fU + xW) + Q(fV + yW)\right] = 0 \end{aligned} \quad (2.6a)$$

$$\begin{aligned} &\Big[\mu_{Ix}(fU + xW) + \mu_{Iy}(fV + yW) - \mu_{It}Z + 2\mu_I W\Big] \times (Px + Qy + f) \\ &\qquad + \mu_I \Bigg[P(fU + xW) + Q(fV + yW) - ZP'x - ZQ'y \\ &\qquad\qquad + Z\frac{(PP'QQ')(Px + Qy + f)}{(P^2 + Q^2 + 1)} \Bigg] = 0 \end{aligned} \quad (2.6b)$$

Equation 2.6a is a nonlinear equation in U, V, W, P, Q, and Z. Equation 2.6b also involves P' and Q' and is thus a more general equation. Of course when setting P', and Q' to *zero* in Equation 2.6b, you end up with Equation 2.6a. The reason for having two different forms of the *same* equation, is due to the specific interest in experimenting with the more simplified situation – where no rotational velocities are involved.

The two equations could turn out to be useful for estimating the following quantities at image point (x,y):

- Surface *orientation* (P and Q)
- Surface *depth* (Z)
- Surface *translational* velocity (U, V, W)
- Surface *rotational* velocity (P' and Q')

The idea is that, at image point (x, y); for the correct values of surface orientation, translational and rotational velocities, the first derivatives in x, y, and t of the image texture density; these equations should equal *zero*. Some numerical *test trials* were used to computationally prove that these equations do indeed equal zero (at order of magnitude from -12 to -6) when given the correct values.

Much of the numerical stability though, rests on the exactness of the derivatives estimated. Derivative estimation was done by minimizing a linear system of taylor polynomials in three variables of order $n \geq 9$. Appendix A shows several results from these trials.

To solve such equations, when the exactness of the derivatives is good enough, one turns to methods of numerical minimization. This report will not deal with the specific methods involved in such algorithms, and all the experiments utilize the existing built-in minimization functions of the Mathematica computer algebra system.

This section took a look at the concept of optic and density curves and derived a number of texture flow equations. Equations 2.5a and 2.5b involve the quantities of optic flow, u and v. It has not been determined whether or not one might be able to use these equations for optical flow estimation. Perhaps such a question can be answered with further work. No experiments have been carried out with these equations.

The other set of texture flow equations, the more "direct" versions seen in Equations 2.6a and 2.6b, does not involve optic flow quantities, but instead involves the image point coordinates x and y, translational velocities $[U, V, W]^T$, and also, for Equation 2.6b, rotational velocities P' and Q'.

To ease the notation later on, the left hand expressions of Equation 2.6a and 2.6b will be known as *constraint equations* Φ and Φ_R, respectively.

Thus we have that:

$$\Phi(U, V, W, P, Q, Z) = 0 \qquad \text{(see Equation 2.6a)}$$

$$\Phi_R(U, V, W, P, Q, Z, P', Q') = 0 \qquad \text{(see Equation 2.6b)}$$

The experimentations were done using only constraint equation Φ. No incorporation of rotation using constraint equation Φ_R has been attempted.

The next chapter, Chapter 3, will present the methods and results from the computational experiments. It introduces a simple but fundamental mathematical structure in the form of the some formal definitions. It elaborates further on *how* exactly the computational framework is able to use the formalisms from this chapter for minimizing the constraint equation Φ.

3. Computational Experiments

This chapter is concerned with the methods used and the results achieved in the *computational* experiments. These are experiments in a purely *synthetic* form – computer simulations. They involve only the abstract mathematical concepts; for instance, the pinhole model geometry, *point sampling*, data flow, and *numerical* texture densities. Section 3.1 introduces a necessary mathematical structure using set theoretic definitions. Section 3.2 gives an overview of the data flow the numerical simulations. Section 3.3 presents the method used to estimate the partial derivatives of the image texture density. Section 3.4 presents the experimental results in the form of a numerical example.

3.1. Formal Structure

The computational experiments are *computer simulations* made using Wolfram's Mathematica[1] – a computer algebra system. Mathematica has strengths in *symbolic manipulation* of mathematical expressions, support of anonymous functions applied to large lists, numerical minimization, regression, and *data visualization* etc. Another particularly strong application, MATLAB[2] by Mathworks, is later used in the camera experiments in Chapter 4 for digital image processing tasks covering, among others, *color segmentation* and *connected-component analysis* (for *real* texture density estimation).

A set of Mathematica scripts constitutes the implementation of the computer simulations. These scripts are part of a larger simulation "framework". This section will explain the general functionality of this framework: *What* is simulated, and *how* it is simulated. No examples of code will be given. An underlying mathematical structure is later defined, using set theoretic notation. Such details are crucial for understanding and simulating the formalisms of Chapter 2. When the mathematical foundations has been explained, section 3.2 is ready to take a look at the actual data flow of the simulations.

The goal from the beginning of the design and implementation were to simulate the process of taking images of a moving rigid surface in space using a fixed perspective

[1] Version 6 was used. Mathematica's website `http://www.wolfram.com`.
[2] Version 7.x was used. `http://www.mathworks.com`.

camera. The camera is modeled using the pinhole model; which means that there is no simulation of focus or other *physical* details – the camera has infinite focus. Points on the surface in space are simply *projected* onto an image plane, at specific time intervals, creating a series of images. The projection is done by applying the camera/image coordinate conversion equations (the relations seen in Equation 1.1 in Chapter 1).

This simple form of simulation is known (in this report) as *point sampling*. The process of point sampling consists of three subtasks: (1) Sampling of the necessary discrete *time steps*, (2) sampling of *surface points* at a subset of the estimated time steps, and (3) sampling of *image points* at the same subset of time steps. To sample (create) 1 *surface* point, it requires the calculation of a set of abstract numerical *surface* texture densities, the same for each *image* point. This point sampling process is the foundation of all the results achieved computationally and Section 3.2 will have more say about the related data flow details.

When the simulation is done, i.e. when the *point sampling* process is complete, another process begins. This is the *minimization* process. It sets up the final *constraint* system and, by user intervention, solves this system, using either *constraint* equation Φ or Φ_R. More details about the important minimization process is postponed until Section 3.2.

The use of the word "*user*" simply refers to the fact that the simulation has reached a stage where some form of interaction with the simulation is needed – a user needs to choose certain parameters for the simulation to continue. Certainly this interaction could also have been scripted.

3.1.1. Notation

The subsections ahead make formal definitions, using *set theoretic* notation. The set of *natural numbers* $\mathbb{N}$ contains the *zero* element. So does $\mathbb{N}_0$. But $\mathbb{N}^+$ does not. The symbol $\mathbb{N}$ is only used when the ambiguity is not important. With other sets, where *non negativity* has meaning, a superscript "+" is used for *positive* numbers, a superscript "$*$" for *nonnegative* numbers, and a superscript "$-$" for *negative* numbers. A subscript "0" is added for *nonpositive* numbers.

For example, for the set of of *integers* $\mathbb{Z}$:

$$\begin{aligned}
\mathbb{Z} &= \{\dots, -2, -1, 0, 1, 2, \dots\} \\
\mathbb{Z}^+ &= \mathbb{N}^+ = \{1, 2, \dots\} \\
\mathbb{Z}^* &= \mathbb{N}_0 = \{0, 1, 2, \dots\} \\
\mathbb{Z}^- &= \{\dots, -2, -1\} \\
\mathbb{Z}_0^- &= \{\dots, -2, -1, 0\}
\end{aligned}$$

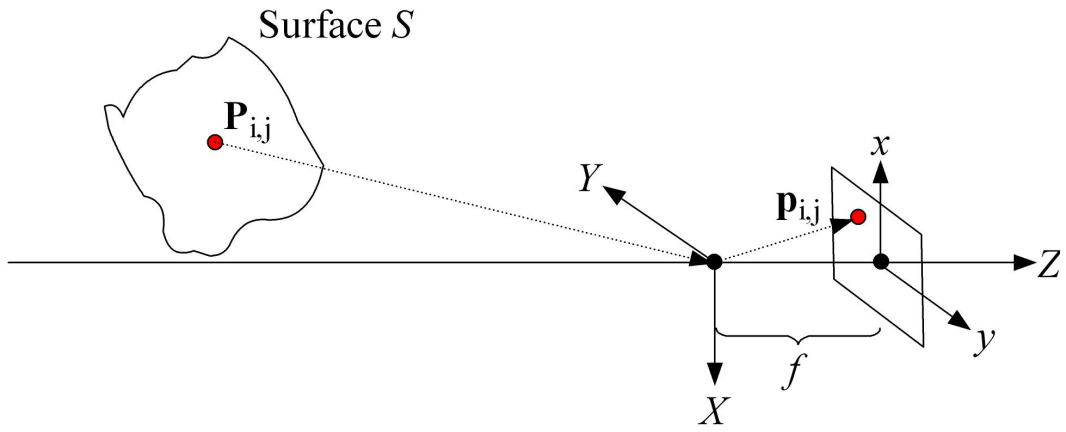


Figure 3.1.: The general situation. A rigid surface S moves in space and the camera with focal length f is fixed. A *surface* point $\mathbf{P}_{i,j}$ projects to *image* point $\mathbf{p}_{i,j}$ (see text).

The next couple of subsections will define a fundamental mathematical structure. The text uses *non-bold non-capital* letters for *scalars* and *functions* (i, j, μ); *non-bold capital* letters for *surfaces*, *sets*, *families of sets*, and also, when no ambiguity can arise, scalars (F, G, H); *bold non-capital* letters for *vectors* ($\mathbf{a}$, $\mathbf{b}$); and *bold capital* letters for *matrices* ($\mathbf{M}$, $\mathbf{U}$). *Calligraphic* letters ($\mathscr{A}$, $\mathscr{S}$) are not used.

3.1.2. Surface, Frames, and Surface Points

A *rigid* surface (two-dimensional manifold) S moves in space, and the camera is fixed in space at origin $\mathbf{O}$ with *focus length* $f \in \mathbb{R}^+$. At time $t_i = (i-1)\Delta t$, where $t_i \in \mathbb{R}^*$, $\Delta t \in \mathbb{R}^+$, and $i, d \in \mathbb{N}^+$ with $1 \leq i \leq d$, k *surface points* $\mathbf{P}_{i,j} = [X_{i,j}, Y_{i,j}, Z_{i,j}]^T$ projects to k *image points* $\mathbf{p}_{i,j} = [x_{i,j}, y_{i,j}]^T$, where $j, k \in \mathbb{N}^+$ and $1 \leq j \leq k$ (see Figure 3.1). The symbol j will be known later on as a *point index*, used for iteration and unique indexing of all k surface and image points.

The symbol i, is the integer *time index*. At the *beginning of time*, at $t_1 = 0$, the time index is $i = 1$ and the clock is exactly *zero* (see Figure 3.2). At time t_1, nothing which is time dependent have yet had any effect on the simulation. The first time that happens, is at time $t_2 = \Delta t$.

The k surface points $\mathbf{P}_{i,j}$, which exist *on* surface S, are used to represent a *subset* of the entire surface S (which by formality can be considered to have *infinite* extent) at time t_i. This subset of surface S is represented as a *cloud* of k surface points $\mathbf{P}_{i,j}$. Methods for defining surface S, and the *extent*, and *resolution* of the surface point cloud, will be explained in a later subsection.

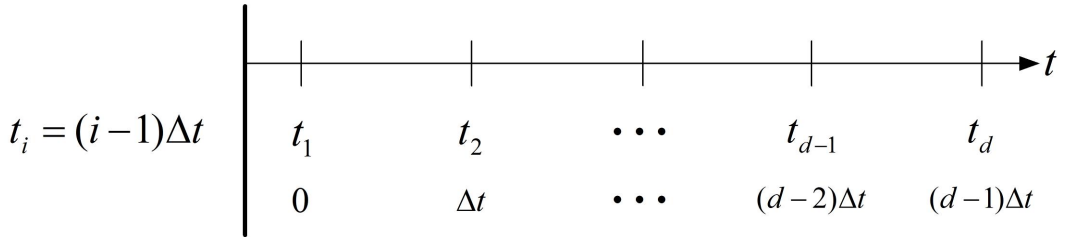


Figure 3.2.: Figure shows how the concept of time is defined and used. Note especially that time begins at $t_1 = 0$ and ends at $t_d = (d-1)\Delta t$. The reason for beginning at $i = 1$ is a detail inherited from *MATLAB* and *Mathematica*, where all lists are 1-indexed and not 0-indexed as is the case with other programming languages such as *C*, *C++*, and *Java*.

A *frame* is a set F_i which contains the k surface points (a 3D point cloud) at time t: $F_i = \{\mathbf{P}_{i,1}, \mathbf{P}_{i,2}, \ldots, \mathbf{P}_{i,k}\}$. A family of sets G, containing the d frames, is also defined: $G = \{F_1, F_2, \ldots, F_d\}$.

3.1.3. Surface Velocity, Rotation, and Simplifications

For each surface point $\mathbf{P}_{i,j}$, two other vectors are defined: (1) a surface *normal* vector $\mathbf{N}_{i,j} = [-P_{i,j}, -Q_{i,j}, 1]^T$, and (2) a translational *velocity* vector $\mathbf{V}_j = [U_j, V_j, W_j]^T$. As a *first* simplification, these velocity vectors are defined to be *constant* velocities.

The *second* simplification, is to define the surface S as a *plane*, with normal vector $\mathbf{N}_i = [-P_i, -Q_i, 1]^T$. This means that all $\mathbf{N}_{i,j}$ are all equal to $\mathbf{N}_i$. A third simplification is related to the translational velocities. Since the surface S is rigid, *all* the velocity vectors $\mathbf{V}_j$ are equal to one (global) *surface* velocity vector $\mathbf{V} = [U, V, W]^T$.

Simulation of the plane surface S is done using simple vector parameterization. A set of 2 three-dimensional vectors $\mathbf{a}_i = [a_{i,X}, a_{i,Y}, a_{i,Z}]^T$ and $\mathbf{b}_i = [b_{i,X}, b_{i,Y}, b_{i,Z}]^T$, referred to here is the *surface basis* $(\mathbf{a}, \mathbf{b})_i$, is used for defining the exact orientation of plane surface S at time t_i.

At this point we know that a plane surface S translates with linear velocity $\mathbf{V}$. To further *simplify* the notation and simulation, the concept of *rotational velocity*, represented by P' and Q', is *excluded* from these mathematical definitions. The translational velocities of plane surface S are thus *zero* ($P' = 0$ and $Q' = 0$). All normal vectors $\mathbf{N}_i$ are now equal to one global surface normal vector $\mathbf{N}$, which of course is *independent* of the time index i.

The same goes for the surface basis – all $(\mathbf{a}, \mathbf{b})_i$ are now equal to one unique surface basis $(\mathbf{a}, \mathbf{b})$ ($\mathbf{a}_i = \mathbf{a} = [a_X, a_Y, a_Z]^T$ and $\mathbf{b}_i = \mathbf{b} = [b_X, b_Y, b_Z]^T$, for all $i = 1, 2, \ldots, d$). Surface S does thus not rotate over time, it only translates with velocity $\mathbf{V}$.

The *cross product* $\mathbf{N}_{basis} = \mathbf{a} \times \mathbf{b}$ is the non-normalized normal vector of surface S:

$$\mathbf{N}_{basis} = \mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_X \\ a_Y \\ a_Z \end{bmatrix} \times \begin{bmatrix} b_X \\ b_Y \\ b_Z \end{bmatrix} = \begin{bmatrix} a_Y b_Z - a_Z b_Y \\ a_Z b_X - a_X b_Z \\ a_X b_Y - a_Y b_X \end{bmatrix} = \begin{bmatrix} N_X \\ N_Y \\ N_Z \end{bmatrix}$$

The $\mathbf{N}_{basis}$ normal vector is not used here. Instead it is is expressed by vector $\mathbf{N}$ using the surface gradients P and Q:

$$\mathbf{N} = \begin{bmatrix} \frac{a_Y b_Z - a_Z b_Y}{a_X b_Y - a_Y b_X} \\ \frac{a_Z b_X - a_X b_Z}{a_X b_Y - a_Y b_X} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{N_X}{N_Z} \\ \frac{N_Y}{N_Z} \\ 1 \end{bmatrix} = \begin{bmatrix} -P \\ -Q \\ 1 \end{bmatrix}$$

The vector $\mathbf{N}$ has its direction toward the camera, and because of this, the surface S can be said to have a *front side* and a *back side*, where the front side faces toward the camera. The surface basis ($\mathbf{a}$, $\mathbf{b}$) thus defines the exact orientation of the surface S at time t_1. And without rotating this surface basis, the surface S does not rotate and it thus defines the *same* surface orientation at all times t_i.

3.1.4. Surface Point Sampling

The k surface points $\mathbf{P}_{i,j} \in F_i$, which represent a subset (or sampling) of surface S at time t_i, are created using the surface basis ($\mathbf{a}$, $\mathbf{b}$), and a set J of k *surface index* coordinates $(m, n)_j \in \mathbb{Z}^2$: $J = \{(m,n)_1, (m,n)_2, \ldots, (m,n)_k\}$. The exact values of all k surface index coordinates $(m,n)_k$ are defined using a global surface *resolution constant* $r \in \mathbb{N}^+$.

For a surface resolution constant r, the definition of set $J = J_r$ can be written as

$$\begin{gathered} J_r = \{(-r,-r)_1, (-r,1-r)_2, \ldots, (-r,0)_{r+1}, \ldots, (-r,r-1)_{2r}, (-r,r)_{2r+1}, \\ (1-r,-r)_{1+(2r+1)}, \ldots, (1-r,0)_{(r+1)+(2r+1)}, \ldots, (1-r,r)_{2(2r+1)}, \ldots, \\ (0,-r)_{1+r(2r+1)}, \ldots, (0,0)_{(r+1)+r(2r+1)}, \ldots, (0,r)_{(1+r)(2r+1)}, \ldots, \\ (r-1,-r)_{1+(2r-1)(2r+1)}, \ldots, (r-1,0)_{(r+1)+(2r-1)(2r+1)}, \ldots, (r-1,r)_{2r(2r+1)}, \\ (r,-r)_{1+(2r)(2r+1)}, \ldots, (r,0)_{(r+1)+2r(2r+1)}, \ldots, (r,r)_{(2r+1)(2r+1)}\} \end{gathered}$$

For example, with $r = 2$ (see Figure 3.4):

$$\begin{gathered} J_2 = \{(-2,-2)_1, (-2,-1)_2, (-2,0)_3, (-2,1)_4, (-2,2)_5, \\ (-1,-2)_6, (-1,-1)_7, (-1,0)_8, (-1,1)_9, (-1,2)_{10}, \\ (0,-2)_{11}, (0,-1)_{12}, (0,0)_{13}, (0,1)_{14}, (0,2)_{15}, \\ (1,-2)_{16}, (1,-1)_{17}, (1,0)_{18}, (1,1)_{19}, (1,2)_{20}, \\ (2,-2)_{21}, (2,-1)_{22}, (2,0)_{23}, (2,1)_{24}, (2,2)_{25}\} \end{gathered}$$

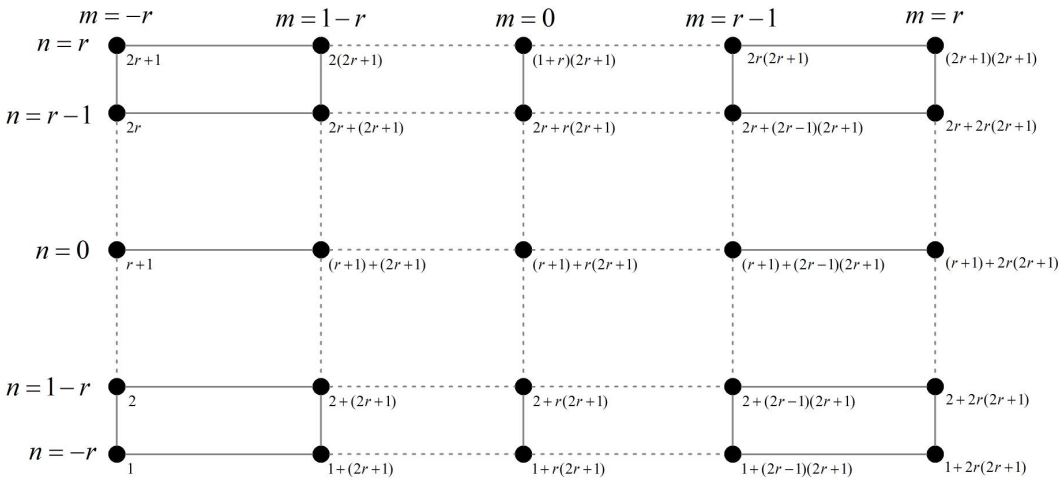


Figure 3.3.: Figure shows the general situation with a point grid of $k = (2r+1)^2$ points. The surface resolution constant is used for defining the exact resolution of the surface point sampling. The value for j, the *point* index, is based in the following column-major *conversion equation*: $j = 1 + r + (1+2r)(r+m) + n$.

The use of the surface resolution constant r is better understood when imagining a two-dimensional square *point grid* of size $2r+1 \times 2r+1$. Such a general point grid is depicted in Figure 3.3. At each point in this grid, an index coordinate $(m,n)_j$ is defined, where j is a *point index* for that specific point. Index coordinate m iterates over columns and n over rows. Each point index j is thus iterated in *column major* form, from the *lower left* point $(-r,-r)_1$, through the *center* point $(0,0)_{(r+1)+r(2r+1)}$, and to the *upper right* point $(r,r)_k$, where $k = (2r+1)^2$.

All the concepts visited so far; surface index coordinates $(m,n)_j$, a surface basis $(\mathbf{a},\mathbf{b})$, a set J with surface index coordinates $(m,n)_j$, a surface resolution constant r, and a point index j; are important formalisms used in the process of surface *point sampling*. The point index j is an implementation detail inherited from the Mathematica scripts, and its primary use is only as a unique numerical identifier for surface (and later image) point. It will be important when section 3.2 takes a look at the actual *data flow*. For more information see figure text in Figure 3.3.

Figure 3.5 shows an even larger point grid, now with $r = 10$. There are a total of $k = (2 \cdot 10 + 1)^2 = 441$ points in this grid. Since the surface resolution constant r defines the *resolution* of the surface point sampling process; i.e., the number of surface points sampled at time t_i, it is important that it is not too large, but also not too small. All experiments, both in this chapter and the next, are carried out with the surface resolution constant set to $r = 10$. This values turned out to be a fine compromise between numerical *exactness* and computational *complexity*.

The points in these point grids are *not* surface points. The surface resolution constant

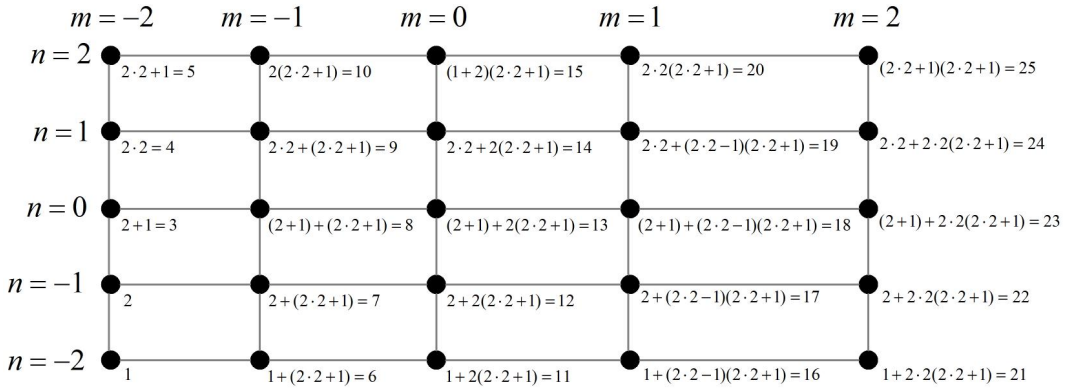


Figure 3.4.: Figure shows an example of the point grid with surface *resolution* constant $K = 2$.

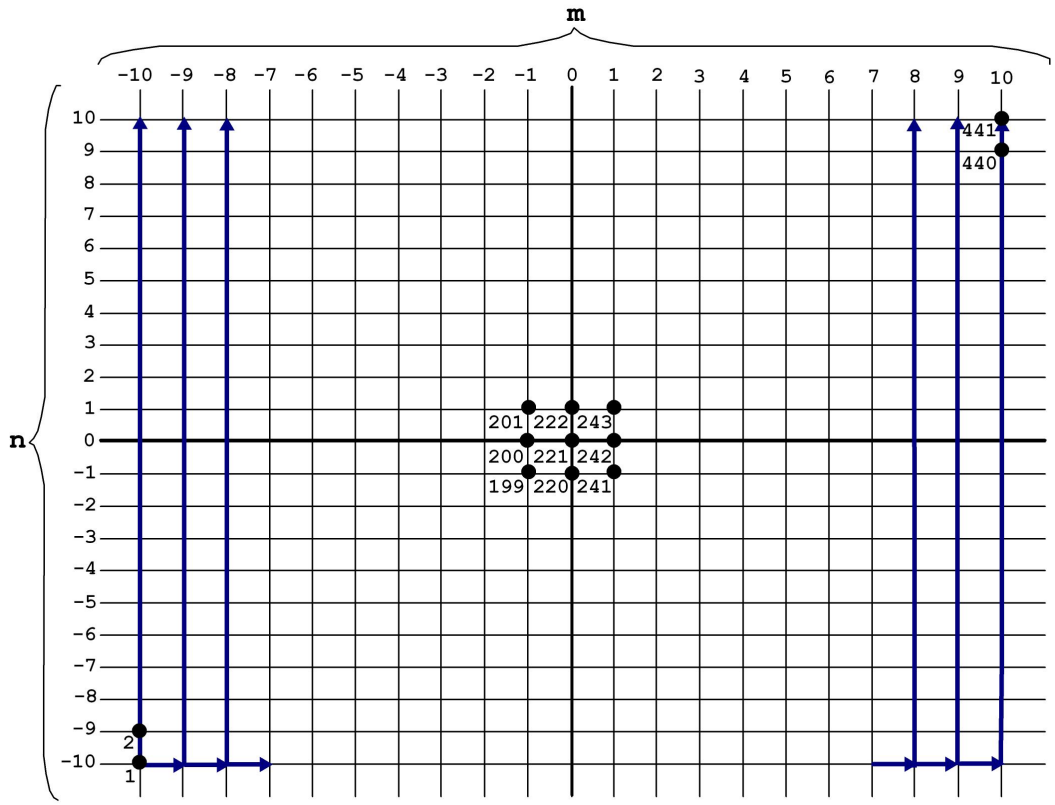


Figure 3.5.: Figure shows an example of the point grid with surface *resolution* constant $r = 10$. The grid has a total of $(2 \cdot 10 + 1)^2 = 441$ points. This is the exact surface resolution used in all the results presented in this chapter and in Chapter 4.

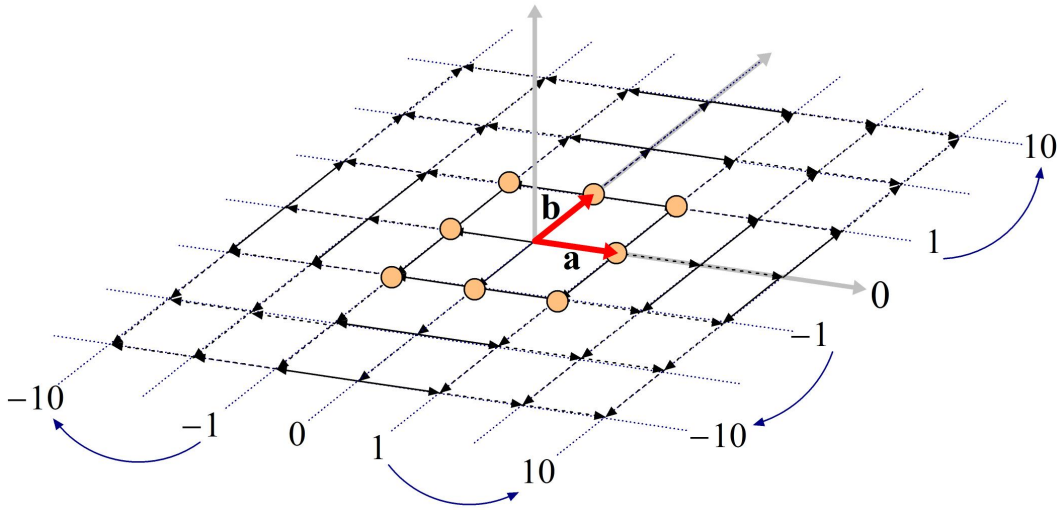


Figure 3.6.: Figure shows how the surface basis $(\mathbf{a}, \mathbf{b})$ is parameterized and used for surface point sampling, with resolution $r = 10$. A surface point $\mathbf{P}_{i,j}$ at time t_i exists at each linear combination of surface basis $(\mathbf{a}, \mathbf{b})$ (see text).

r only defines the resolution of the surface point sampling. To talk about surface points $\mathbf{P}_{i,j}$ at time t_i, we need to add information about surface S and its orientation. This is done using the surface basis $(\mathbf{a}, \mathbf{b})$, mentioned earlier. The plane surface S and the point cloud of surface points $\mathbf{P}_{i,j}$ at time t_i, are a result of a parameterization of the surface basis $(\mathbf{a}, \mathbf{b})$.

Figure 3.6 shows how such a parameterization works. The resolution of the parameterization is determined by the resolution constant r, which has the value $r = 10$. All surface points $\mathbf{P}_{i,j}$ at time t_i are defined (sampled) at each point formed by linear combinations of the surface basis $(\mathbf{a}, \mathbf{b})$. The surface index coordinates m_j and n_j are used as coordinates into this parameterization.

The benefit of using this method of surface point sampling is that (1) the resolution of the sampling is determined exactly by the resolution constant r, and (2) the use of the linear combinations of the surface basis $(\mathbf{a}, \mathbf{b})$ naturally defines both the extent of the sampling on the plane surface S, and (3) it defines a *center* surface point $\mathbf{P}_{1,c}$, with surface index coordinate $(0,0)_c$, where $c = (r+1) + r(2r+1)$. A third vector, a *surface origin* vector $\mathbf{O}_S$, defines the location of this center point at the beginning of time – at time index t_1.

It is now finally possible to give the mathematical relation between surface basis $(\mathbf{a}, \mathbf{b})$, time index i, and the surface index coordinates m_j and n_j. The surface point sampling, the simulation of surface points, is done by using the following vector equation:

$$\mathbf{P}_{i,j} = \mathbf{O}_S + m_j\mathbf{a} + n_j\mathbf{b} + t_i\mathbf{V} \tag{3.1}$$

3.1.5. Numerical *Surface* Texture Density

In the computational experiments, the *surface* texture density is defined purely as a numerical concept. Each surface point $\mathbf{P}_{i,j}$ has a set $D_{S,j}$ of h *surface* texture density values $\mu S_{j,l}$, where $h, l \in \mathbb{N}^+$ with $1 \leq l \leq h$, *mapped* to it:

$$D_{S,j} = \{\mu S_{j,1}, \mu S_{j,2}, \ldots, \mu S_{j,h}\} = \{T_1(m_j, n_j), T_2(m_j, n_j), \ldots, T_h(m_j, n_j)\}$$

Each function T_j defines a *surface* texture density value $\mu S_{j,l}$ at one specific point on the surface, independent of time index i. These functions are only dependent of the surface index coordinates m_j and n_j. Also, the symbol l, is known here as the *texture density index*.

3.1.6. Images and Image Points

An *image* taken by the camera is a set I_i of k image points $\mathbf{p}_{i,j} = [x_{i,j}, y_{i,j}, f]^T$: $I_i = \{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}, \ldots, \mathbf{p}_{i,k}\}$. The camera takes a series of d images. A family of sets H contains these images: $H = \{I_0, I_1, \ldots, I_d\}$.

3.1.7. Numerical *Image* Texture Density

Each image point $\mathbf{p}_{i,j}$, also has a set $D_{I,i,j}$ of h *image* texture density values $\mu I_{i,j,l}$, mapped to it:

$$D_{I,i,j} = \{\mu I_{i,j,1}, \mu I_{i,j,2}, \ldots, \mu I_{i,j,h}\} = \{\tau_{i,j} T_1(m_j, n_j), \tau_{i,j} T_2(m_j, n_j), \ldots, \tau_{i,j} T_h(m_j, n_j)\}$$

The function $\tau_{i,j} \equiv \tau_{i,j}(x_{i,j}, y_{i,j}, P, Q, Z_{i,j}) \equiv \tau_{i,j}(\mathbf{p}_{i,j}, \mathbf{N}, Z_{i,j})$ equates to:

$$\tau_{i,j} = \frac{dS}{dI} = \frac{{Z_{i,j}}^2}{f} \frac{\sqrt{P^2 + Q^2 + 1}}{P x_{i,j} + Q y_{i,j} + f} \tag{3.2}$$

Here $\frac{dS}{dI}$ is the ratio between the area of the *surface* patch at *surface* point $\mathbf{P}_{i,j}$, and the area of the *image* patch at *image* point $\mathbf{p}_{i,j}$.

3.1.8. Summing up

This section has made many definitions and it has also explained in depth, how the plane surface S is "mapped" onto a surface basis parameterization. A considerable effort was put into explaining the surface resolution constant r, which explicitly has been set to $r = 10$. Such detail are considered important for understanding how the simulation works, and it is believed that such an understanding gives the reader the necessary tools to fully understand the results of this entire work.

3.2. Overview of Data Flow

This section will give a brief overview of the data flow involved in the computer simulations. This section will also comments shortly, throughout the text, on certain geometrical details related to the data flow. The data flow is important since it can explain, without too much detail, how the involved parameters and data is used.

The intent with this section is to support the later results from the constraint minimization. The more one understands the data flow behind the results, the more one is likely to understand those results, and be able to critically comment on them.

The level of rigor is minimized though. For instance, the data flow does not cover the algorithms used and how such are programmed. In this section, the examples of data flow, seek to explain, in general, what the *data* represents, and why and when it exists.

The main focus in the data flow examples, is the *point sampling* process. As explained in Section 3.1, the point sampling process consists of three subtasks:

1. Sampling of the necessary *time steps*.
2. Sampling of *surface points* at a *subset* of the sampled time steps.
3. Sampling of *image points* at the same subset of time steps.

Subsection 3.2.1 to 3.2.3 will each explain the data flow of these subtasks.

The subtasks mentioned above are purely *simulation* tasks. When the simulation is done, some form of data structure has been created. This data structure is known here as the “work array”, since it is actually nothing more than a large multi-dimensional array or list.

The *work array* will contain all sampled *surface* and *image* point coordinates, all sampled *surface* and *image* texture densities. It will also contain a *truth-table* with all known geometrical values related to the surface S. These are the normal vector $\mathbf{N}$ and the surface velocity vector $\mathbf{V}$.

When the work-array has been created, the *minimization* process can begin. It also involves three subtasks:

1. Estimate image texture density derivatives for all *user-selected image* points in one image.
2. Construct a *constraint* system, using *constraint* equation Φ or Φ_R, *image* texture density derivatives, and a list of user-selected unknowns.
3. Minimize this created system.

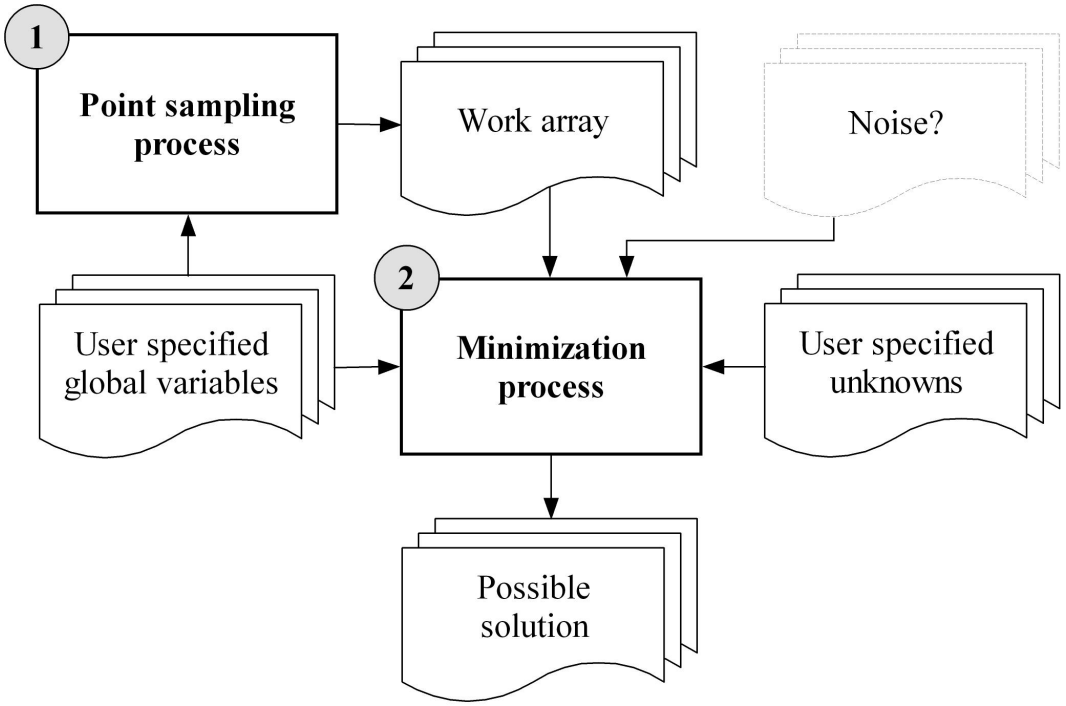


Figure 3.7.: The computational experiments are made possible by two main processes: (1) The *point sampling* process and (2) the *minimization* process. Certain *global* variables need to be defined for the point sampling process to work. The *work array* is needed by the minimization process as well as other things. In the end, a possible constraint solution is returned – which is the very goal with the primary method.

Subsection 3.2.4 will provide a brief overview of the data flow involved in the minimization process.

Figure 3.7 shows a flow chart of the two main processes and the related data and its flow. The figure introduces the concept of *error* in the form of *noise*. Numerical errors could have been simulated and used in the minimization process. Due to time constraints the experiments have been made without the simulation of error.

3.2.1. Sampling of *Time Steps*

The list of d time steps, contains, at index i, the actual time $t_i = (d-1)\Delta t$. The time interval Δt is an important *global* symbol, and at this stage it needs to been already explicitly defined.

Two other global quantities are also defined at this stage: (1) The resolution constant $r = 10$, and (2) the surface basis $(\mathbf{a}, \mathbf{b})$, which defines two *orthogonal* vectors with the same magnitude.

The actual number of time steps, the symbol d, which also later (at least formally) defines the number of *frames* and *images*, is not defined explicitly. Imagine that the surface has a constant velocity component W, such that it moves toward the lens; toward $\mathbf{O}$. At some point in time, if the value of d allowed it, the surface would collide with the XY-plane, and cross over to the other "half-space" on the positive Z-axis. Surface points would now effectively be behind the lens, inside the camera – which would make no sense.

All experiments have been made with $W > 0$. This means that the surface always moves toward the camera somehow. Because of this, the four corner surface points, with surface indexes $(-10,-10)_1$, $(-10,10)_{21}$, $(10,-10)_{421}$, and $(10,10)_{441}$, needs to be sampled to see for what value of i, one of these four corner points is first at $Z = 0$. See Figure 3.8. This particular value of i, referred to as $i_{COLLIDE}$, is used to determine the *actual* number of discrete time steps: $d = i_{COLLIDE} - 1$. Thus time stops just a single time step of Δt, before one of the surface corners collides with the XY-plane.

The time step list can get very large. The simulation is a "one-shot" point sampling process, and it can take some time to sample all points at all times. Because of this, the actual point sampling is only done on a *subset* of the time steps. Because of this, it is important that $d \geq 3$ to give adequate "room" for the subset, which is also the case in all the experiments.

This subset is defined by two parameters: (1) An integer $i_{PADDING}$ and (2) an integer i_{TOI} (Time-Of-Interest), where $0 \leq i_{PADDING} \leq d$ and $i_{TOI} - i_{PADDING} < i_{TOI} < i_{TOI} + i_{PADDING}$. See Figure 3.9. The index i_{TOI} defines where in the time step list the subset is *centered*. The actual time of this time index is t_{TOI}.

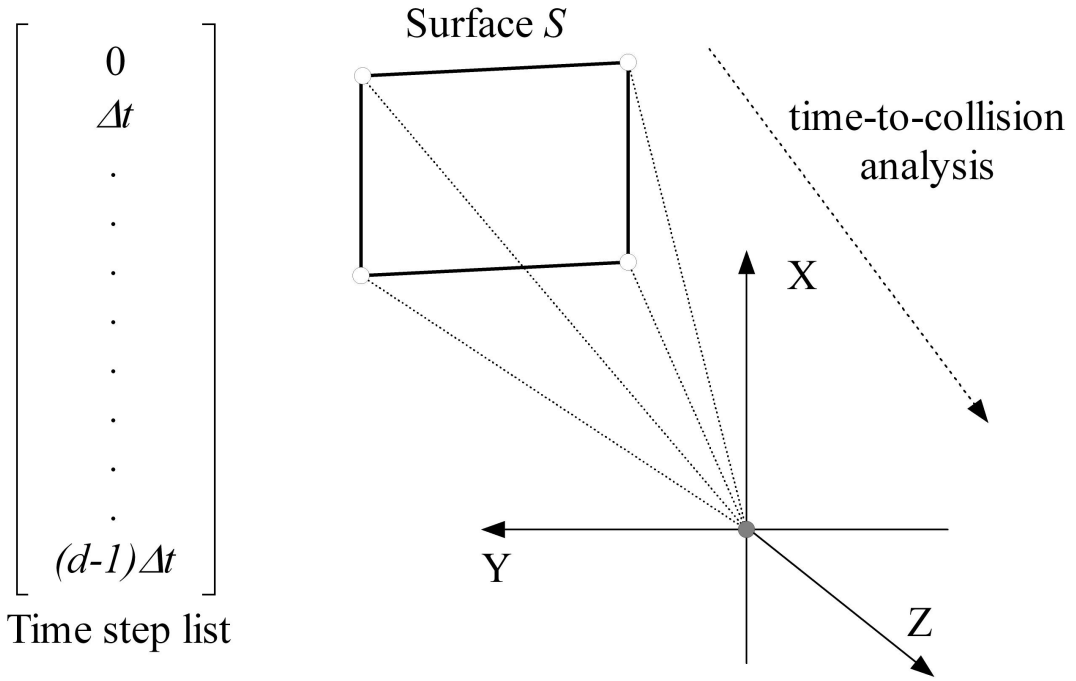


Figure 3.8.: Since $W > 0$ in all experiments, a time-to-collision analysis is used to estimate the limit of *time* itself. It makes no sense to sample surface points at times when they would exist in the "half-space" of the positive Z-axis (see text).

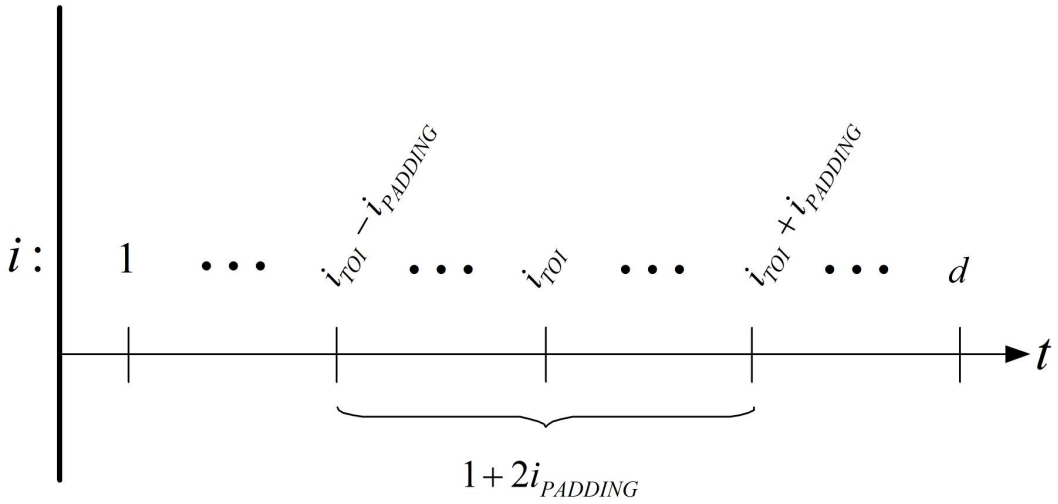


Figure 3.9.: A general example that shows how a *subset* of the full time step list is defined using two parameters: $i_{PADDING}$ and i_{TOI} (see text).

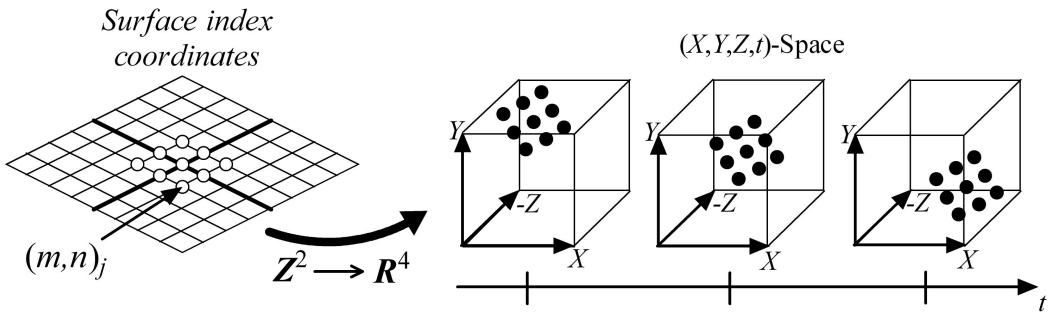


Figure 3.10.: Surface points $\mathbf{P}_{i,j}$ are sampled from surface index coordinates $(m,n)_j$. The sampling can be seen as a *mapping* from $\mathbb{Z}^2$ to $\mathbb{R}^4$; – the (X,Y,Z,t)-space. The *surface* point sampling is done using Equation 3.1.

In the experiments, the time-of-interest index i_{TOI}, is simply the middle time index ($i_{TOI} = (d+1)/2$) or, if d is even, the *lower median* ($i_{TOI} = d/2$). The odd quantity $1 + 2i_{PADDING}$ defines the number of time steps, where surface and image point are actually sampled.

In the experiments $i_{PADDING} = 3$. This number ensures that enough images, i.e. seven images, are eventually processed, for numerical estimation of the partial derivatives of *image* texture density.

3.2.2. Sampling of *Surface Points*

When $i_{PADDING}$ and i_{TOI} has been defined, it is now possible to generate a *subset* of the family of sets G, which contains all *frames* F_i for $i = 1, \ldots, d$ – which each contains surface points for that specific time step.

The sampling of surface points is done by using Equation 3.1. The *time* index i iterates for each of the $1 + 2i_{PADDING}$ *active* time steps (actual time index values determined by i_{TOI}). In each of these "time-iterations", the *point* index j iterates from 1 to $k = (2r+1)^2$, where each of these "point-iterations" corresponds to *one* surface point in *one* frame F_i at time t_i.

Since we know that $k = (2 \cdot 10 + 1)^2 = 441$ and $i_{PADDING} = 3$, the full iteration count is $441 \times (1+2\times 3) = 3087$. For each of these approximate 3000 high-level tasks, it is also necessary to evaluate the h *surface* texture density functions $T_1, \ldots, T_h$, mentioned in Subsection 3.1.5.

Figure 3.10 shows the general idea of the data flow involved in sampling surface points. Each surface index coordinate $(m,n)_j$ can be mapped directly to one surface point $\mathbf{P}_{i,j}$. All surface point indexes, which exists in $\mathbb{Z}^2$, are are mapped to unique (surface) points in $\mathbb{R}^4$; – the (X,Y,Z,t)-space.

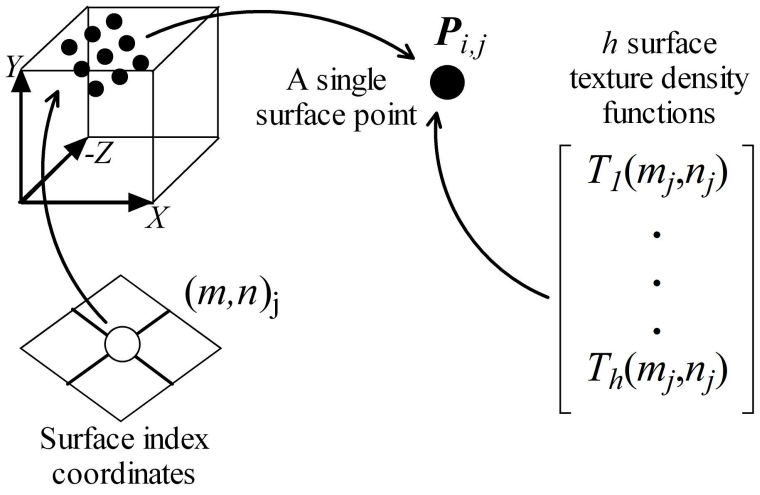


Figure 3.11.: A single surface index coordinate $(m,n)_j$ maps to a single surface point $\mathbf{P}_{i,j}$. For each surface point $\mathbf{P}_{i,j}$ a set of h *surface* texture density values $\mu S_{j,l} = T_l(m_j,n_j)$ is calculated. The functions T_l are *abstract* density functions which can be used as a "model" for the texture density on the surface, at specific surface index coordinates $(m,n)_j$ (see text).

For each surface point, a set of h *surface* texture density values $T_l(m_j,n_j) = \mu S_{j,l}$, for $l = 1, \ldots, h$, is calculated. See Figure 3.11.

In the experiments the number h of *surface* texture density functions is $h = 6$. This means that for each of the exactly 3087 point samplings in $\mathbb{R}^4$, 6 function evaluations are needed. This corresponds to a total of $3087 \times 6 = 18522$ *surface* texture density function evaluations.

3.2.3. Sampling of *Image Points*

When all the 3087 image points in $\mathbb{R}^4$ have been sampled, it is now possible to generate a *subset* of the family of sets F, which contains all *images* I_i for $i = 1, \ldots, d$ – which each contains all image points for that specific time step.

The image points are sampled using the *projective* relations between *camera* and *image* coordinates. These relations were shown in Chapter 1 (Equation 1.2) on page 5, but are shown again for convenience (now with adequate subscripts):

$$x_{i,j} = -f\frac{X_{i,j}}{Z_{i,j}}, \qquad y_{i,j} = -f\frac{Y_{i,j}}{Z_{i,j}} \tag{3.3}$$

Figure 3.12 depicts the general idea behind the data flow in the *image* point sampling procedure. Since there is a *1-to-1* correspondence between all *surface* and *image*

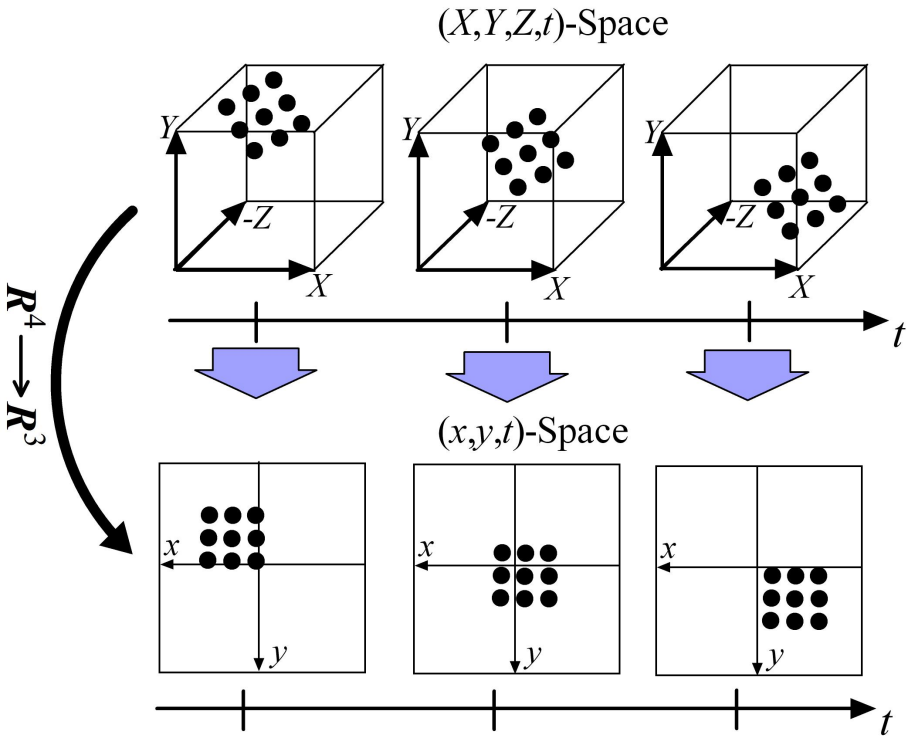


Figure 3.12.: The image point sampling process is generally as *mapping* from $\mathbb{R}^4$ to $\mathbb{R}^3$; – the (x, y, t)-space or *image* space. This mapping is done using Equation 3.3 (see text).

points, a total of 3087 *image* points are also sampled, which gives a sum total of $2 \times 3087 = 6174$.

The *image* points sampling procedure is similar to the *surface* point sampling procedure. Again the *time* index i is iterated over the $1 + 2i_{PADDING}$ active time indexes. For each of these iterations the *point* index is iterated from 1 to 441. At each (i, j)-iteration, an image point $\mathbf{p}_{i,j}$ is sampled using the two equations in Equation 3.3.

For each *image* point, a set of h *image* texture density values $\mu I_{i,j,l} = \tau_{i,j} T_l(m_j, n_j) = \tau_{i,j} \mu S_{j,l}$, for $l = 1, \ldots, h$, is calculated. See Figure 3.13.

The computational experiments only involve *image* texture densities $\mu I_{i,j,l}$. These are *scaled* versions of the "projected" *surface* texture density $\mu S_{j,l}$. And they are scaled with exactly the correct ratio between the corresponding surface and image patches, and this ratio is $\tau_{i,j} = \frac{dS}{dI}$.

The *Texture Density Equation* $\tau_{i,j}$, as defined in Equation 3.2, is an important expression which, by involving *surface* and *image* parameters, equates the necessary ratio $\frac{dS}{dI}$. See also Figure 3.14.

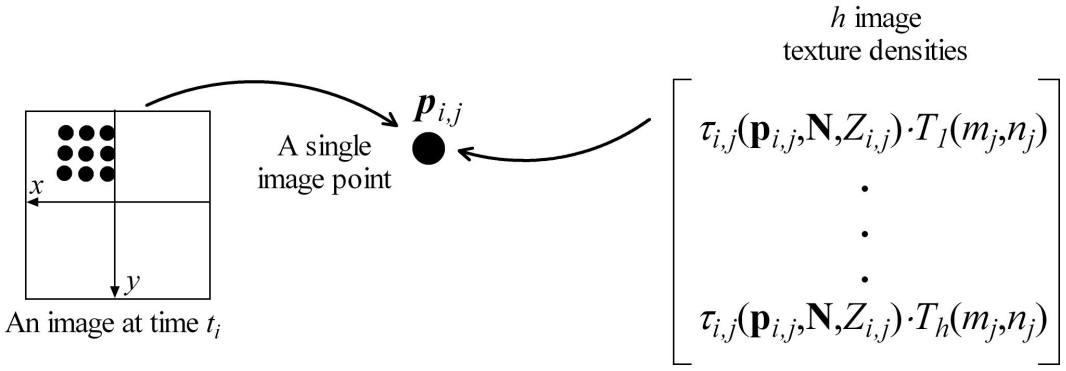
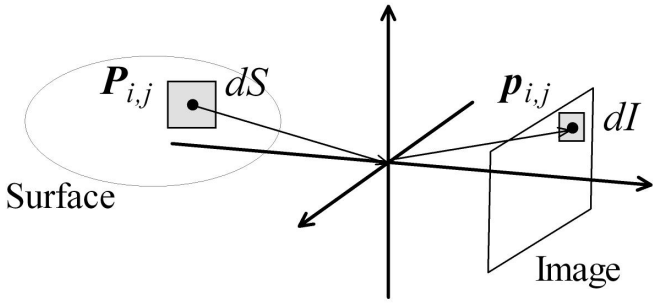


Figure 3.13.: For each *image* point, a set of h *image* texture density values $\mu I_{i,j,l} = \tau_{i,j} T_l(m_j, n_j) = \tau_{i,j} \mu S_{j,l}$, is calculated. These *image* texture densities are scaled *surface* texture densities. The scaling is done using the *Texture Density Function* $\tau_{i,j}$, as defined in Equation 3.2 (see text).



$$\frac{dS}{dI} = \tau_{i,j}(\mathbf{p}_{i,j},\mathbf{N},Z_{i,j}) = \frac{Z_{i,j}^{\ 2}}{f}\frac{\sqrt{P^2+Q^2+1}}{Px_{i,j}+Qy_{i,j}+f}$$

Figure 3.14.: The Texture Density Function $\tau_{i,j}$ equals the ratio between the areas of the *surface* and *image* patches dS and dI. This ratio is $\frac{dS}{dI}$. This is an important formalism used in the synthesis. The dS and dI are never really defined in the computational experiments. Only the ratio between them is; determined by local geometrical measures such as *surface* point distance $Z_{i,j}$, *image* coordinates $(x_{i,j},y_{i,j})$, and the normal vector $\mathbf{N}$ (see text).

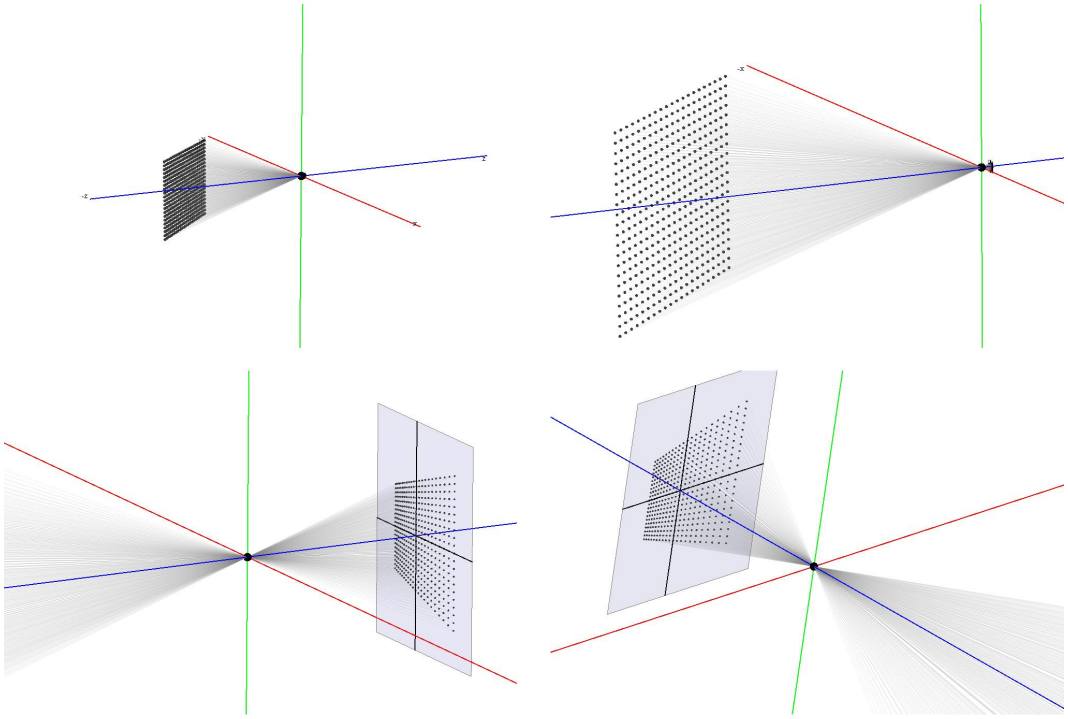


Figure 3.15.: A visualization done using *Mathematica*. The four graphics show how perspective projection is to be understood geometrically. The situation depicts a plane surface with normal vector $[1,0,1]^T$ ($P=-1$, $Q=0$) (see text).

Figure 3.15 gives some examples of the graphical visualization capabilities of Mathematica. The graphics shows four examples of the actual geometry involved in the image sampling process – which of course is simply a perspective projection simulation of mapping 3D surface points to 2D image points using a *focal length* f.

The situation depicted involves a plane surface, approximated by 441 surface points which are coplanar and equidistant in the plane. The plane has normal vector $\mathbf{N} = [1,0,1]^T$, which means that $P=-1$ and $Q=0$. The grays lines depict the rays of light and the dot in the center of the coordinate system is the camera origin $\mathbf{O}$.

The *focal length* is small compared to the size of the surface point cloud (which is determined by the magnitude of the *surface basis* $(\mathbf{a},\mathbf{b})$). This has the effect of also making the actual image point cloud relatively small in extent. The graphics in Figure 3.15 "zooms" in on the visualization, up to the point where the projected image points are visible. They were created using the coordinates estimated in the surface and image point sampling process and by using the Mathematica function "`Graphics3D[]`".

The image points forms a coplanar point cloud at $Z=f$, parallel with the XY-plane – usually referred to as the image plane. Depending on the orientation of the plane

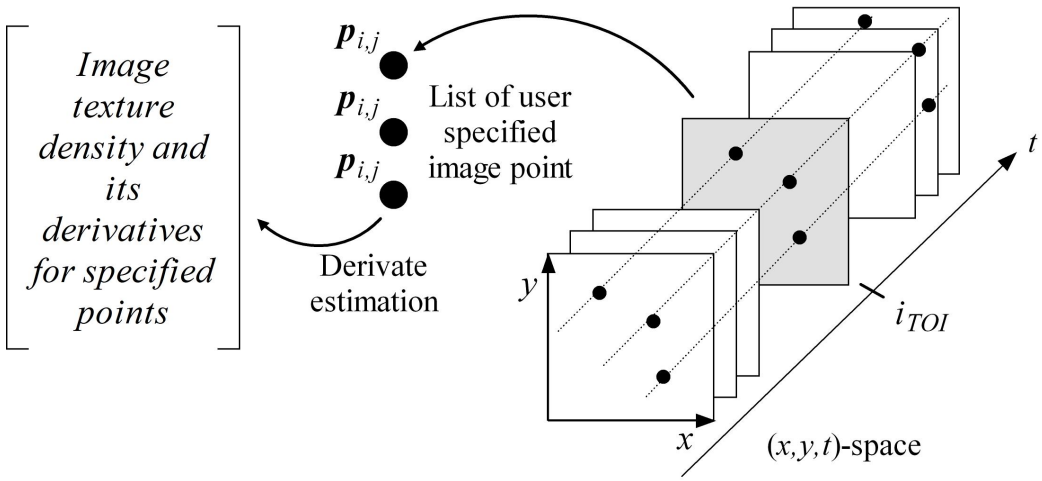


Figure 3.16.: The first subtask in the *minimization* process deals with estimating derivatives. Before this subtask can be begin though, a user needs to select k_{WORK} image points at time t_{TOI}. For each of these image points, *h image* texture density derivatives needs to be estimated (see text).

surface, the projected image points forms a *quadrilateral*[3] region of points on the image plane. When the plane is frontal, i.e. $P = 0$ and $Q = 0$, they form a square region.

In the current situation in Figure 3.15, due to the scaling effect of perspective projection, the surface points farthest away from the camera projects closer[4] together on the image plane. The image points thus form an *isosceles trapezoid*[5] and not a square.

3.2.4. Brief Overview of the Minimization Process

This subsection will take a brief look at data flow involved in the minimization procedure. This will give an overview of how exactly the *image* texture density derivatives and the *constraint* equations Φ and Φ_r, are used to create a *constraint* system.

To avoid the use of too many subscripts some notational abuse seems like a good idea from this point on. The *image* texture density derivatives in x, y and t, are known as $\mu x_{i,j,l}$, $\mu y_{i,j,l}$, and $\mu t_{i,j,l}$, respectively.

The minimization process needs a list of user specified points. The total amount of points that the user specifies is k_{WORK} – it is a *work* set of points, all chosen from the *work array* mentioned in the beginning of this section. The point index j has up to this point been an index used for iteration. In the minimization process the different

[3] A quadrilateral is a polygon with four sides and four vertices.
[4] An example of the difference in image point density, was shown in Subsection 1.2.2 on page 12.
[5] A quadrilateral with two opposite sides parallel and two other sides of equal length.

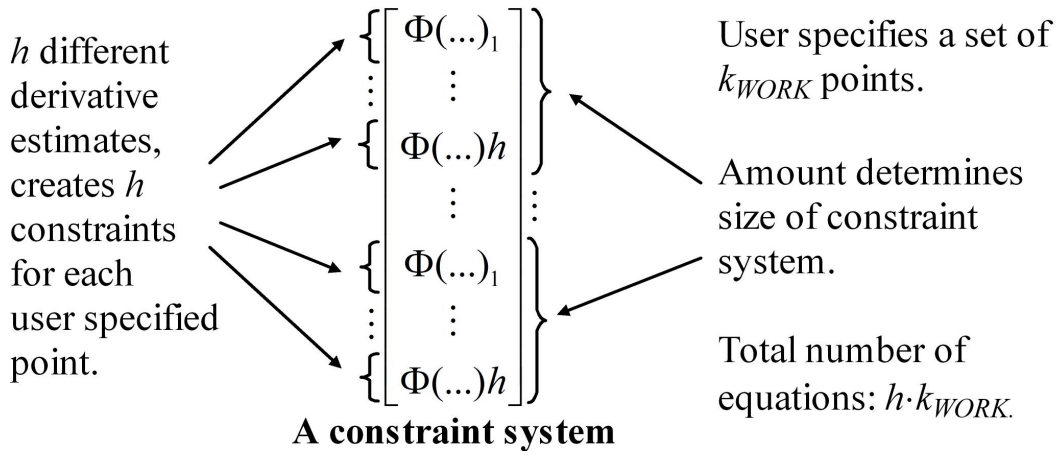


Figure 3.17.: When the all the $h \times k_{WORK}$ derivative estimations are done, where each of those also involves finding *three* partial derivatives in x, y, and t, the minimization process continues with building a *constraint* systems using *constraint* equations Φ. This system contains a total of $h \times k_{WORK}$ equations. The next step is to solve this system (see text).

indexes, from 1 to 441, are now used as unique identifiers for the points selected by the user.

The minimization process begins when the user has selected a set of k_{WORK} points at time t_{TOI}, and the first that happens is an estimation of the *image* texture density derivatives. Figure 3.16 shows in general where we are in the process. The estimation is done in (x, y, t)-space and the estimation of derivatives is done for each user specified image point. There are h *image* texture densities defined for each of these image points, and this results in $h \times k_{WORK}$ derivative estimates. Each of these gives an estimate for $\mu x_{i,j,l}$, $\mu y_{i,j,l}$, and $\mu t_{i,j,l}$, for each image point.

When the derivatives have all been estimated, the *constraint* system is built. Figure 3.17 shows the general idea, using constraint equation Φ. The unknowns of the constraint equation are also user specified but this has no effect on the size of the system. The system always has $h \times k_{WORK}$ equations.

One important thing in the primary method is of course it reliance on the extraction of image texture density, and also the fact that more than *one* density measure can be extracted at any given location in the image. Such considerations are not necessary in the computational experiments. They deal with the synthesis and asks what it really involves to robustly minimize the constraint system. It appears that more than one density measure per image point is indeed necessary for the system to be solvable. Section 3.4 has more to say about this, when actual results of minimization are shown.

3.3. Estimating Partial Derivatives

The computational experiments are fully synthetic. They involve a full algebraic representation of *surface* texture density in the form of the functions $T_1, \ldots, T_h$. *Image* texture density is represented as the same functions, but scaled with the *Texture Density Equation* (or *Function*) $\tau_{i,j}$.

Expressing the algebraic derivatives of these function would probably be possible. But when looking further than pure synthetic simulations, i.e. toward the *camera* experiments which involve *only* real intensity-based *image* texture density measures, algebraic expressions of the partial derivatives cannot be found. A multi-variable fitting would perhaps be possible, but the effort was considered unnecessary.

The derivatives are instead estimated by finding the numerical linear least square solution to a system of Taylor polynomials[6] $\Gamma_p(x, y, t)$ of arbitrary order p.

3.3.1. The Numerical Differentiation Scheme in General

We are seeking to find numerical estimates of the *image* texture density function μ_I which is a function in both time (t) and space (x, y). The estimation is based on the points the user selects at the beginning of the minimization process.

As an example, let us consider just one of these selected image points[7]: $\mathbf{p}_0 = [x_0, y_0, t_0]^T$. Beyond this single image point, a total of $7 \times 441 - 1 = 3086$ points also exists: $\mathbf{p}_1$, $\ldots$, $\mathbf{p}_{3086}$.

This is quite a large number of points to consider. Instead we define a spherical neighborhood with radius R, with $\mathbf{p}_0$ at the center of this sphere. The points considered, will be the points with a distance below or equal to R, from $\mathbf{p}_0$. These points, other than $\mathbf{p}_0$, will be known here equivalently as just $\mathbf{p}_k$, *or* as the points $\mathbf{p}_1, \ldots, \mathbf{p}_k$.

The differences on all three axes, from point $\mathbf{p}_0$ to all other image points $\mathbf{p}_1, \ldots, \mathbf{p}_k$, are always known by simple coordinate analysis. The differences on the x-, y-, and t-axis, from point $\mathbf{p}_0$ to point $\mathbf{p}_k$ are $\Delta x_k = x_k - x_0$, $\Delta y_k = y_k - y_0$, and $\Delta t_k = t_k - t_0$, respectively. See also Figure 3.18.

Given that the points $\mathbf{p}_k$ are *near* $\mathbf{p}_0$, *Taylor's theorem* states that the function value $\mu_I(x_k, y_k, t_k) \equiv \mu I(x_k, y_k, t_k) \equiv \mu I(\mathbf{p}_k)$ can be approximated by the Taylor polynomial $\Gamma_p(x_k, y_k, t_k) \equiv \Gamma_p(\mathbf{p}_k)$ of arbitrary order p. Such polynomials involve the 1st to p'th order partial derivatives.

[6] The use of the Γ-symbol should *not* be confused with the actual *Gamma function* – an extension of the factorial function to real and complex numbers.

[7] The subsets i and j are implicitly defined at all such points, but currently they are not important.

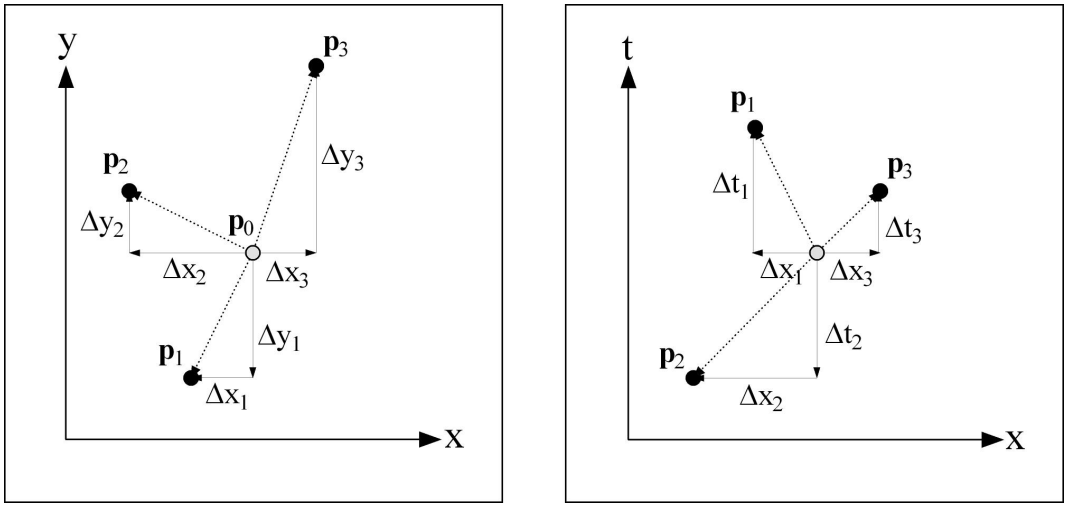


Figure 3.18.: An example with a *current* point $\mathbf{p}_0$ and three other points $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$. The differences on each of the three axes can be determined by simple coordinate analysis. This results in a set of differences, Δx_k, Δy_k, Δt_k for $k = 1, \ldots, 3$. These differences are used to construct a linear system that, when minimized, gives estimates for the three *first* partial derivatives of μ_I.

The function values are already known by definition of the *surface* density functions $T_1, \ldots, T_h$. This means that, for each point $\mathbf{p}_k$, an equality between $\Gamma_p(\mathbf{p}_k)$ and $\mu_I(\mathbf{p}_k)$ can be created, using the differences Δx_k, Δy_k, and Δt_k from point $\mathbf{p}_0$ to point $\mathbf{p}_k$.

When done for all points $\mathbf{p}_k$, the result is a linear system which can be attempted solved for the first partial derivatives $\mu I_x(\mathbf{p}_0)$, $\mu I_y(\mathbf{p}_0)$, and $\mu I_t(\mathbf{p}_0)$ (or minimized in a least square sense).

The unknowns of this system will be the *first* partial derivatives μI_x, μI_y, and μI_t, as well as higher order derivatives when $p > 1$. In the experiments p is at least 9, which includes *ninth*-order partial derivatives. It turned out that the least square minimization method in Mathematica made better estimates of the first derivatives, when higher order derivatives were included.

The linear system with arbitrary order p, for image point $\mathbf{p}_0$, is in general constructed as follows:

$$\begin{bmatrix} \Gamma_p(\mathbf{p}_1) \\ \Gamma_p(\mathbf{p}_2) \\ \vdots \\ \Gamma_p(\mathbf{p}_k) \end{bmatrix} \cdot \begin{bmatrix} \mu I_x(\mathbf{p}_0) \\ \mu I_y(\mathbf{p}_0) \\ \mu I_t(\mathbf{p}_0) \\ \vdots \\ \text{Higher order} \\ \text{derivatives} \end{bmatrix} = \begin{bmatrix} \mu I(\mathbf{p}_1) \\ \mu I(\mathbf{p}_2) \\ \vdots \\ \mu I(\mathbf{p}_k) \end{bmatrix}$$

Note that the remainder terms of the polynomial Γ_p are *not* used and are all consid-

ered *zero*. Using it it would require knowledge of function values between $\mathbf{p}_0$ and the $\mathbf{p}_k$ points, which is not available. The current usage of Taylor polynomials could be seen as a effort in modeling the *image* texture density flow with $p \to \infty$, i.e. with Taylor series, which are the limits of Taylor polynomials where the remainder is exactly *zero*. But of course p is finite and always positive.

The next subsection will take a look at Taylor polynomials in three variables of arbitrary order p.

3.3.2. Taylor's Theorem in *Three* Variables of Arbitrary Order

It took quite an effort finding a working definition of Taylor polynomials in *three* variables. Many multi-variable definitions of *Taylor's theorem* use *multi-index notation* which can be difficult to read. Such notation will *not* be used here.

Readable notation was eventually found on page 277 in *Schaum's Outline of Advanced Calculus* [24]. It gives a two-variable definition of Taylor's theorem of arbitrary order. The notation is easily expanded to three variables.

Taylor's theorem (without remainder terms) for function $\mu_I(x, y, t)$ of arbitrary order p at point $\mathbf{p}_0$ using differences Δx_k, Δy_k, and Δt_k to point $\mathbf{p}_k$, is defined as:

$$\mu_I(x_0 + \Delta x_k, y_0 + \Delta y_k, t_0 + \Delta t_k) \approxeq \Gamma_p(x_0 + \Delta x_k, y_0 + \Delta y_k, t_0 + \Delta t_k) \tag{3.4a}$$

which is equivalent to the shorter notation:

$$\mu I(\mathbf{p}_k) \approxeq \Gamma_p(\mathbf{p}_k)$$

The Taylor polynomial Γ_p at point $\mathbf{p}_0$ (the p'th order approximation at point $\mathbf{p}_k$) are defined recursively. Polynomial Γ_0 is a special case:

$$\Gamma_0(\mathbf{p}_k) = \mu I(\mathbf{p}_0)$$

and for $p > 0$:

$$\Gamma_p(\mathbf{p}_k) = \Gamma_{p-1}(\mathbf{p}_k) + \frac{1}{p!}\left(\Delta x_k \frac{\partial}{\partial x} + \Delta y_k \frac{\partial}{\partial y} + \Delta t_k \frac{\partial}{\partial t}\right)^p \mu I(\mathbf{p}_0) \tag{3.4b}$$

Naturally $x_k = x_0 + \Delta x_k$, $y_k = y_0 + \Delta y_k$, and $t_k = t_0 + \Delta t_k$, and the differences Δx_k, Δy_k, and Δt_k thus appear in Equation 3.4b as independent quantities. For readability, this is important since these differences can be considered as the actual *system parameters*.

One expands $\left(\Delta x_k \frac{\partial}{\partial x} + \Delta y_k \frac{\partial}{\partial y} + \Delta t_k \frac{\partial}{\partial t}\right)^n$ by using the *trinomial theorem* which gives

the power of trinomial sums:

$$(a+b+c)^p = \sum_{s_1+s_2+s_3=p} \frac{p!}{s_1!\,s_2!\,s_3!} a^{s_1} b^{s^2} c^{s_3} \tag{3.5}$$

where s_1, s_2, s_3 are non-negative integers. The summation is taken over all s_i such that $\sum_{i=1}^{3} s_i = n$ for each sum iteration. This means that the sum will only contain terms for which the three iterators s_1, s_2, s_3 sum to 3. Note that, in this definition, quantities of the form 0^0 are *defined* and equal 1.

The notation use $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, and $\frac{\partial}{\partial t}$. When multiplied, they equate partial derivatives. For instance:

$$f(x)\left(\frac{\partial}{\partial x}\right)^2 = \frac{\partial^2}{\partial x^2} = f_{xx} = f^{(2)}$$
$$f(x)\left(\frac{\partial}{\partial x}\right)^n = \frac{\partial^n}{\partial x^n} = f^{(2)}$$
$$f(x,y)\frac{\partial}{\partial x}\frac{\partial}{\partial y} = f(x,y)\frac{\partial^2}{\partial x \partial y} = f_{xy} = f(x,y)\frac{\partial^2}{\partial y \partial x} = f_{yx} = f^{(1,1)}$$

Notice that second and higher order *crossed* partial derivatives are always considered equal: $f_{xy} = f_{yx}$, $f_{xyt} = f_{tyx}$. It is usually assumed, for p times continuously differentiable functions, that it does not matter in what order of variable the differentiation is done. Caution should be advised though, with functions like the *image* texture density μI.

In efforts of actual implementation on digital images, the differentiability of this function is never certain since it originates from a discrete *image operator*.

3.3.3. An Example of Full Taylor Expansion

As an example, $\Gamma_3(\mathbf{p}_k)$ can be expanded. The expression will recursively contain $\Gamma_0(\mathbf{p}_k)$, $\Gamma_1(\mathbf{p}_k)$, and $\Gamma_2(\mathbf{p}_k)$. Rather trivial cases are $\Gamma_0(\mathbf{p}_k) = \mu I(\mathbf{p}_0)$ and $\Gamma_1(\mathbf{p}_k) = \Gamma_0(\mathbf{p}_k) + \Delta x_k \mu I_x(\mathbf{p}_0) + \Delta y_k \mu I_y(\mathbf{p}_0) + \Delta t_k \mu I_t(\mathbf{p}_0)$.

For $\Gamma_2(\mathbf{p}_k)$ we have (all derivatives at point $\mathbf{p}_0$):

$$\begin{aligned}\Gamma_2(\mathbf{p}_k) &= \Gamma_1(\mathbf{p}_k) + \frac{1}{2}\left(\Delta x_k \frac{\partial}{\partial x} + \Delta y_k \frac{\partial}{\partial y} + \Delta t_k \frac{\partial}{\partial t}\right)^2 \mu I(\mathbf{p}_0) \\ &= \Gamma_1(\mathbf{p}_k) + \frac{1}{2}\Delta {x_k}^2 \mu I_{xx} + \Delta x_k \Delta y_k \mu I_{xy} + \frac{1}{2}\Delta {y_k}^2 \mu I_{yy} \\ &\quad + \Delta x_k \Delta t_k \mu I_{xt} + \Delta y_k \Delta t_k \mu I_{yt} + \frac{1}{2}\Delta {t_k}^2 \mu I_{tt}\end{aligned}$$

For P_3 we have (all derivatives at point $\mathbf{p}_0$):

$$\begin{aligned}\Gamma_3(\mathbf{p}_k) &= \Gamma_2(\mathbf{p}_k) + \frac{1}{3!}\left(\Delta x_k \frac{\partial}{\partial x} + \Delta y_k \frac{\partial}{\partial y} + \Delta t_k \frac{\partial}{\partial t}\right)^3 \mu I(\mathbf{p}_0) \\ &= \Gamma_2(\mathbf{p}_k) + \frac{1}{6}\Delta {x_k}^3 \mu I_{xxx} + \frac{1}{2}\Delta {x_k}^2 \Delta y_k \mu I_{xxy} + \frac{1}{2}\Delta x_k \Delta {y_k}^2 \mu I_{yyx} \\ &\quad + \frac{1}{6}\Delta {y_k}^3 \mu I_{yyy} + \frac{1}{2}\Delta {x_k}^2 \Delta t_k \mu I_{xxt} + \Delta x \Delta y_k \Delta t_k \mu I_{xyt} + \frac{1}{2}\Delta {y_k}^2 \Delta t_k \mu I_{yyt} \\ &\quad + \frac{1}{2}\Delta x_k \Delta {t_k}^2 \mu I_{xtt} + \frac{1}{2}\Delta y_k \Delta {t_k}^2 \mu I_{ytt} + \frac{1}{6}\Delta {t_k}^3 \mu I_{ttt}\end{aligned}$$

3.3.4. An Example of a Linear Three-Variable Taylor System

For clarity is seem like a good idea to also give an example of a linear system of Taylor polynomials Γ_p – which could be called a linear tree-variable "*Taylor system*".

The matrix equation for this Taylor system using Γ_p is:

$$\mathbf{G}_p \mathbf{x}_p = \mathbf{b}_p \tag{3.6}$$

For simplicity and space restrictions the system is built using Γ_2. The matrix $\mathbf{G}_2$ looks like this:

$$\mathbf{G}_2 = \begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta t_1 & \frac{1}{2}\Delta {x_1}^2 & \Delta x_1 \Delta y_1 & \frac{1}{2}\Delta {y_1}^2 & \Delta x_1 \Delta t_1 & \Delta y_1 \Delta t_1 & \frac{1}{2}\Delta {t_1}^2 \\ \Delta x_2 & \Delta y_2 & \Delta t_2 & \frac{1}{2}\Delta {x_2}^2 & \Delta x_2 \Delta y_2 & \frac{1}{2}\Delta {y_2}^2 & \Delta x_2 \Delta t_2 & \Delta y_2 \Delta t_2 & \frac{1}{2}\Delta {t_2}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \Delta x_k & \Delta y_k & \Delta t_k & \frac{1}{2}\Delta {x_k}^2 & \Delta x_k \Delta y_k & \frac{1}{2}\Delta {y_k}^2 & \Delta x_k \Delta t_k & \Delta y_k \Delta t_k & \frac{1}{2}\Delta {t_k}^2 \end{bmatrix}$$

Notice that the single Γ_0 term is missing – it is subtracted in $\mathbf{b}_2$.

And the vectors $\mathbf{x}_2$ and $\mathbf{b}_2$:

$$\mathbf{x}_2 = \begin{bmatrix} \mu I_x(\mathbf{p}_0) \\ \mu I_y(\mathbf{p}_0) \\ \mu I_t(\mathbf{p}_0) \\ \vdots \\ \text{see higher order} \\ \text{derivatives above} \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} \mu I(\mathbf{p}_1) - \mu I(\mathbf{p}_0) \\ \mu I(\mathbf{p}_2) - \mu I(\mathbf{p}_0) \\ \vdots \\ \mu I(\mathbf{p}_k) - \mu I(\mathbf{p}_0) \end{bmatrix}$$

3.3.5. Solving the Taylor System

When the point sampling process is done the minimization process begin. As input to this process, a list of image points are needed. A value for order p is also defined.

For *each* of all these selected image points, the matrix equation $\mathbf{G}_p\mathbf{x}_p = \mathbf{b}_p$ is created. The rows of matrix $\mathbf{G}_p$ are based on the k neighborhood points which lies within in a distance of R from the current user selected images point.

Mathematica support least squares minimization using function `LeastSquares[]`. This function seeks to find the $\mathbf{x}_p^*$ which minimizes $\left\|\mathbf{G}_p\mathbf{x}_p^* - \mathbf{b}_p\right\|_2$.

When such a solution $\mathbf{x}_p^*$ has been found, the three first elements, μI_x, μI_y, and μI_t, are used as the partial derivative estimates. These estimates, and the actual value of μI, are hereafter inserted into *constraint* equation Φ (this is done once for the total of h different models of *image* texture density). When the derivatives has been estimated for all user selected points and for all *image* texture densities defined, a final non-linear *constraint* system is constructed and attempted solved (or minimized in a non-linear least square sense).

This section took a detailed look into the work required for numerically estimating derivative. The methods seems very stable as the results in the next section will also show.

The order p is very important. Appendix B shows some graphics of some *constraint trials* with different order p. These trials are created by simulating all known values for P, Q, Z, U, V, and W and inserting these into constraint equation Φ for all image points at time t_{TOI}. The derivatives are hereafter estimated with a specific order p and also inserted into constraint equation Φ for all image points at time t_{TOI}. The exact constraint value should thus be *zero* at all image points. It is evident from Appendix B, that with higher order p, the constraint trial gets closer to zero.

3.4. An Example with Results

The goal of this section is to present some concrete results using the context of a numerical example. It will for the first time involve *physical* quantities with *SI* units. A single situation, calibrated in space and time, involves several variables to defined. Such a single situation is presented in this section and it means that a number of variables will be explicitly defined. It concludes by solving a *constraint system* built using constraint equation Φ. Appendix B will show further results from this example and another with different surface parameters.

3.4.1. Global Setup Parameters

A number of variables, which could be called *global*, have already been defined throughout the text in this chapter. This section briefly combines these definition into a single place, and also makes explicit definitions of other global variables. All

the parameters defined for *surface basis*, translational *velocity*, and time index, surface, and image *point sampling* etc.; are all identical to the calibrated quantities used in the *camera experiments* in Chapter 4. The example shown here, is just the synthetic version of the single example of a camera experiment used in Chapter 4. Figure 3.19 shows the general situation.

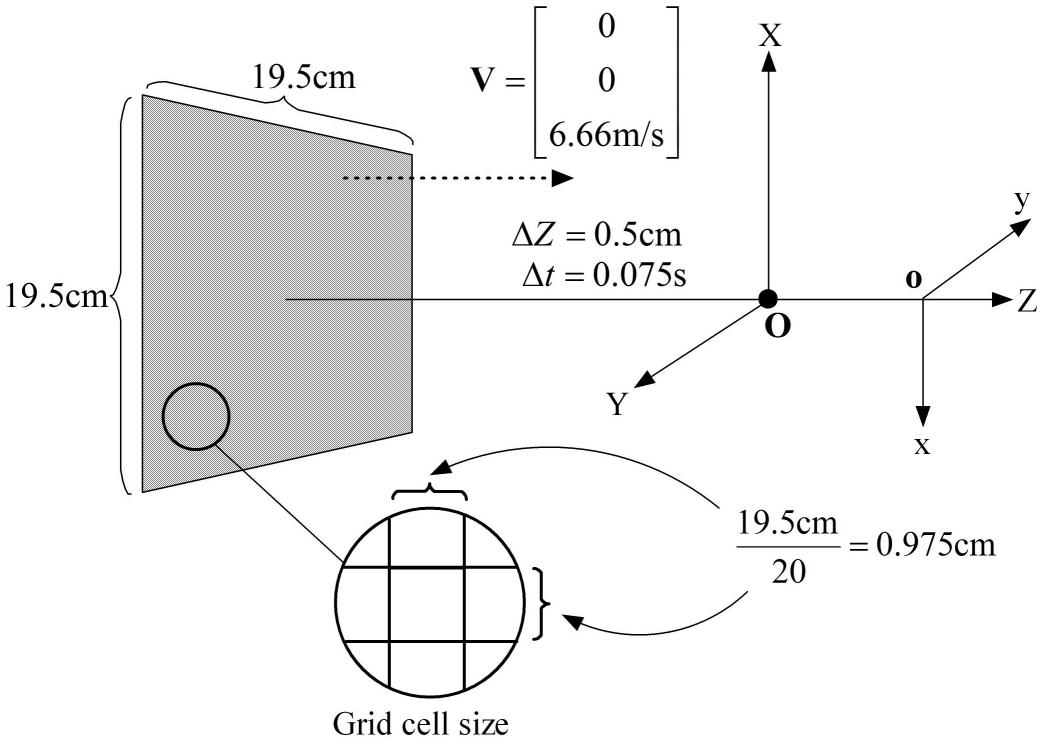


Figure 3.19.: The general situation. A square frontal plane surface translates toward the camera with velocity vector **V**. The square surface had dimension 19.5 cm. The surface points on the surface, are formed on a grid with square cells of dimension 0.975 cm. The movement follows the Z-axis. There is no movement in the X- and Y-axis (see text).

In the following some numerical definitions will be made. These are physical quantities with units. Since the experiments involve the concept of moving surfaces, the scale of distances and length are very important. A good scale of such measures appears to be units in *centimeters* in 10^{-2} m or cm. *Time* is measured/given in *centiseconds* i.e. 10^{-2} s or cs.

The *resolution* constant is $r = 10$. This logically implies that the number of *surface* points and *image* points at any time t_i is $k = 441$. The global symbol d defines the number of frames F_i and images I_i. It was explained in Subsection 3.2.1 that this quantity is determined by a *time-to-collision* analysis with the origin **O**. This was only necessary if $W > 0$, which is exactly the case in this example since $\mathbf{V} = [0, 0, W]$, where $W = \frac{\Delta Z}{\Delta t}$, with $\Delta Z > 0$. The plane thus translates toward the camera in a direct

line following the Z-axis; there is *no* movement in the X- and Y-axis. This is also the case with the camera experiments. Also, the camera focal length is set to $f = 5$ cm.

Variable	Value	Description
r	10	Surface resolution constant.
k	$(2r+1)^2 = 441$	Number of points in frame F_i and image I_i.
$i_{PADDING}$	3	Time index padding for active subset.
d_{ACTIVE}	$1 + 2i_{PADDING} = 7$	Number of *active* time indexes.
f	5 cm	Camera focal length.

Table 3.1.: Parameter Group A – *Global Setup*.

The simulation is only done for a subset of all times; from time index $i = i_{TOI} - i_{PADDING}$ to time index $i = i_{TOI} + i_{PADDING}$. In this example $i_{PADDING} = 3$. Determining i_{TOI} is not possible before d is defined. And the definition of d is independent on the surface basis $(\mathbf{a}, \mathbf{b})$ (which defines $\mathbf{N}$), and the velocity $W = \frac{\Delta Z}{\Delta t}$ m/s. Table 3.1 summarizes the definitions given so far into one group of parameters: Group A – *Global Setup*.

3.4.2. Surface Parameters

The surface basis $(\mathbf{a}, \mathbf{b})$ defines the orientation of plane surface S and the coplanar distance between the surface points $\mathbf{P}_{i,j}$ at time t_i. In this example, as well as in the camera experiments, the plane is *frontal* with $\mathbf{N} = [0, 0, 1]$, i.e. $P = 0$, and $Q = 0$. For surface S to be frontal, the Z-component of vectors $\mathbf{a}$ and $\mathbf{b}$ must be *zero*. Thus the surface basis $(\mathbf{a}, \mathbf{b})$ is a set of vectors, spanning a *vector space* in the (X, Y)-plane *only*.

The size of this basis, i.e. the length of the two vectors $\mathbf{a}$ and $\mathbf{b}$, is determined using calibration parameters of a plane surface in space. In the camera experiments, in Chapter 4, a physical plate with a certain dimension is used. On this plate, a piece square of *textured* cardboard is attached, with a specific dimension of 19.5 cm.

This piece of cardboard can be considered as the subset of the plane surface S, on which surface points $\mathbf{P}_{i,j}$ are defined, at time t_i. The surface points are coplanar and equidistant *in* their plane. The surface basis is defined such that exactly 44l surface points form a rectangular grid on the piece of cardboard. Obviously this grid is the result of the *surface* point sampling procedure. The square grid cells of this grid have the dimension $\frac{19.5}{\sqrt{441}-1}$ cm $= 0.975$ cm. This length is exactly the length of the X-component in $\mathbf{a}$ (Y-component is 0) and the Y-component $\mathbf{b}$ (X-component is 0) (see also Figure 3.19).

The plane surface S has a certain origin in space; its location at time t_1. This surface origin is defined by the *surface origin* vector $\mathbf{O}_S = [X_0, Y_0, Z_0]^T$. The surface origin vector $\mathbf{O}_S$ defines the location (in space) of this *center* point, and it is parallel with the

Variable	Value	Description
a	$[0.9750\text{ cm}, 0.0000\text{ cm}, 0.0000\text{ cm}]^T$	Surface basis vector.
b	$[0.0000\text{ cm}, 0.9750\text{ cm}, 0.0000\text{ cm}]^T$	Surface basis vector.
N	$[0,0,1]$, thus $P=0$ and $Q=0$	Surface normal vector.
$\mathbf{O}_S$	$[X_0, Y_0, Z_0]^T = [0,0,-74.704\text{ cm}]^T$	Surface origin vector.

Table 3.2.: Parameter Group B – *Surface location and orientation*.

$Z-axis$. The Z-axis also intersect the surface at its *center* surface point (at surface index coordinate $(0,0)_{221}$) and $\mathbf{O}_S$ thus also lies *on* the Z-axis. The exact calibrated value is a result from choice: $\mathbf{O}_S = [0,0,-74.704\text{ cm}]^T$. The frontal plane surface S is thus at a distance of exactly 74.704 cm from the camera.

Table 3.2 summarizes the parameters presented in this subsection: Parameter Group B – *Surface location and orientation*.

3.4.3. Time and Velocity Parameters

The distance ΔZ, traveled on the Z-axis per time interval Δt, was explicitly chosen when doing the camera experiment example. The exact quantity is $\Delta Z = 0.5$ cm.

The definition of the time interval Δt is a results of a numerical calibration. At first, a realistic guess is made on its value. The point sampling procedure at time t_{TOI} is hereafter repeated, using all the parameters defined so far, combined with other parameters defined later. On each repeat, the time interval Δt is manually adjusted, until the scale of *time* is comparable to the scale of *image* point coordinates at time t_{TOI}.

Without this calibration, the estimated *image* texture density derivatives contain errors of such magnitude that the final constraint minimization fails. The numerical cause to this, lies with the linear least square minimization used for estimating derivatives.

The time interval was calibrated to $\Delta t = 0.075\ 10^{-2}$s. The surface velocity component in the Z-axis is thus $W = \frac{\Delta Z}{\Delta t} = \frac{0.5\ 10^{-2}\,\text{m}}{0.075\ 10^{-2}\,\text{s}} = 6.667$ m/s.

Knowing the size and orientation of the surface basis and the velocity W, makes it is possible to estimate the *total* number of time indexes d. The time-collision-analysis is simple because the surface is frontal. The surface point used for analysis is the *center* point with surface index coordinate $(0,0)_{221}$.

Equation 3.1 on page 40 defines a vector equation for space coordinates. We are interested in $Z > 0$. The following equation, the third component of point vector

$\mathbf{P}_{d,221}$, is solved for d:

$$\begin{aligned} Z_0 + t_i W = 0 \Leftrightarrow \\ Z_0 + (d-1)\Delta t W = 0 \Leftrightarrow \\ -74.704\ 10^{-2}\,\mathrm{m} + (d-1)\cdot 0.075\ 10^{-2}\,\mathrm{s}\cdot 6.667\,\mathrm{m/s} = 0 \Leftrightarrow \\ d = \lfloor 150.408 \rfloor = 150 \end{aligned}$$

Using the *floor* function, makes sure that d remains integer and that $Z > 0$. This means that a total of $d = 150$ time indexes can be defined before collision with camera origin $\mathbf{O}$ occurs. The middle number of these is number 75, which should the value of d_{TOI}. Instead $d_{TOI} = 11$, simply because only a maximum of exactly 21 images were created in the corresponding camera experiment. Finally, the exact time at time index d_{TOI} is $t_{TOI} = (d-1)\Delta t = (11-1)\cdot 0.075\ 10^{-2}\,\mathrm{s} = 0.75\ 10^{-2}\,\mathrm{s}$.

Variable	Value	Description
Δt	$0.075\ 10^{-2}\,\mathrm{s}$	Time interval.
ΔX	0 cm	X-axis distance interval.
ΔY	0 cm	Y-axis distance interval.
ΔZ	0.5 cm	Z-axis distance interval.
$\mathbf{V}$	$[U,V,W]^T$	Surface velocity vector.
U	0 m/s	X-axis velocity component.
V	0 m/s	Y-axis velocity component.
W	$\frac{0.5}{0.075}\,\mathrm{m/s} = 6.667\,\mathrm{m/s}$	Z-axis velocity component.
d	149	Total number of time indexes.
i_{TOI}	11	*Time-Of-Interest* index.
t_{TOI}	$(d-1)\Delta t = 0.75\ 10^{-2}\,\mathrm{s}$	*Time-Of-Interest* time.
R	0.25 cm	Derivative estimation (x,y,t)-radius.

Table 3.3.: Parameter Group C – *Time and velocity*.

Table 3.3 summarizes the parameters presented in this subsection: Parameter Group C – *Time and velocity*.

3.4.4. Surface Texture Density Functions

The surface texture density is modeled using $h = 6$ functions $T_l(m,n)$. These functions are defined in Table 3.4: Parameter Group D – *Surface texture density functions*.

The nature of these functions is basically a result of trial and error. This set of six functions appears to give a functional model of texture density – i.e. a synthetic model which makes it possible to minimize the constraint system in the end and estimate the 3D surface structure. The results presented in this report, were carried using *all* these functions and *only* these functions.

Variable	Value
$T_1(m,n)$	m
$T_2(m,n)$	n
$T_3(m,n)$	$m+n$
$T_4(m,n)$	$\cos n$
$T_5(m,n)$	$\cos m$
$T_6(m,n)$	$\cos m + \cos n$

Table 3.4.: Parameter Group D – *Surface texture density functions*.

Figure 3.20 shows the six *surface* texture density functions $T_1, \ldots, T_6$, in the form of 2-dimensional *density plots*. Certainly these six functions are infinitely and continuously differentiable.

Though it certainly would be possible, no algebraic expressions of the corresponding *image* texture density functions will be given. Such expression will be dependent on a large number of variables and since these explicit expressions were never used, it would serve no purpose showing them here.

3.4.5. Parameters for Partial Derivative Estimation

The radius R used in the derivative estimation method, was also a manually calibrated quantity. It was calibrated to 0.25 cm. The order used is $p = 12$.

Variable	Value	Description
R	0.25 cm	Derivative estimation (x, y, t)-radius.
p	12	Chosen order of Taylor System

Table 3.5.: Parameter Group E – *Derivative Estimation*.

For clarity these two parameters are listed are also defined in Table 3.5: Parameter Group E – *Derivative Estimation*.

3.4.6. User Specified Image Points

The many variable definitions up until now has been absolutely necessary for the sake of clarity. In the simulation run of the computational experiment example, we have now just finished with the point sampling procedure. The next procedure, the minimization procedure, needs a list of k_{WORK} image points to continue.

Of course there is the question: Why even bother with selecting more than *one* point? The reason is related to the final goal, minimization of a constraint system. A *first result* of considerable importance, is the fact that the *solvability* of the system is *not*

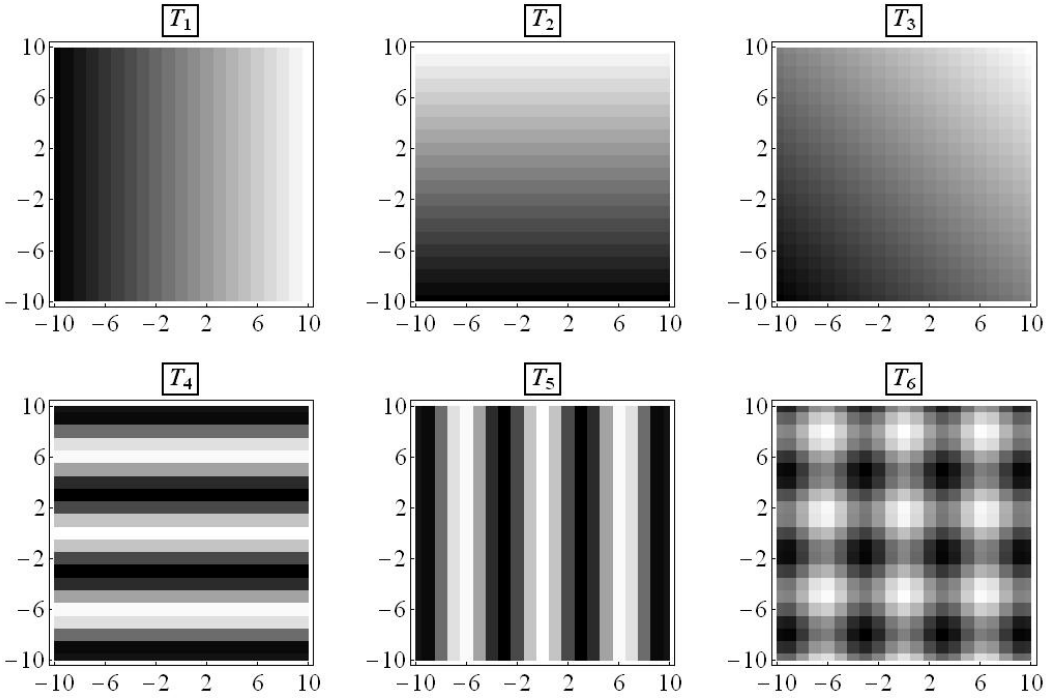


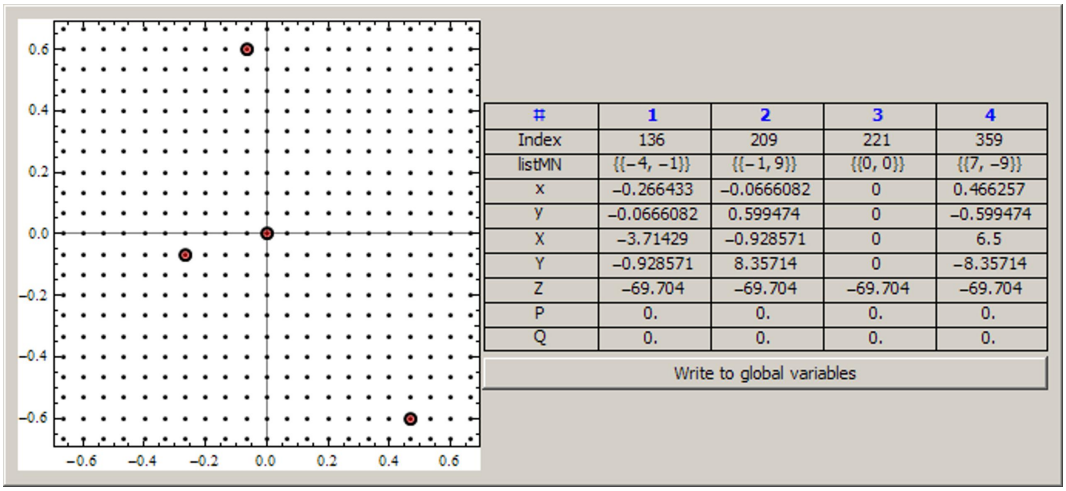
Figure 3.20.: Density plot of the $h = 6$ *surface* texture density functions T_l, for $l = 1, \ldots, 6$.

Variable	Value	$(m,n)_j$
$\mathbf{p}_{11,136}$	$[-0.2664, -0.0666]^T$	$(-4,-1)_{136}$
$\mathbf{p}_{11,209}$	$[-0.0666, 0.5995]^T$	$(-1,9)_{209}$
$\mathbf{p}_{11,221}$	$[0.0000, 0.0000]^T$	$(0,0)_{221}$
$\mathbf{p}_{11,356}$	$[0.4663, -0.5995]^T$	$(7,-9)_{356}$

Table 3.6.: Parameter Group F – *User selected image points*.

good when only *one* image point is used. On the other hand, when *two* or *more* image points are used, the solvability seems almost *guaranteed* (for variables P, Q, and Z).

A set of $k_{WORK} = 4$ points are chosen. Figure 3.21 shows a GUI created with Mathematica. This GUI makes it possible, rather comfortably, to choose a set of image points. To the left in the GUI the current set of image points are shown (the *image* I_i for $i = i_{TOI}$ which is *image* $G_1 1$). The larger dots are the ones which have been selected for further analysis. To the right in the GUI, the *truth-table* is shown. For each of the four image points shown selected, the truth table shows (1) the corresponding point iterator index j, (2) the corresponding surface index coordinate (m,n), (3) the image x-coordinate, (4) the image y-coordinate, (5) the corresponding space X-coordinate, (6) the corresponding space Y-coordinate, (7) the corresponding space Z-coordinate (depth), (8) the corresponding X-axis surface gradient P, and (9) the corresponding Y-axis surface gradient Q.



#	1	2	3	4
Index	136	209	221	359
listMN	{{−4, −1}}	{{−1, 9}}	{{0, 0}}	{{7, −9}}
x	−0.266433	−0.0666082	0	0.466257
y	−0.0666082	0.599474	0	−0.599474
X	−3.71429	−0.928571	0	6.5
Y	−0.928571	8.35714	0	−8.35714
Z	−69.704	−69.704	−69.704	−69.704
P	0.	0.	0.	0.
Q	0.	0.	0.	0.

Figure 3.21.: Version 6 of Mathematica has advanced capabilities for creating *dynamic* content such as GUIs. This dynamic GUI makes it possible to choose a number of image points for further consideration in the minimization process. To the left of the image points chosen, the *truth-table* for the image points is shown. This truth table contains much of the information which one seeks to estimate using constraint minimization (see text).

Variable	Value	$(m,n)_j$
$\mathbf{P}_{11,136}$	$[-3.7143, -0.9286, -69.7040]^T$	$(-4,-1)_{136}$
$\mathbf{P}_{11,209}$	$[-0.9286, 8.3571, -96.7040]^T$	$(-1,9)_{209}$
$\mathbf{P}_{11,221}$	$[0.0000, 0.0000, -69.7040]^T$	$(0,0)_{221}$
$\mathbf{P}_{11,356}$	$[6.5000, -8.3571, -69.7040]^T$	$(7,-9)_{356}$

Table 3.7.: Parameter Group G – *Corresponding surface points*.

The image points selected in Figure 3.21 are the points $\mathbf{p}_{11,136}$, $\mathbf{p}_{11,209}$, $\mathbf{p}_{11,221}$, and $\mathbf{p}_{11,356}$. Their coordinates and corresponding surface index coordinate are shown in Table 3.6: Parameter Group F – *User selected image points*.

The coordinates of the corresponding surface points $\mathbf{P}_{11,136}$, $\mathbf{P}_{11,209}$, $\mathbf{P}_{11,221}$, and $\mathbf{P}_{11,356}$ are shown in Table 3.7: Parameter Group G – *Corresponding surface points*.

For completeness the numerical values for the estimated partial derivatives of the *image* texture density in all of the 4 selected points, are also shown in Table 3.8: Parameter Group H – *Image texture density derivatives*.

j	l	μI	μI_x	μI_y	μI_t
136	1	-7.7738×10^{2}	2.9177×10^{3}	5.6843×10^{-14}	2.2305×10^{2}
	2	-1.9435×10^{2}	-1.3642×10^{-12}	2.9177×10^{3}	5.5763×10^{1}
	3	-9.7173×10^{2}	2.9177×10^{3}	2.9177×10^{3}	2.7882×10^{2}
	4	1.0501×10^{2}	9.0768×10^{-8}	2.4408×10^{3}	-4.5369
	5	-1.2703×10^{2}	-2.1952×10^{3}	1.024×10^{-9}	-3.1639×10^{1}
	6	-2.9057×10^{2}	-2.1952×10^{3}	1.5672×10^{3}	9.627
209	1	-1.9435×10^{2}	2.9177×10^{3}	-4.4338×10^{-12}	5.5763×10^{1}
	2	1.7491×10^{3}	1.7053×10^{-12}	2.9177×10^{3}	-5.0187×10^{2}
	3	1.5548×10^{3}	2.9177×10^{3}	2.9177×10^{3}	-4.4611×10^{2}
	4	-1.7707×10^{2}	7.617×10^{-12}	-1.3243×10^{3}	1.098×10^{2}
	5	1.0501×10^{2}	2.4408×10^{3}	1.0997×10^{-5}	-4.5369
	6	1.851×10^{2}	2.4408×10^{3}	-2.6977×10^{3}	1.3481×10^{2}
221	1	$0.$	2.9177×10^{3}	1.4211×10^{-12}	2.4158×10^{-13}
	2	$0.$	-3.2969×10^{-12}	2.9177×10^{3}	-1.0232×10^{-12}
	3	$0.$	2.9177×10^{3}	2.9177×10^{3}	-3.2969×10^{-12}
	4	1.9435×10^{2}	6.8784×10^{-6}	5.195×10^{-6}	-3.7175×10^{1}
	5	1.9435×10^{2}	8.9655×10^{-6}	7.1995×10^{-6}	-3.7175×10^{1}
	6	1.9435×10^{2}	8.9546×10^{-6}	2.9006×10^{3}	-3.7175×10^{1}
359	1	1.3604×10^{3}	2.9177×10^{3}	1.6826×10^{-11}	-3.9034×10^{2}
	2	-1.7491×10^{3}	-5.9117×10^{-12}	2.9177×10^{3}	5.0187×10^{2}
	3	-3.8869×10^{2}	2.9177×10^{3}	2.9177×10^{3}	1.1153×10^{2}
	4	-1.7707×10^{2}	-3.73×10^{-11}	1.3243×10^{3}	1.098×10^{2}
	5	1.4652×10^{2}	-1.9057×10^{3}	3.5587×10^{-5}	5.6955×10^{1}
	6	6.6424×10^{1}	-1.9057×10^{3}	-2.6977×10^{3}	-8.2396×10^{1}

Table 3.8.: Parameter Group H – *Image texture density derivatives*.

3.4.7. Minimization Results in General

The remaining subsections of this section will each present a certain minimization *situation* and *result*. These different situations are based on different choices of image points (out of the set of four already defined) and on the choices of unknowns.

Some methods for solving or minimizing non-linear system, requires an initial point for every variable solved for. Since it was never a goal in this work to make a thorough investigation into the exact numerical behavior of these constraint systems, the initial point is simply set to *zero* in all minimization attempts and for all variables.

Subsection 3.4.8 focuses on *single point* minimization. These are situations where only *one* specific image point is used for building the constraint system. All 6 texture density models/functions are still used. Such constraint systems will thus contain 6 different equations, all based on constraint equation Φ.

Subsection 3.4.9 focuses on multiple point minimization. This refers to minimization attempts using *more* than just one image point to build the constraint system. The number of points used in this example is the exact number of user selected image points: $k_{WORK} = 4$. The number of equations in the constraint system is thus $h \cdot k_{WORK} = 6 \cdot 4 = 24$.

3.4.8. Single Point Minimization

The constraint system can be built using one or more image points. In each case it will create a non-linear system of $h = 6$ equations, each corresponding to the different model/function of image texture density.

The minimization was done for each of the four already chosen image points. This subsection will show a small subset of the results from these minimization situations. The difference of situation lies in the choice of number of variables. There are a total of six unknowns in the constraint equation Φ. These are P, Q, Z, U, V, W. There are 6 situations with one variable, 15 with two, 20 with three, 15 with four, 6 with five, and 1 situation with all six variables. In total there are 63 combinations of these six unknown.

The results for all 63 combinations, for two of the selected image points, can be found in Appendix B. Tables B.1 to B.6 show the full numerical results achieved when using different methods of minimization. This subsection will show a few minimization results and some 2D and 3D plots which should convey clearly, the characteristics of the different situations of minimization, when attempted using local and/or global search techniques.

Figures 3.22 to 3.25 show the two-dimensional plots resulting when attempting single point, single variable, minimization. Plots are shown for all four image points originally selected, and for all six variables.

The thin lines in the plots corresponds to the equations in the constraint system. For a single point, for a single variable; there exists $h = 6$ of such lines in the corresponding constraint system. The thick curve is the squared norm-2 of the system. The black point is the correct known value, taken from the truth-table.

The minimum of these curves is a global minimum and noticeably this minimum is correct in *all* plots, within errors of $\pm 10^{-2}$. Though the plots are only for one single variable in one single image point, this is still an important step in the right direction. The reader is encouraged to compare these plots with tables of parameters presented earlier in this section.

Remember, for instance, that the current scene depth is $Z = -69.704\,\text{cm}$ and the current velocity on the Z-axis is $W = -6.667\,\text{m/s}$. The minimization at all four image points, for Z and W, is likely to be very acceptable. This is indeed also the case, which can seen in Tables B.1 and B.4 for image points $\mathbf{p}_{11,336}$ and $\mathbf{p}_{11,359}$.

Figure 3.26 shows a 3D plot of the two-variable case for P and Q, for image point $\mathbf{p}_{11,136}$. The planes in the image corresponds to the 6 equations in the constraint system. The surface thus naturally corresponds to the squared norm-2 of the constraint system.

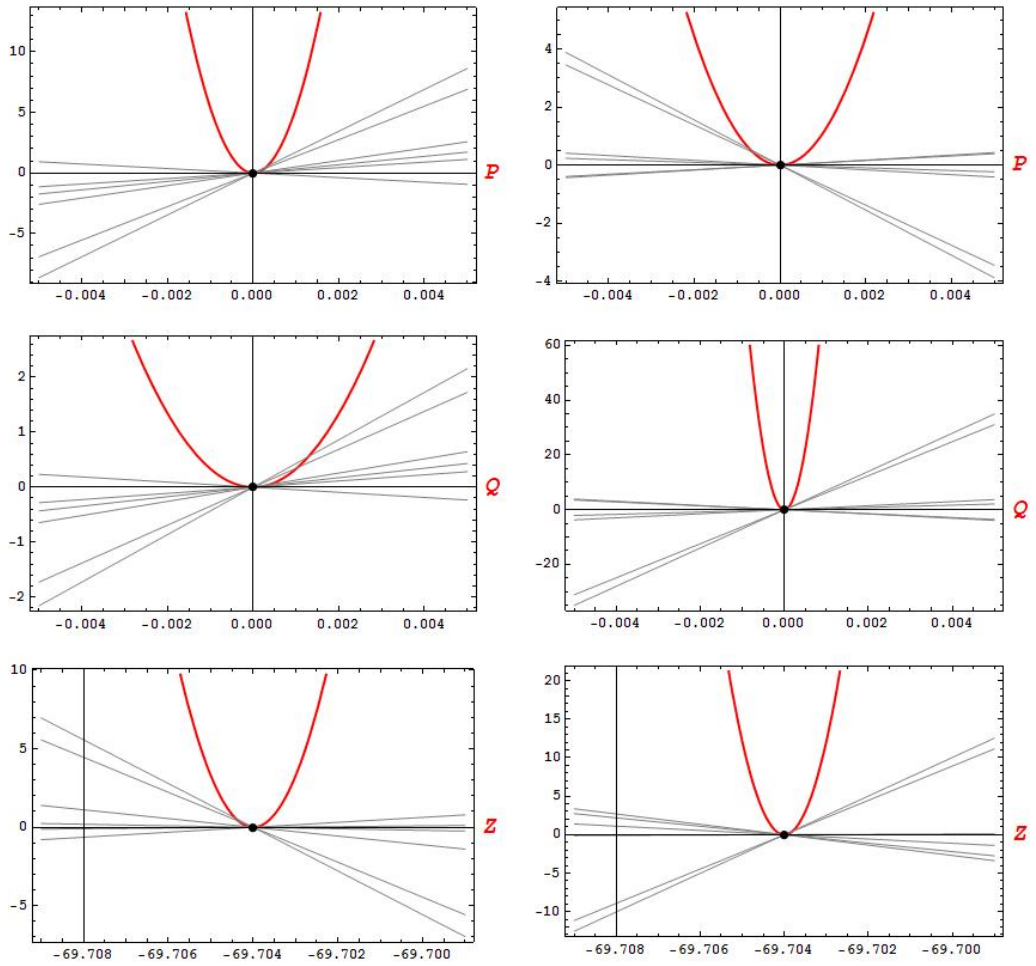


Figure 3.22.: Plots for single point minimization for P, Q, and Z. The two columns show plots for image points $\mathbf{p}_{11,136}$ and $\mathbf{p}_{11,209}$.

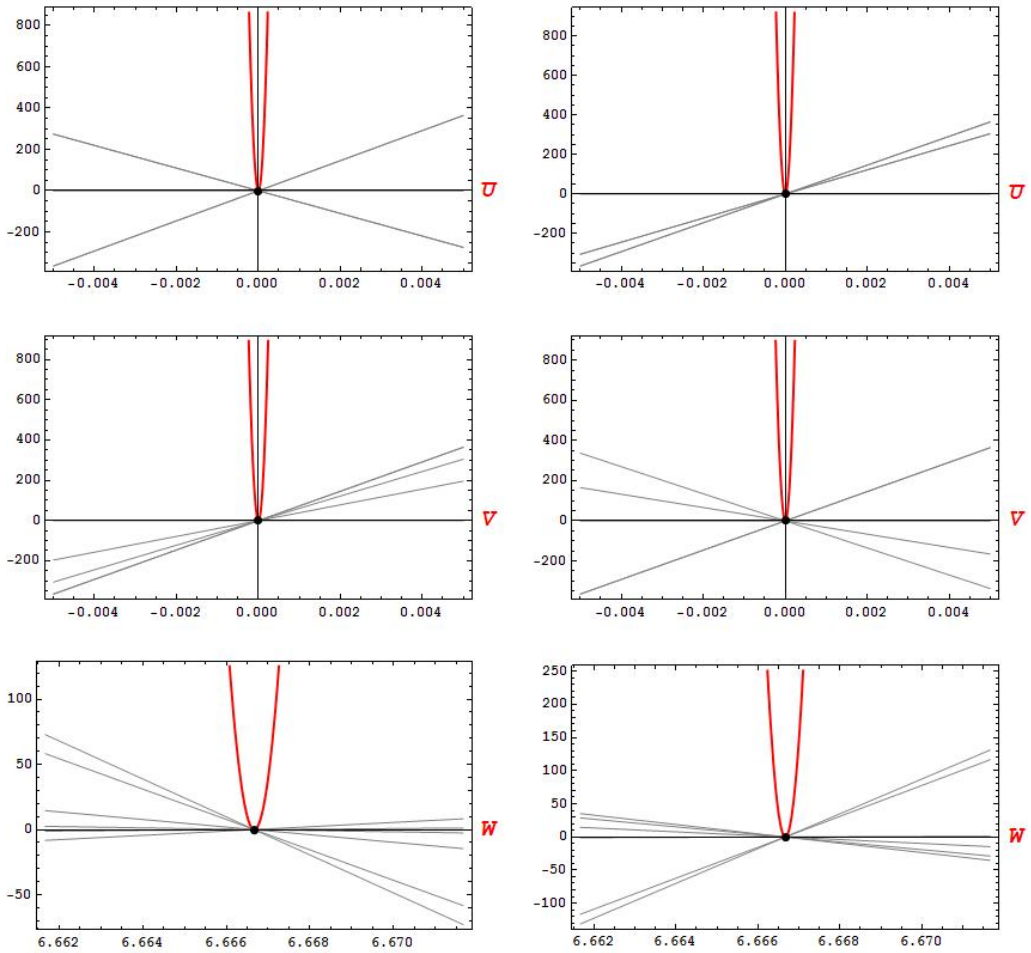


Figure 3.23.: Plots for single point minimization for U, V, and W. The two columns show plots for image points $\mathbf{p}_{11,136}$ and $\mathbf{p}_{11,209}$.

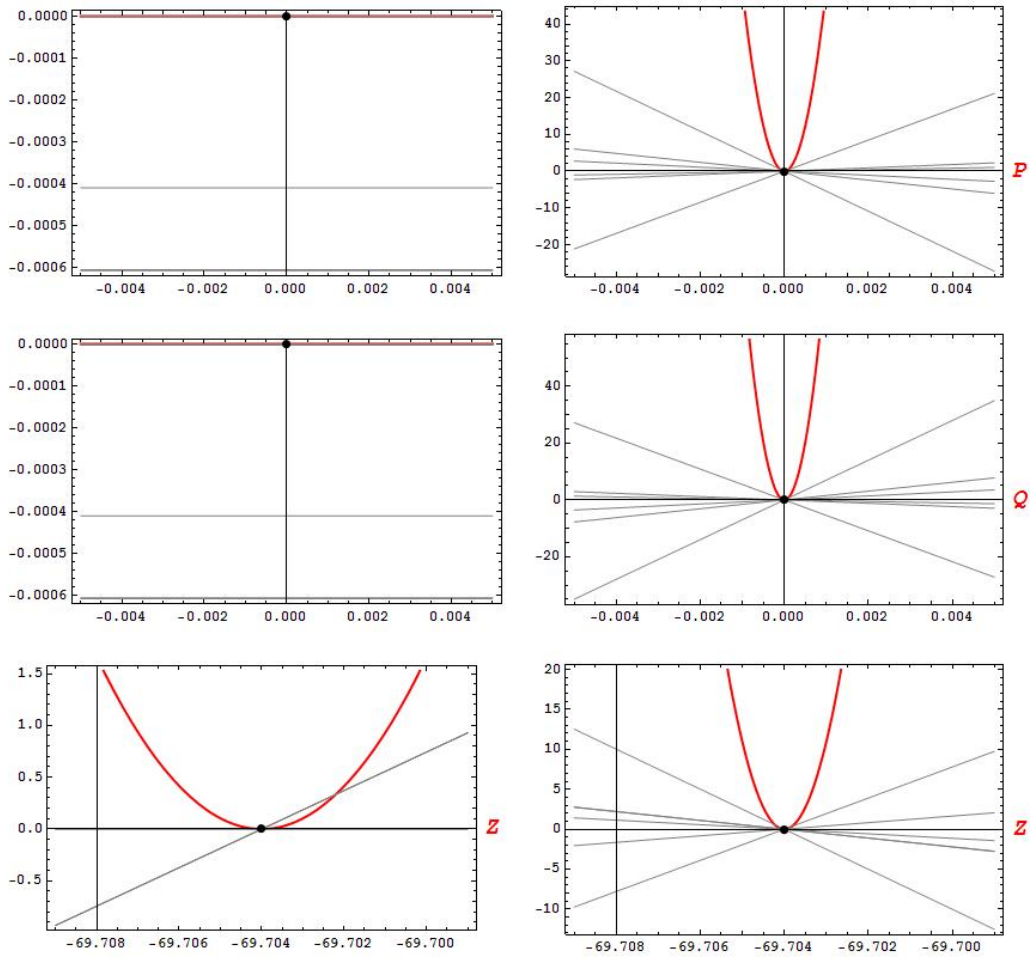


Figure 3.24.: Plots for single point minimization for P, Q, and Z. The two columns show plots for image points $\mathbf{p}_{11,221}$ and $\mathbf{p}_{11,359}$.

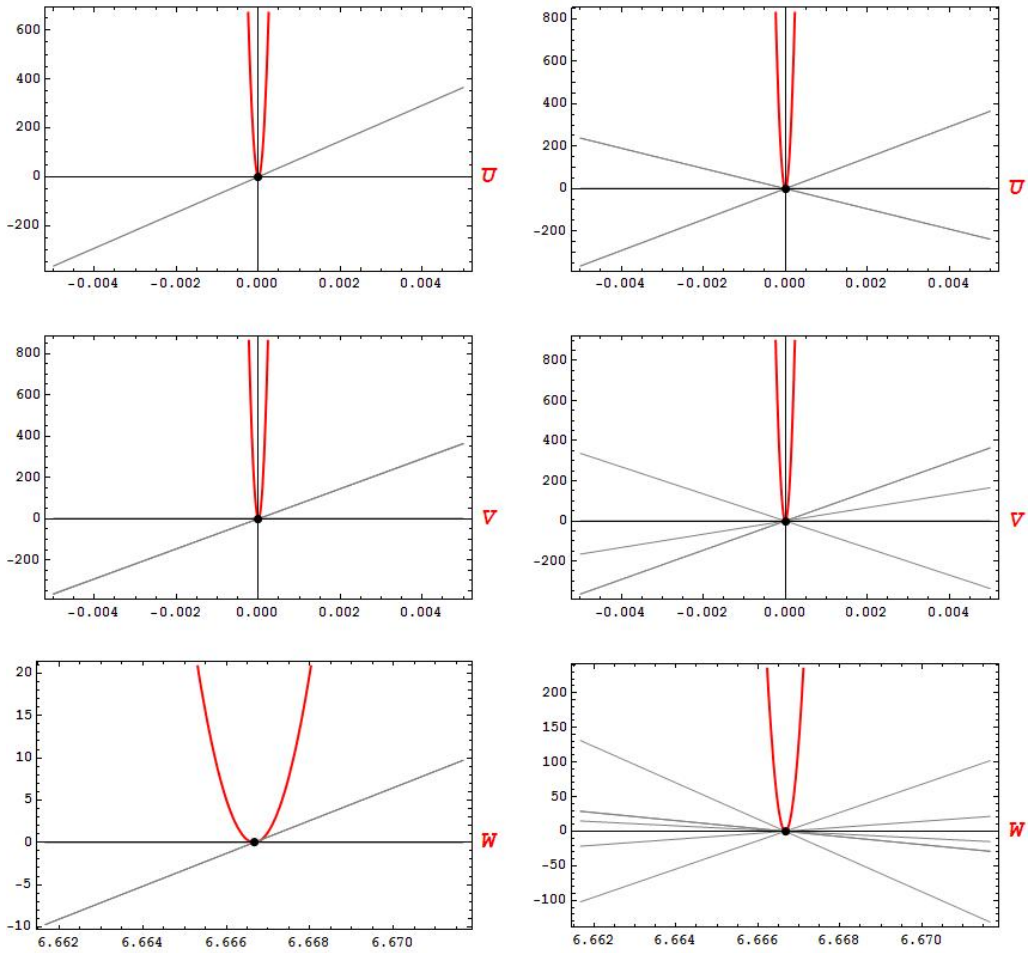


Figure 3.25.: Plots for single point minimization for U, V, and W. The two columns show plots for image points $\mathbf{p}_{11,221}$ and $\mathbf{p}_{11,359}$.

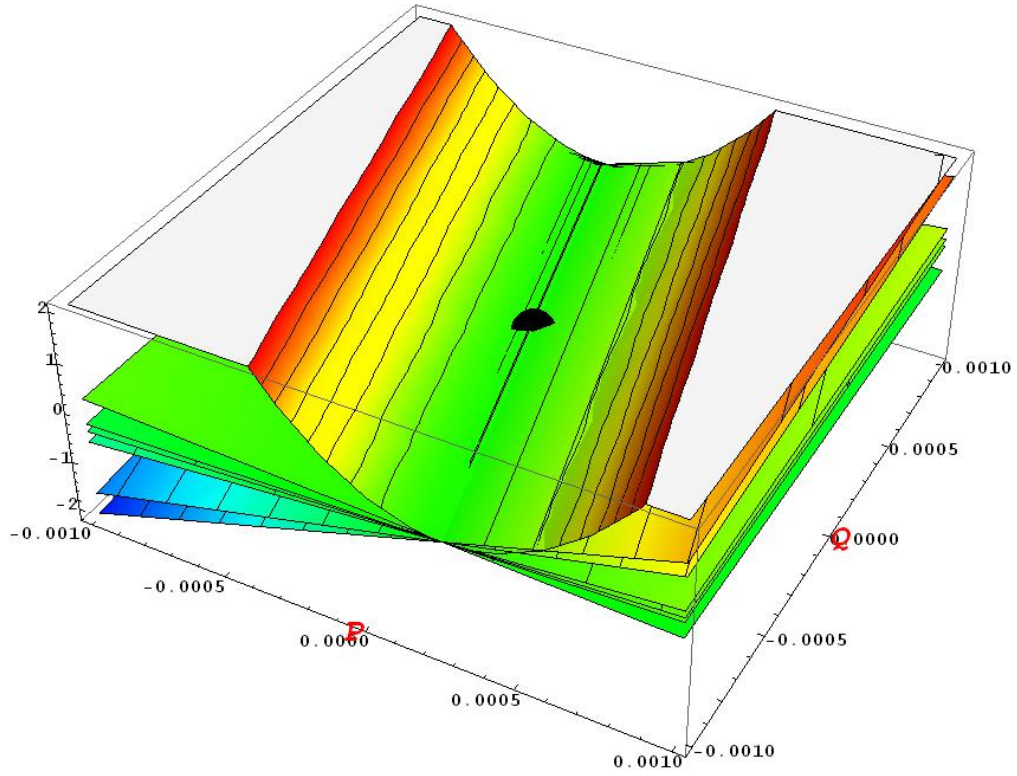


Figure 3.26.: 3D plot for single point, two-variable, minimization; for variables P and Q, for image point $\mathbf{p}_{11,136}$.

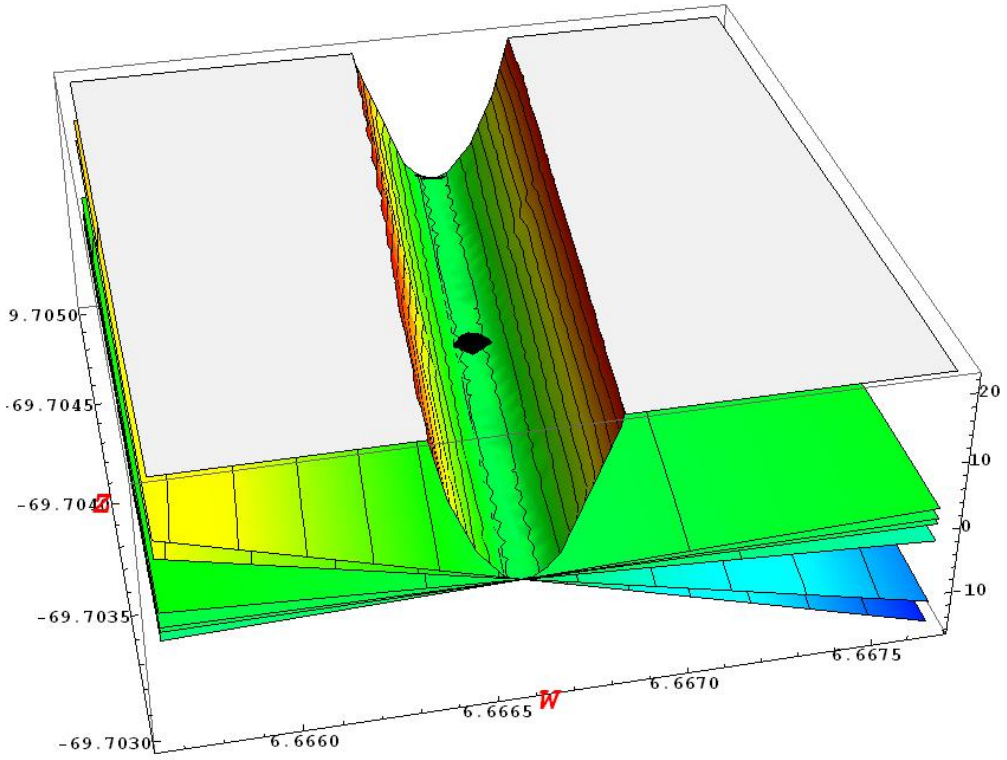


Figure 3.27.: 3D plot for single point, two-variable, minimization; for variables Z and W, for image point $\mathbf{p}_{11,136}$.

An important result is witnesses here. For certain situations, with two or perhaps several variables, the constraint equations form planes which intersect in one line. This corresponds to an unfortunate effect for the squared 2-norm surface. As can be seen, the surface forms one large folded "piece of paper". The "fold" is the infinite line which intersect with the correct solution. The initial point used in the minimization thus becomes very important. It was set to *zero* in all minimization attempts.

For variables P and Q, which both equals *zero* in this example, the minimization does converge at exactly this minimum, which is no surprise since the initial point is zero for both variables.

Figure 3.27 show another 3D plot for variables Z and W, the only two *non-zero* variables in the current example. This plot also forms a global *fold* on an infinite line with the correct solution. As can be seen in Table B.1, local search techniques were not successful in solving correctly for an Z and W. Tables B.2 and B.3 show that *global* search techniques were also not successful. This is probably due to the fact the initial point for local search, is far from the correct solution.

This subsection took a look at single point minimization. Important results were pre-

sented. Especially the importance of the initial point of local minimization. Further minimization results are shown in Appendix B. The *bold* values in the tables, correspond to values which are close to the true values (within an error of $\pm 10^{-2}$).

3.4.9. Multiple Point Minimization

This subsection will briefly show some numerical results and some 3D graphical plots created from a few examples of multiple point minimization attempts, using all 4 image points.

When minimizing using more than one image point the choice of variables is different than for one point minimization. The question is not only limited to *what* variables to solve for. One must also consider the expected *surface characteristics* solved, for in each image points.

As an example, consider estimating surface orientation and depth for a rigid translating non-rotating non-frontal plane surface. This would certainly include the variables P, Q, and Z. The other tree variables, U, V, and W would be considered known. The two surface gradient variables, P and Q, can now both be considered *surface global*; i.e. they are the same for all 4 image points. In such a case, only the variables P and Q would be used in the constraint system. On the other hand, the scene depth is *surface local*; i.e. it is different for each image point. The constraint system thus also contains the variables $Z1$, $Z2$, $Z3$, and $Z4$, each corresponding to one of the 4 image points.

For different non-planar surfaces, P and Q are *surface local* parameters. The constraint system will thus contain the variables $P1$, $P2$, $P3$, $P4$, $Q1$, $Q2$, $Q3$, and $Q4$. Including the variables for depth.

Do to the inherent complexity in choosing for variables and their surface characteristics, some simplifications will be necessary. Tables B.7 to B.9 show the results from the 63 combinations of the multiple point minimization, when considering, in each case, all six variables to be surface *global*. These are the *only* multiple point minimization results which involve the variables for velocity, i.e. U, V, W.

The remaining results for multiple point minimization will only involve variables P, Q, and Z. Two further minimization examples are shown: (1) with P and Q as surface *global* and Z as surface *local*, and (2) with all three variables as surface *local*.

Tables 3.9 and 3.9 show the mentioned minimization results. The minimization is correct for all the variables.

The remaining parts of this subsection will present a few 3D plots which should give a better understanding of what actually happens when more image points are used to build the the constraint system.

P	Q	Z1-Z4		Res
7.7704×10^{-8}	-1.7460×10^{-8}	Z1	-6.9704×10^{1}	7.5718×10^{-7}
		Z2	-6.9704×10^{1}	
		Z3	-6.9704×10^{1}	
		Z4	-6.9704×10^{1}	

Table 3.9.: Numerical results of multiple point minimization using all four image points (P and Q as surface *global* and Z as surface *local*).

P1-P4		Q1-Q4		Z1-Z4		Res
P1	-1.9194×10^{-7}	Q1	4.7984×10^{-8}	Z1	-6.9704×10^{1}	7.3374×10^{-7}
P2	-2.2395×10^{-10}	Q2	2.0155×10^{-9}	Z2	-6.9704×10^{1}	
P3	0.	Q3	0.	Z3	-6.9704×10^{1}	
P4	6.4154×10^{-8}	Q4	-8.2483×10^{-8}	Z4	-6.9704×10^{1}	

Table 3.10.: Numerical results of multiple point minimization using all four image points (all three variables as surface *local*).

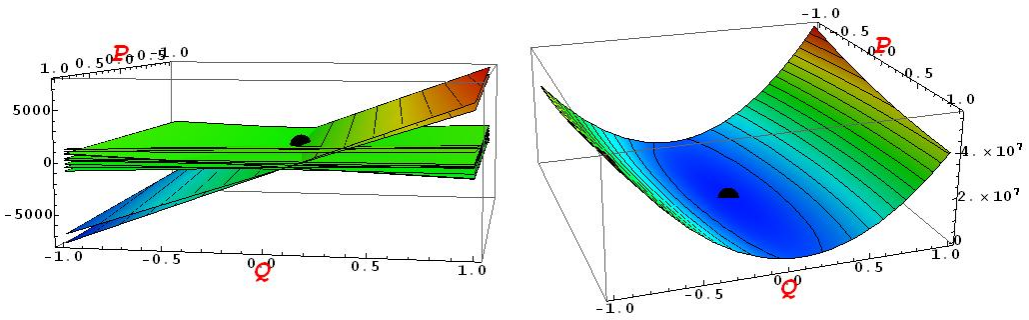


Figure 3.28.: Multiple point minimization; all four image points; for P and Q as surface global variables (see text).

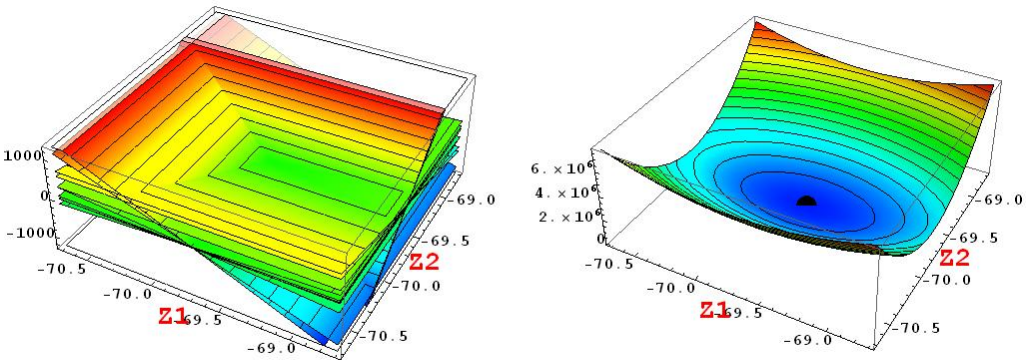


Figure 3.29.: Multiple point minimization; only two of the four image points; for $Z1$ and $Z2$ as surface local variables (see text).

Figure 3.28 is a 3D plot for a special minimization attempt with only variables: P and Q, but using all four image points. The graphic to the left show how 24 planes intersect in a single point. This point is the correct value for P and Q, namely $(0,0)$. The graphic to right shows the squared 2-norm surface which seems to have a global minimum.

Thus it seems, that by adding several image points, the constraint system becomes easier to minimize. Global minimums seems to form – which is quite positive.

Figure 3.29 show a 3D plot for a different minimization attempt. Here, only two image points are used: (1) $\mathbf{p}_{11,136}$ and (2) $\mathbf{p}_{11,209}$) and we only solve for $Z1$ and $Z2$ (surface local minimization). Again a global minimum seems to form, even with surface local minimization using only image points. Appendix B shows further results for multiple point minimization, using the numerical context of this example.

This chapter turned out to be a longer chapter, including details from the simulation itself, the data flow, estimation of partial derivatives, and a full numerical example with some some important results.

The next chapter will introduce the methods applied and the results achieved in the camera experiments. That chapter is not nearly as long as this one, due to the fact that the computational experiment did receive greater attention throughout the project period. The majority of the time was put into designing, testing, and debugging the computational experiments.

The time was well spent, it seems. It actually is possible to solve for 3D structure in many circumstances – especially if multiple point minimization is used. Naturally the next step is the camera experiments and the broader application focus that they have.

4. Camera Experiments

This chapter is concerned with the camera experiments. It takes a first look at the experimental setup in Section 4.1. Section 4.2 explains how textures were synthesized and analyzed. Especially how image texture density was attempted estimated in image sequences of a textured surface. Certain difficulties with segmentation of texels in real camera images did turn to be quite problematic for the end goal.

By the time of writing, no estimates of 3D surface structure has been achieved in the camera experiments. The problems faced in the segmentation part, turned out to be detrimental to the final goal and the time schedule. Section 4.3 will have final comments on this.

4.1. The Experimental Setup

The camera experiments are much different than the computational experiments. They involve a physical experimental setup, a real digital camera, and a real textured (plane) surface.

When considering that the end goal was not achieved, this chapter will direct its focus on the image processing methods applied and the problems faced. Before that, this section will take a look at the experimental setup and briefly mention some possible sources for error.

A brief account on the building process will be given in Subsection 4.1.1. Later, Subsection 4.1.2 presents an overview of the geometry and the different parts of the setup.

4.1.1. The Building Process and Likely Sources of Error

The building process involved different kinds of considerations and methods. Over a time span of several months the experimental setup was designed, built, measured, and calibrated. And calibrated again. Applying knowledge and material from the mechanics engineering department; tools such as rulers, calipers, tape measure, etc.; the setup slowly its took form.

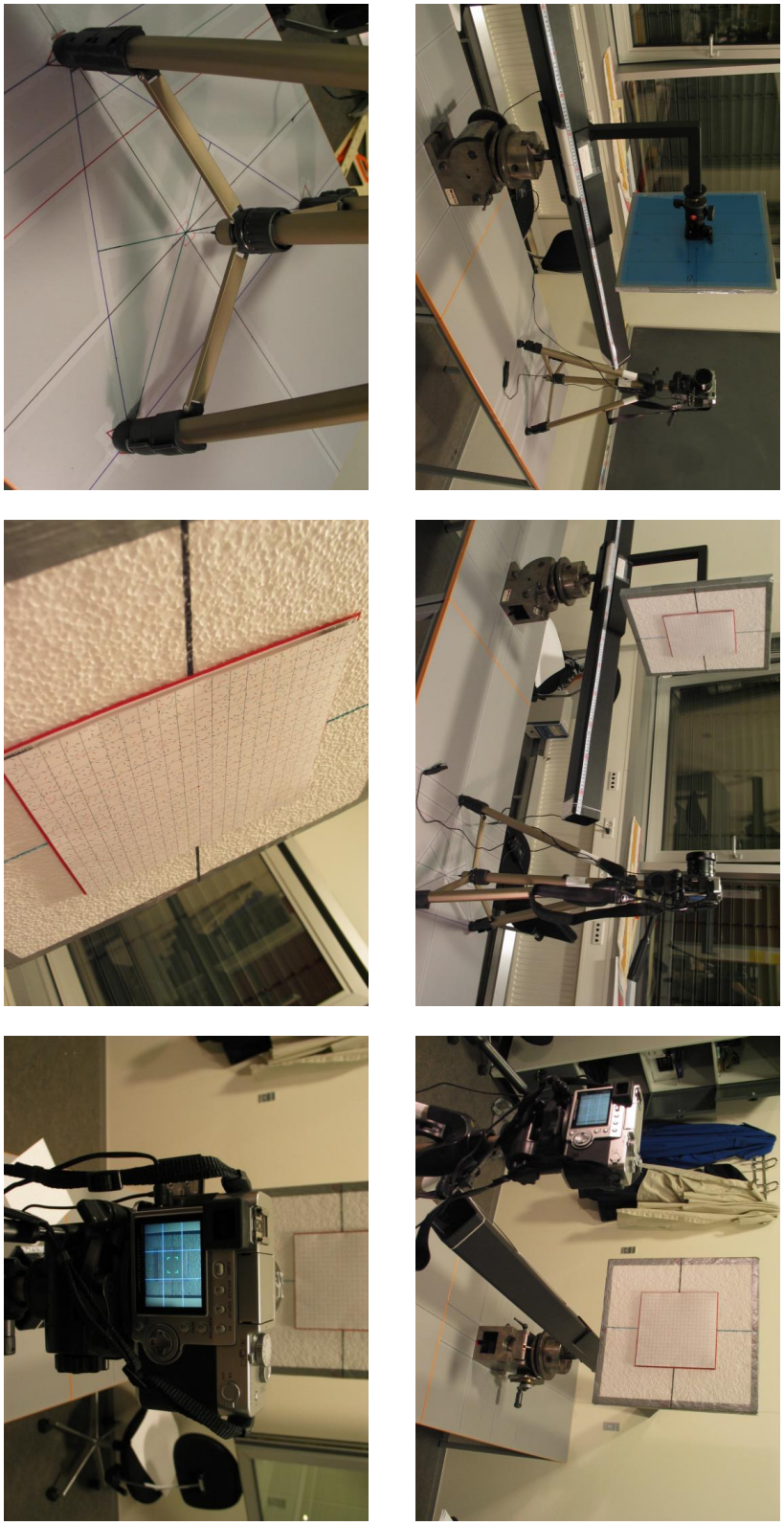


Figure 4.1.: Different views of the experimental setup built for the camera experiments.

The photographs in Figure 4.1 show the experimental setup from different views. As can be probably be gleaned from these images, much of the work in building the setup consisted of taking measurements. Lots of measurements. Almost all of these measurements were made with precisions down to a scale of 500 μm.

The manual caliper used, was equipped with a *vernier* scale which allowed measurements to be made down to a scale of 10 μm. When possible, such precision was attempted.

At the time, when the design and preliminary assembly stage was set to begin, many of the results from computational experiments had already been achieved. The path was clear to continue with the camera experiments. The process began with designs consisting of crude hand-drawn schematics. Communications with the mechanics engineering department (*MED*) was initiated and raw material was selected.

It was at this point that some mistakes were made. The precision requirements were simply not specific enough. This resulted in the use, by the MED, of assembly methods which made later calibrations very difficult. For instance, welding was used. The heating will inevitably change the topology of the metal surfaces enjoined, especially at scales of ± 10 μm.

Also, the choice of some of the material were later determined to be a possible source of calibration error. The iron metal-frame, used as a *motion-track*, was produced primarily for industrial use, i.e. with much lower precision requirements than needed in computer vision experimental setups. The experimental setup was attempted calibrated, with these sources of errors in mind.

The camera experiments also implied the existence of actual textured surfaces. From the beginning the surface was simplified to a plane. A plastic plate of approximate dimensions 40 mm $\times$ 40 mm, considered to be acceptably planar, was chosen as the corresponding surface.

Since surface orientation is especially important to the primary method, a way of adjusting the orientation of the plastic plate was needed. A camera *ball-head* became the method of choice. The ball-head, originally intended to calibrate the orientation of cameras, was believed to meet the precision requirements.

The calibration of the camera ball-head showed to be very difficult due to its freedom of movement. Calibration was attempted, with the plastic plate removed, by attaching a small class 2 battery-driven laser pointer on the ball-head, having the laser beam acting as the "normal vector" to the plate. A preliminary idea was to create a small subset of the surface gradient space (P,Q) on a wall adjacent to the setup. When the laser beam was pointed to a coordinate in this subspace, the orientation could be considered calibrated.

It was later determined that the laser pointer itself was impossible to calibrate. It was not professional equipment and the source of error was probably similar to that of the iron metal frame. Having exhausted both options and time, it was at this stage

decided that the camera experiments would only show results achieved with a frontal plane, moving toward the camera in a straight line following the Z-axis.

Different kinds of textures were synthesized in Mathematica. By using its symbolic graphical programming languages, this showed to be almost unbelievably easy. For simplicity reasons, it was decided that the textures had to consist of independent texels, easily separable. Two types of textures were created: (1) A texture consisting of small colored circles, and (2) a texture consisting of short colored line segments of various rotations. Both were printed on standard white cardboard paper with unknown specularity.

Eventually the first type of texture, the one with circles, was discarded. It was found difficult to separate texels by color only. The idea of making a texture consisting of line segments was based on the hypothesis that, by combining color and line orientation, the texel separation and counting would be easier.

This hypothesis showed to be very true on a synthetic image with colored lines. But, as later sections will describe, this form of texture eventually turned to suffer from the same problems as the circle-textures: difficulties with color separation.

A *single* experiment, using a *single* line-segment texture, and a *single* image sequence was carried out. The numerical context used in this experiment, is identical to the one defined and used in the numerical example in Section 3.4.

Section 4.2 and 4.3 will both have more to say about the methods in this single experiment and the results achieved.

4.1.2. Geometry and Parts

Figure 4.2 gives an overview of the experimental setup and its parts. The setup is fully manual; everything is moved by hand. The *texture-plate* is attached on a camera ball head, which again is attached to an L-shaped movable metal arm – the *surface-arm*. The surface-arm is fully detachable and slides on an iron-frame –the *motion-track*.

The motion track acts at the line-of-travel for the surface. In the single experiment, this line-of-travel is parallel with the line-of-sight in the camera. The motion-track rest on a large metal block – the *center-block*. When originally designed, the intent was to have a motion-track with two freedoms of movement: *yaw* and *pitch*. The idea was to have the motion track centered on some form of pivoting mechanism which allowed these two forms forms of rotation.

An existing mechanism, originally intended for other precision work, was proposed by the MED to shorten the time spent on construction. This mechanism was used as the center-block. Using this quite heavy mechanism, a yaw rotation can be applied to the motion track by turning a handle. On the other hand, the effort required to change this mechanism into different positions of pitch-rotation, did eventually turn out to be

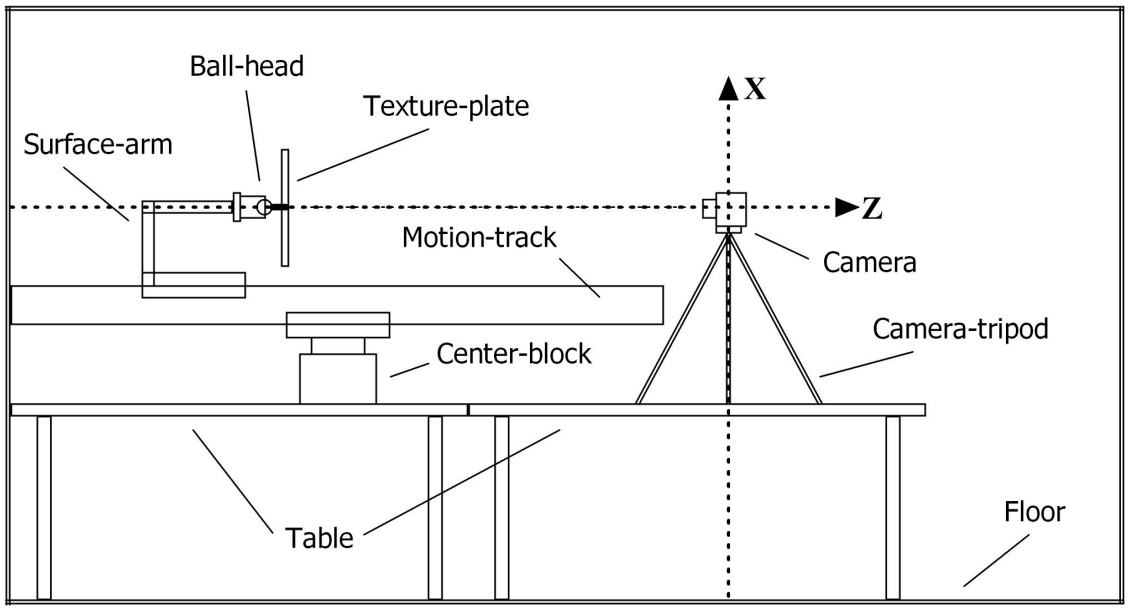


Figure 4.2.: A diagram which should give an overview of the geometry and different parts of the experimental setup. As expected the camera is at the origin. The focal length and the image plane is somewhere close to the origin, naturally inside the camera. It was found impossible to calibrate for concept of a camera origin and a focal length distance to an image plane. The camera was simply calibrated to be as close to the origin as possible (see text).

be counter productive. Great force needs to applied using a hammer or similar tool. Thus only yaw rotation is possible. A yaw rotation corresponds to $V \neq 0$.

The camera, a *Leica Digilux 2*, is mounted on a standard camera-tripod. This tripod rest on a table surface. The tripod is not fastened to this table surface, which makes possible final adjustments before imaging begins. The tables are standard office tables with wooden surfaces.

Using this setup, is is possible to create image sequences of a moving plane textured surface. This is done by attaching a texture to the texture-plate (textures can be attached and removed using velcro). The motion-track is adjusted to the wanted line-of-travel. In the current experiment, imaging was done manually by first moving the surface-arm 5 mm and then taking and image; moving and taking an image; repeated 21 times until 10 cm had been covered on the motion track.

The digital Leica camera used, has many different options for controlling the imaging. The ISO setting used was *ISO100*. Settings for shutter speed and aperture were controlled automatic by the camera (known as "*programmed automatic exposure mode*"). Auto focus was used and the *exposure metering method* was set to *multi-field metering*. Different settings for *white balance* was used and tested for. Eventually the manual setting for white balance was used. Standard white paper was used for manual calibration.

4.2. Texture Synthesis and Analysis

This section will present the methods used for texture synthesis and analysis. Subsection 4.2.1 takes a brief look at the texture and the methods used to create them. The analysis covers (1) texel segmentation and (2) texel counting. Methods and problem related to texel segmentation is presented in Subsection 4.2.2. Subsection 4.2.3 does the same for texel counting.

4.2.1. Texture Synthesis

It was determined that the best textures would be simple textures consisting of separable texel-groups divided into separable groups or subgroups of texels. Each of these groups thus corresponds to the different surface texture densities.

A *texel-group* is simply a subset of all the texels in one texture. The texels of a single texel-group will need to have some form of similarity in either color or shape. Two texels belonging to two different groups will need to have some form of dissimilarity in either color or shape. texel-groups can have subgroups of texels. The same similarity/dissimilarity rules applies recursively. No texel covers part of another texel. A minimum distance between all texels was enforced in the synthesis.

It was decided that a texture would need to have at least 6 separable groups with texels. This number was inherited directly from the computational experiment where $h = 6$ *surface* texture density functions are used to numerically model the texture density on the surface.

Two textures were created:

1. A *circle-texture* with 6 *texel-groups*. Each texel-group contains 300 circles with a diameter of 1 mm. There are a total of 6 texel-groups; each with circles of different color. These 6 colors are: *red*, *green*, *blue*, *cyan*, *magenta*, and *yellow*. No subgroups are defined. A global minimum distance was set to 0.5 mm.

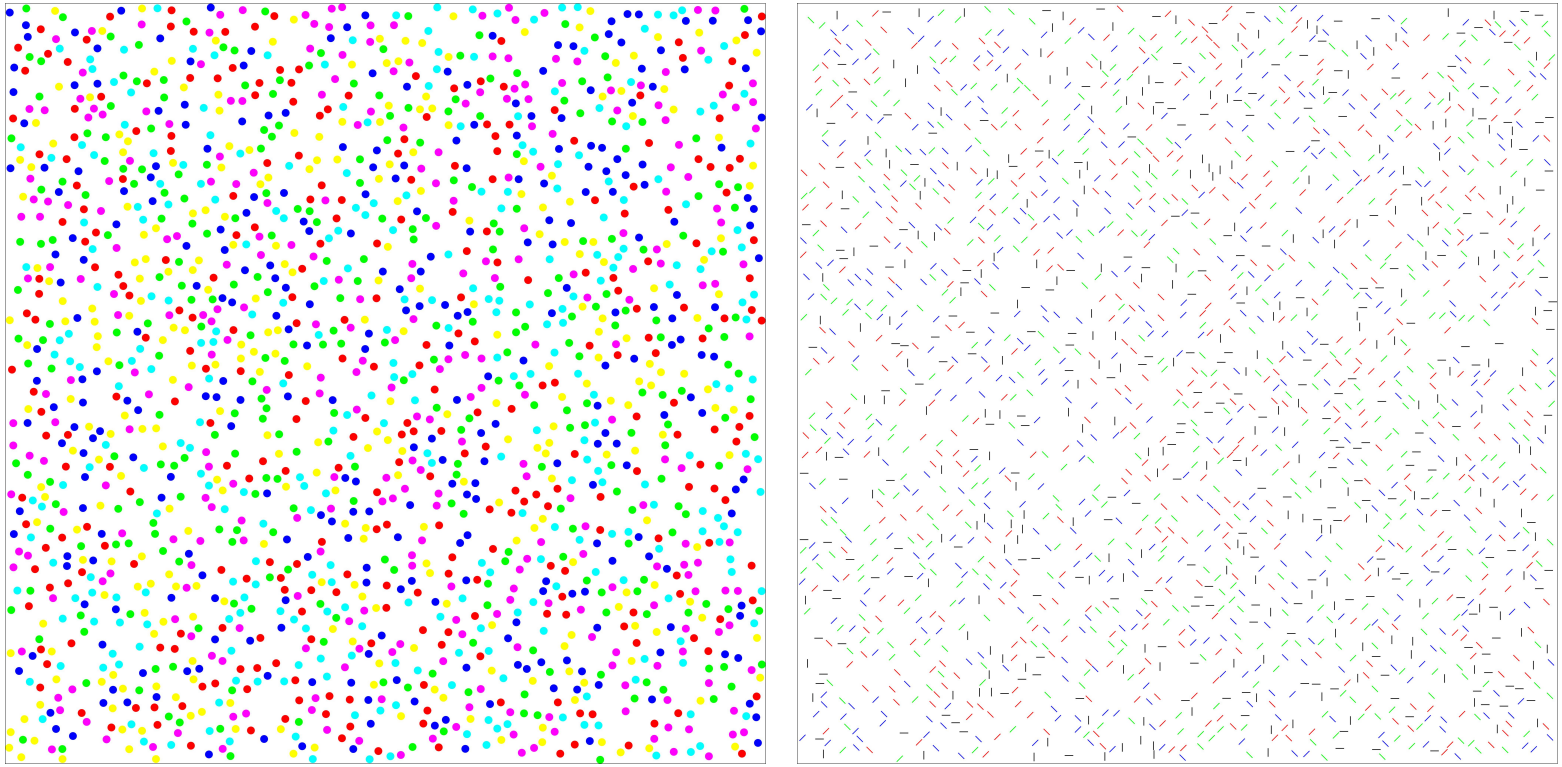
2. A *line-texture* with 4 *texel-groups*, where each group consists of yet again 2 subgroups. Each texel-group contains 500 line segments with a length of 2 mm. There are a total of 4 texel-groups; each with line segments of different color. These 4 colors are: *black*, *red*, *green*, and *blue*. The 2 subgroups with the color black, each has 250 line segments with an orientation of (1) $-90°$ and (2) $0°$. The 2 subgroups of the 3 other texel-groups has orientation (1) $-45°$ and (2) $+45°$. A global minimum distance was set to 0.25 mm (see text below).

The definitions above requires further elaboration. Both textures were synthesized such that all texels would fit on a piece of 19.5 cm $\times$ 19.5 cm cardboard paper. Figure 4.3 shows two synthetic prints of the two textures. These prints are similar to the cardboard-prints used in the experiments.

The circles in the circle-texture were placed by sampling the coordinates of the circle centers using a *pseudo-random* number generator (SRG) (the *mersenne twister* was used). The SRG would generate a center coordinate within coordinate limits. The coordinate generated was hereafter checked for collision with other already placed circles, taking in considerations such as radius and minimum distance.

A "bad-candidate" is a generated center-coordinate which collides with an existing circle. The probability for this to happen is naturally higher in the later stages of the synthesis, were more circles had been successfully placed. A bad-candidate *limit* of 10000 was used. If this limit was reached, the circle placement was aborted. This only happened when too many circles, too large a radius, and/or to large a minimum distance was used. And since pseudo-random number generation was used, a second run sometimes turned out to be successful for all circles.

The line-texture was synthesized using the same techniques. A circle-texture is also created. But instead of colored circles, line segments are places such that their center point is a circle center and their length is equal to the circle diameter.



Texture with circles

Texture with line-segments

Figure 4.3.: Synthetic versions of the two types of textures used. Images were created using the original digital raster files used to print the textures on cardboard paper.

4.2.2. Texel Segmentation

The texel segmentation is crucial for later stages that *counts* actual textures using connected-component analysis. This concept was visited briefly already in Chapter 1 where an example of synthetic density in an image was given (Figure 1.8 on page 1.8).

The segmentation in the camera experiments was attempted by using simple non-uniform color quantization. MATLAB has facilities which converts an *RGB-image* to an *indexed-image* with *colormap*. The colormap contains only the colors wanted from the original RGB-image. Dithering was *not* used.

A subset of RGB colors are used. For instance, to create the circle-texture, six colors were used: *red*, *green*, *blue*, *cyan*, *magenta*, and *yellow*. These are the quantization-colors for the circle-texture. For the line-texture there are: *black*, *red*, *green*, and *blue*.

Non-uniform color quantization works by partitioning the RGB-space into non-uniform volumes with the quantization colors as the center colors. The size of the volumes is such that all original colors in the original RGB-image, are mapped to a quantization color.

This method appeared at first to be quite a strong formalism for segmentation based on color only, and it obviously worked on the synthetic versions of both textures (the original raster graphics). But it did eventually turn out to be quite sensitive to the lighting used when the image sequence was created.

The experiments were created in a room with quite a complex lighting situation. Large glass-openings exists which gives passage to the hall-lighting outside the room. And the room itself has low-energy lighting in the form of a single halogen-lamp in the ceiling. Darkening of the room was very early in the project deemed unnecessary. This was a certain mistake since quite complex lightning situations do occur.

It now seems as a good idea to a look at the actual images created using these textures. Figure 4.4 show image number 11 in an image sequence created using the circle texture. The white balance in the Leica camera was set to an automatic mode best suited for *halogen* lighting. An extra light source was also used for this particular sequence (a lamp with a standard 60 W incandescent light bulb). This light-source was tilted down onto the texture plate, from above the camera. This created a complex lighting situation, with two lighting sources. Also, this complex lighting situation did not correspond to the white balance chosen in the camera.

To the left in Figure 4.4 the full image is seen (original dimensions are 2560×1920 pixels). The image to the right shows a close-up of the textured area of the texture-plate. The complex lighting situations is apparent in the close-up when looking at the yellow circles. The ones at the top of the texture seems to be almost specular. At least the color saturation seems stronger than in the lower parts of the texture.

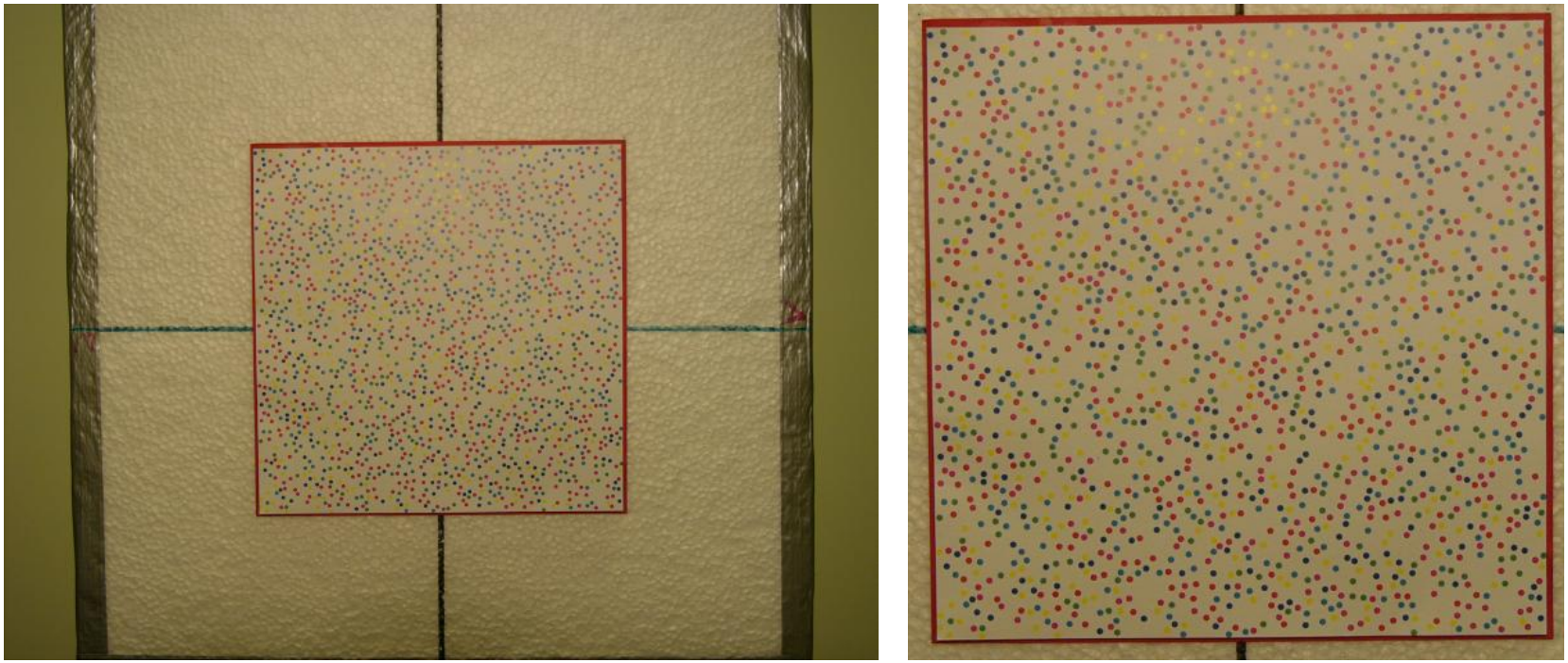


Image 11 from sequence

Textured area of texture-plate

Figure 4.4.: Actual photographs from the image sequence created using the circle texture.

This effect is combination of the unfortunate placement of the incandescent light source and the fact that the coloring substance applied to the cardboard paper by the color laser printer used, seems to have a specular surface when the yellow hue is produced.

Non-uniform color quantization using the 6 colors mentioned for the circle-texture (combined with *white* for the background) was applied to the textured part of the sequence-image shown in Figure 4.4.

Figure 4.5 shows the quantization results. The left image show a close-up of the synthetic version of the line-texture. The middle image shows a close-up of a preprocessed[1] version of sequence-image 11. The image to the right shows the quantized version of the same area. These image give a good example on the status of the segmentation. The magenta circles are mostly read, and the cyan circles contains blue pixels. This is certainly a step in the right direction, but further work is needed to segment the circles in circle-texture.

These inconsistencies with colors turned out to be quite problematic for later steps involving the counting of texels. The actual number of texels in a certain image patch dI should be a cue leading to surface orientation *alone*. But this number is apparently *not* invariant to lighting conditions. The fact that light sources has an effect on the *image* texture density will most certainly be a problem which cannot be easily ignored in later work.

Figure 4.6 shows sequence-image 11 from a sequence crated using the line-texture. The sequence was created under more simple lighting conditions. Only the halogen lighting source in the ceiling was used and the white balance was calibrated manually using the surface of white paper as *reference-white*. The hue of the entire image seems *cooler* (more blue) than for the sequence-image for the circle-texture in Figure 4.4.

Again we turn to non-uniform color quantization. Figure 4.7 show the results of attempting to quantize a preprocessed version of the sequence-image shown in Figure 4.6. The colors used were *black*, *red*, *green*, and *blue* (and *white* for the background).

Again the left image show a close-up of the synthetic line-texture, middle image a close-up of the sequence-image shown in Figure 4.6, and the right image show the quantized version, resulting from using the 5 colors mentioned earlier.

Comparing the synthetic to the sequence-image (left compared to middle) it is evident that the color conditions have changed completely. The segmentation is evidently even worse than with the circle texture.

These examples have shown that segmentation based on color only, is perhaps not as strong an invariant as first assumed. At least not when the experimental lighting

[1] A contrast enhancement and a standard median filter with a neighborhood of size 3×3 was applied to all sequence-images.

Figure 4.5.: Examples on segmentation of colored circles using non-uniform color quantization.

Textured area of texture-plate

Image 11 from sequence

Figure 4.6.: Actual photographs from the image sequence created using the line-segment texture.

A close-up of the synthetic version

A close-up of the original image

The color-quantized version

Figure 4.7.: Examples on segmentation of colored line-segments using non-uniform color quantization.

conditions could not be controlled any further. In later experiments the room used for the experiments will need be darkened and forms of controlled lighting will need to be applied.

4.2.3. Texel Counting

Since the actual texel segmentation showed to be unsuccessful with real camera images no image texture density estimates have been produced, and thus no minimization attempts can occur.

In the event that texel segmentation should become successful, the addition of the methods of this subsection should make true image texture density estimates possible.

This subsection will present the methods used and results achieved when seeking to estimate texture density using the *synthetic* version of the line-texture.

The texel *cardinality* measures were done using *hough transform* analysis for finding lines in images. The midpoint of these lines is used as an indicator for a *texel*. These texel indicators are counted using a fixed and *distinct* neighborhood operation.

Remember that the line-texture consists of a total of 8 subgroup, each containing line-segments with different color and different orientation. Since the target image is fully synthetic, the color quantization works with no problem; it is able to correctly segment the image into binary images containing *black*, *red*, *green*, and *blue* line-segments.

The remaining part of this subsection will deal only with the red lines. This image still contains two subgroups of texels: (1) red lines with a -45° orientation and (2) red lines with a $+45^\circ$ orientation. The goal is yet again to segment the binary image with red lines into two images containing lines with the different orientation.

These two images are further processed such that only the midpoint of each line is a foreground pixel (a pixel with the value 1 in a binary image). These two images are used for the texel cardinality estimation, using the midpoint of the lines as a texel *indicator*.

The hough transform maps foreground pixels in a binary image to curves in a (θ, ρ)-space or *hough*-space. This space is usually referred to as the *hough accumulator* – especially when a resolution is applied to the hough-space. The resolution for θ and ρ where both set to increments of 0.25.

The hough transform can be used to find collinear points in an image. The goal here is to find all the points in the image, which forms line-segments with orientations $\pm 45^\circ$. To do so, the hough accumulator is filtered such that it only contains the *theta*-bands at or close to $\pm 45^\circ$. Every other accumulator cell is set to *zero*.

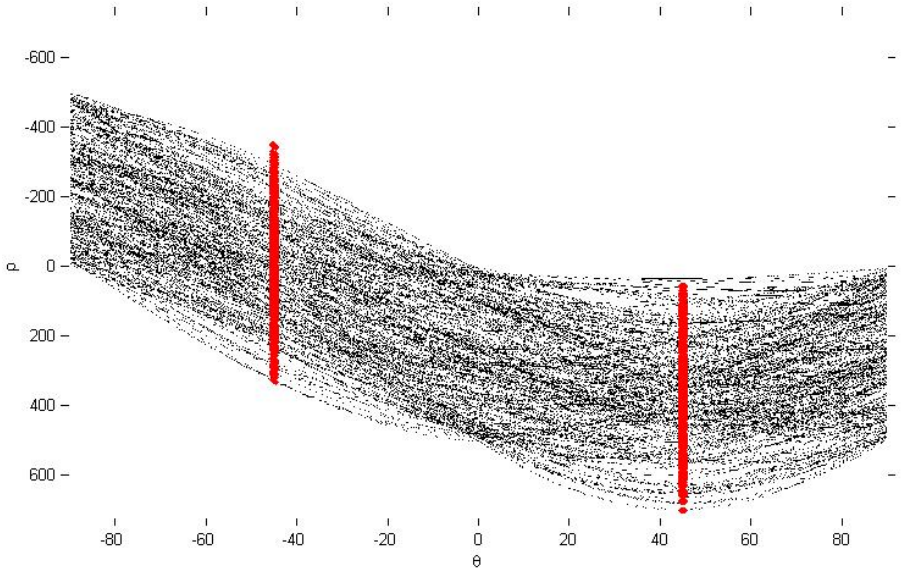


Figure 4.8.: An example of a filtered hough accumulator. The marked *theta*-bands represents the subspace used to find the lines with orientation $-45°$ and $+45°$. Note that the full hough accumulator is shown here.

Figure 4.8 shows an example of a filtered hough accumulator. The full hough accumulator is shown. The marked bands at $\pm 45°$ indicates the only preserved cells in the full hough accumulator.

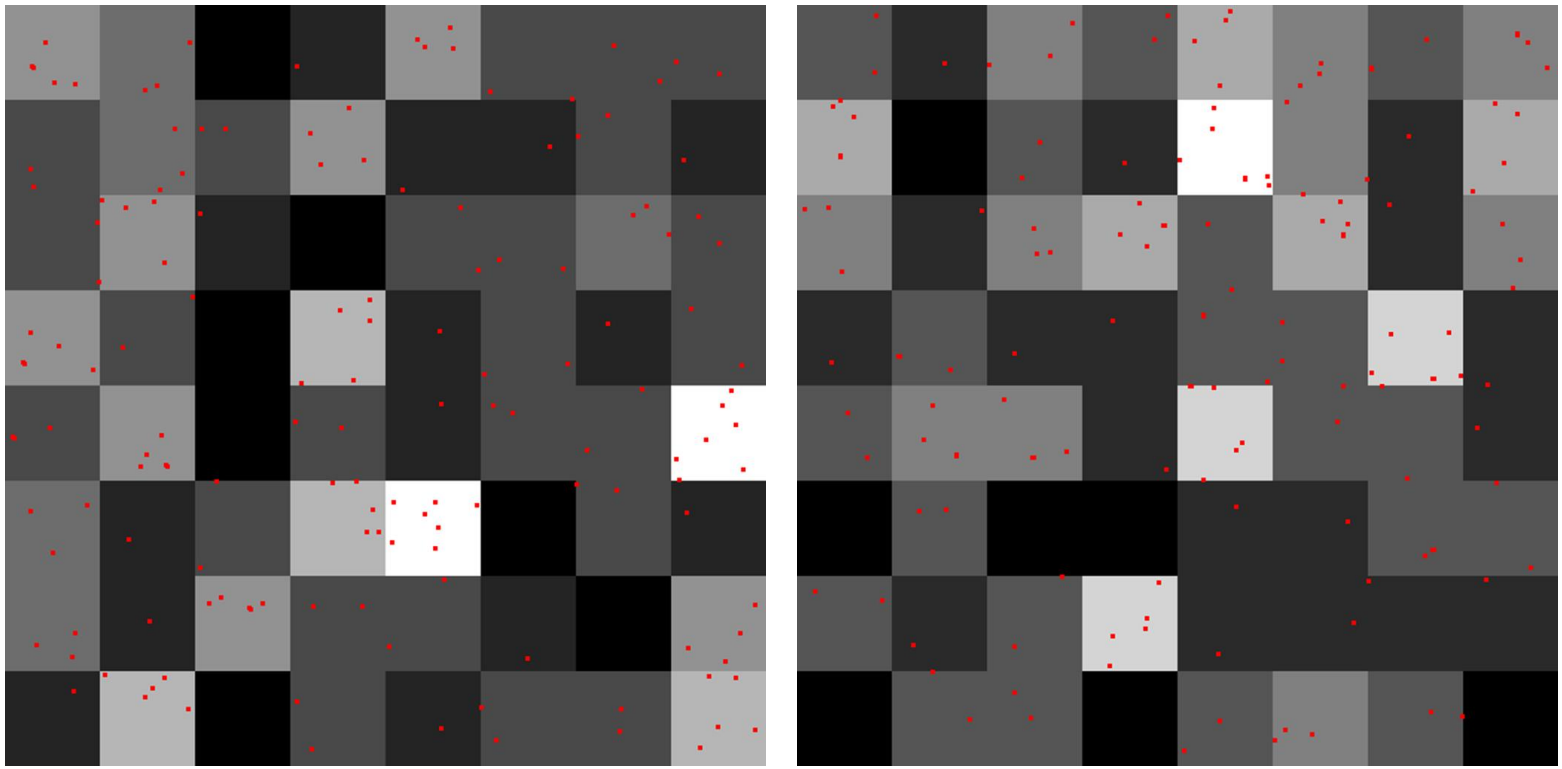
All the cells preserved are used to make an inverse hough transform, to find the set of (x, y) coordinates in the image, which corresponds to points forming lines with orientation $\pm 45°$.

Figure 4.9 shows the results achieved with this orientation-based segmentation using a filtered hough transform. The results seems promising. In the left image, all the lines with a $-45°$ orientation have been marked with a thick red dot. The same happens in the left image for lines with a $+45°$ orientation.

These thick red dots are the texel indicators; the midpoint of each line-segment. The final step is now to apply a distinct neighborhood analysis on the two images containing the texel indicators. In each neighborhood a connected component analysis capable of counting groups of *connected* pixels was applied (8-connectivity was used).

The results are seen in Figure 4.10 using a neighborhood of size 64×64 pixels. The images show the density plots for the texel group containing red line-segments (compare with the left image in Figure 4.3). The set of texel indicators have been superimposed onto the density plots. In the gray scale density plots, black is a minimum count, and white is a maximum count.

Figure 4.9.: Examples on *synthetic* sub-group segmentation of *red* line-segments based on orientation using a *filtered* hough accumulator.



−45° orientation +45° orientation

Figure 4.10.: Density plots of the sub-groups red lines (synthetic version). The markings of each rotation is superimposed as dots on each plot. A neighborhood of size 64 × 64 pixels was used (black is a minimum count, and white is a maximum count).

4.3. Results and Conclusive Remarks

The camera experiments have *not* been able to show results for 3D structure estimation. This was primarily due segmentation difficulties and problems with light conditions in the experimental environment.

The following series of problems were observed in the process:

- Calibration of the experimental setup showed to be extremely difficult. The assembly methods applied and material used has inevitably introduced errors into the system. But, due to the segmentation difficulties, is has not been possible to test for these error by seeking to minimize a constraint system. It is possible that, when actual texture density can be measured, these error will still have a detrimental effect on the goal at hand.

- The experimental environment was probably unsuitable for these kinds of experiments. The room used has several glass openings which easily introduces other light-sources; for instance hall-lighting. This light could apparently not be switched off manually. Darkening of the room was deemed unnecessary due to the belief that texture analysis is *insensitive* to light conditions in general. This proved to be wrong.

- Light conditions are very important indeed. Of course this is obvious when color is used as a dissimilarity measure for segmentation of texels. It is plausible that, with further work, the texel segmentation could be based purely on texel shape instead of color.

The following results were achieved:

- The hough transform was successfully applied to a synthetic (binary) texture containing line segments with different orientations. By filtering the hough transform, the foreground pixels, corresponding to line segments close to a particular orientation, could be identified. Using these image points, a list of actual line segments; including the two endpoints, θ, and ρ; could be generated.

- By searching through this list of line segments, the correct line segments are identified, the midpoints is found, and texel indicators are thus available.

- The texel indicators can be used directly for image texture density estimations, using a distinct neighborhood operation and 8-connected component analysis.

The problems faced and the mistakes made throughout the experiments are certainly also results not easily ignored. There is much to learn here. Especially, that for color based texel segmentation to work, the lighting conditions must be calibrated as much as possible.

One final remark is related to what one expects to know about the system. In these experiments, the nature of the texture was always expected to be known. The algorithms applied, use special knowledge about the texture (does it have lines, does it have circles, what colors to expect).

This is important when seeking to generalize the problem to arbitrary textures. This was evidently never the goal with this work. If this is attempted, then certainly more advanced measures of texture density estimation needs to be developed. Also, it is not known, if the concept of texture density can be usefully generalized to all surfaces, which we would, under most circumstances, still refer to as textured.

This chapter took a brief look at some of the work applied in the camera experiments. Results with 3D structure estimation would certainly have been positive. But the problems faced in the camera experiments only makes the final goal inconclusive – not impossible.

5. Summary and Conclusions

This chapter is a summary of this report. Section 5.1 presents all the concrete results. Section 5.2 takes a look at the possibilities for future work and concludes this report with some final remarks. A short summary is also present in the *abstract* in the beginning of the report on page v.

5.1. Concrete Results

This section will present a list of *concrete results* achieved in this work. These results will be presented in two brief subsections: Subsection 5.1.1 will present the results achieved in the computational experiments. Subsection 5.1.2 will do the same for the result achieved in the camera experiments.

5.1.1. Results – Computational Experiments

In general the work carried out in the computational experiments, proves (computationally) that the formalisms behind the primary methods are correct.

This is apparently the first work which have shown, computationally, that the concept of texture flow can be useful for direct estimation/extraction of 3D structure.

- Synthetic constraint systems can be built and solved using both local and global minimization techniques.
- The solvability of the constraint system is certainly sensitive to the number of variables, the number of image points, and the surface characteristics expected for each image point.
- Using a local search with an initial points set to *zero* in all unknowns, did show to be counter productive for some situations, especially when only a *single* image point is used to create the constraint system.
- Using two or more image points, the constraint system seems to become much easier to minimize. A global minimum seems to occur at the correct (synthetic) values.

- When considering all variables to be *surface local*, the number of unknowns are increased and the solvability will in general decrease. Adding more image points to the constraint system seems to yet again increase the solvability.

5.1.2. Results – Camera Experiments

The camera experiments faced certain problems with the experimental setup and the methods chosen for texel segmentation was very sensitive to the lighting conditions and the general camera setup.

The main goal of estimating 3D structure in an image sequence created with a digital camera was not reached. The results are currently inconclusive.

- A result was achieved in the sense that a lesson was learned about the sensitivity of texture analysis to the lighting conditions. The sensitivity is high.
- A possible method for estimating image texture density in textures containing line segments was developed. Results were shown to be successful on synthetic texture.

Later parts of Chapter 4 has further conclusions on the results achieved and the problems faced.

5.2. Future Work and Final Reflections

Future work on *the primary method* for direct estimation of 3D surface shape and velocity using texture flow is certainly possible. This is perhaps a very large understatement. This work opens up for a new and large research potential in all areas related to monocular 3D surface reconstruction, using texture density cues from image sequences of textured surfaces.

5.2.1. Limitations of the Primary Method

It seems plausible, that the primary methods, with its reliance on texture density, is a strong method for direct estimation of 3D structure. But it is not necessarily a strength that the method relies so heavily on a texture feature. Texture is a very complex image feature, and texture density is certainly no less complex. It is plausible that the primary method is only applicable for situations where the surfaces has *known* texture.

Since the method relies on the concept of texture density and the possibilities for estimating it, further work into the exact definition and extraction of texture density

should be highly encouraged. Texture has been addressed in the general areas of advanced image analysis. Research into this field takes much use of advanced stochastic methods to define, synthesize, and analyze textures from the real world. Perhaps the primary method could, somehow, benefit from such fields of research.

Texture density is in general, quite a simple concept. It is an image features which is expected to exists in all textures. But the simplicity breaks down when seeking to generalize over all textures. What to count? What is a texel?

Most importantly, for the primary method to work, the definition of texture density should probably not be vague and simple. In any regard, it seems absolutely crucial that such a concept receives new attention.

This brief chapter took a look at the major results achieved in this work. This chapter concludes the report.

A. Computational Experiments – Constraint Trials

This appendix chapter shows some three-dimensional plots of the constraint trials. The constraint trials are numerical tests made using constraint equation Φ. The primary intention with these trials was to test the usefulness of the partial derivatives estimated for different values of order p. The 3D plots shows very clearly that the derivatives estimated becomes increasingly exact and useful, for increasing order p.

A.1. Remarks

The 3D plots in this appendix show how the constraint equation Φ behaves if all known values for P, Q, Z, U V, W, and focal length f, at all image points (x, y), are inserted into it. This also involves the surface texture density and the *scaled* image texture density (using $\tau_{i,j}$). The results shown here are from trial runs using $T_6(m, n) = \cos m + \cos n$, which is the most complex function used to model the surface texture density.

For the formalisms behind the primary method to be correct, when inserting these values into constraint equation Φ, it should equal *zero* or very close to it. The 3D plots shows that, for increasing order p, constraint equation Φ does behave exactly like that. It equates very close the zero when $p > 9$ for all image points.

The plots shows trial runs for $p = 1$, 3, 6, 9, and 12. The minimization results shown in other chapters use $p = 12$. All other variables are identical to the ones defined in the numerical example in Section 3.4. The surfaces shown, could be called *constraint surfaces*. These surfaces are depicted in the (x, y)-space.

A.2. Graphics

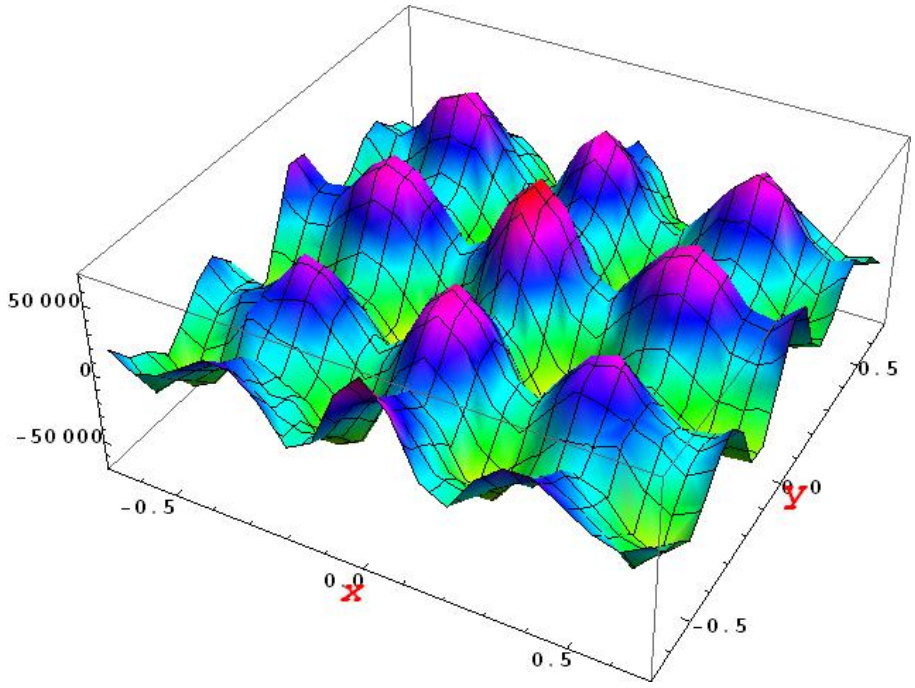


Figure A.1.: Constraint surface for $p = 1$.

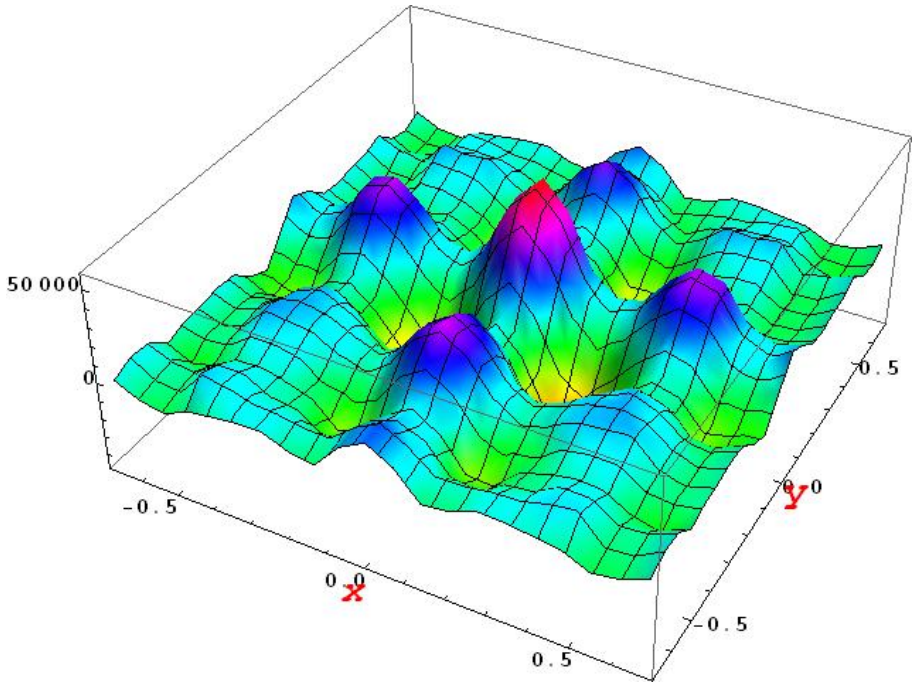


Figure A.2.: Constraint surface for $p = 3$.

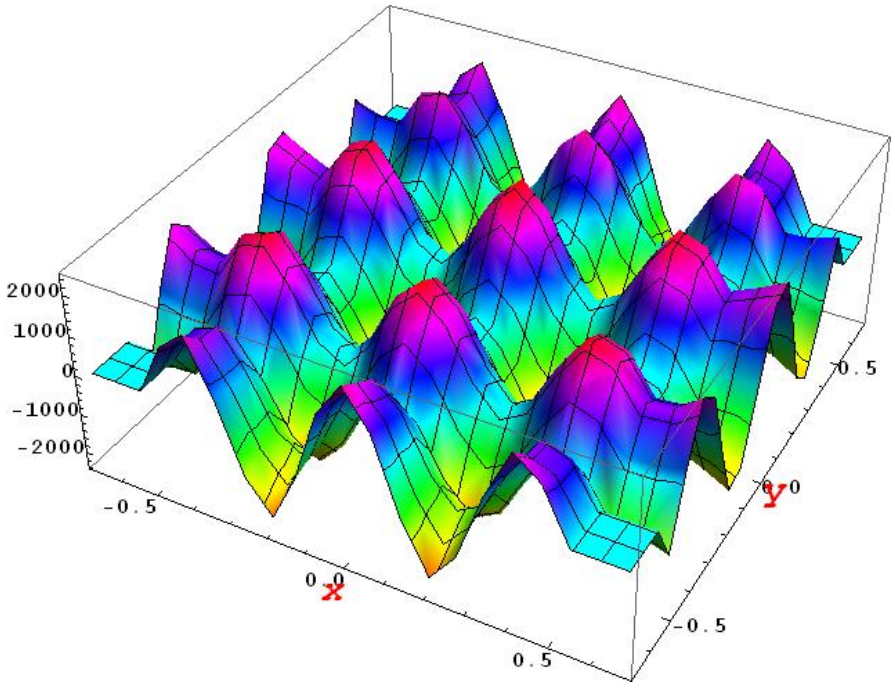


Figure A.3.: Constraint surface for $p = 6$.

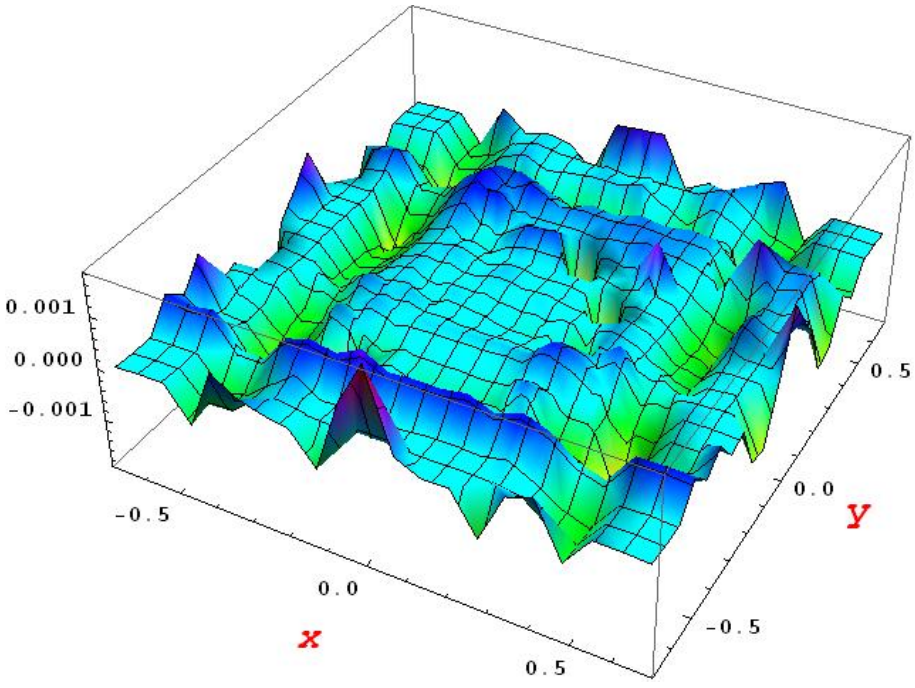


Figure A.4.: Constraint surface for $p = 9$.

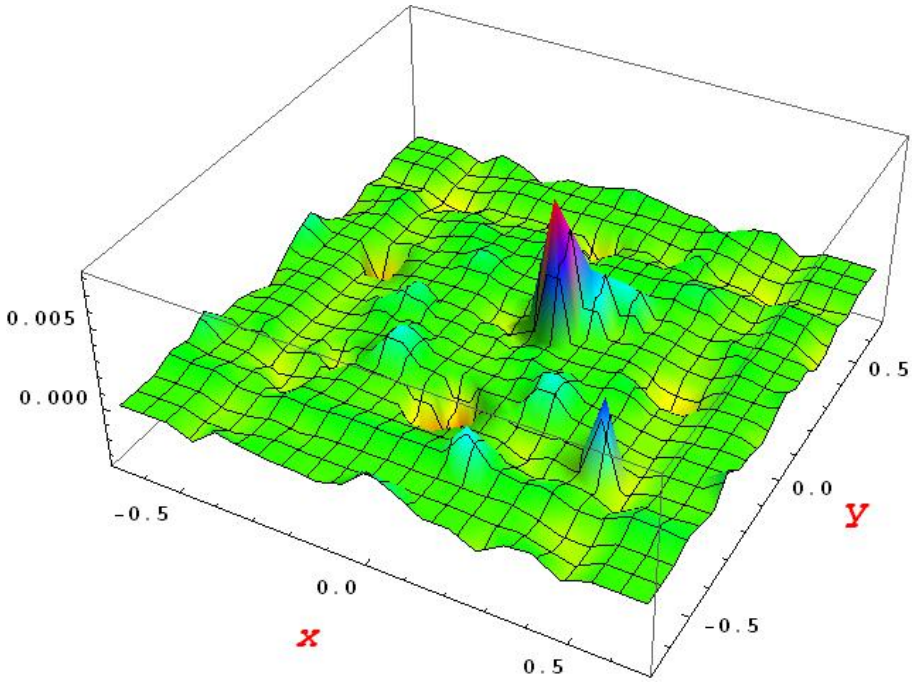


Figure A.5.: Constraint surface for $p = 12$.

B. Computational Experiments – Minimization Results

This appendix chapter show the results achieved in the computational experiments.

B.1. Remarks

The sole purpose of this appendix chapter is to show results from the computational experiments. The results are presented in page-size tables and 3D plots. The last column in all tables contains the *residual* of the minimization, i.e. the value to which the particular constraint system was minimized (in the least squares sense i.e. the sum of squares).

Take special notice of the the fact that different methods were used for minimization. Mathematica can apply *local* minimum search methods, e.g. *newton's method*, *conjugate gradient*, *levenberg marquardt*, and others. These methods are chosen automatically by Mathematica and the results achieved this way, do not convey the type of local search method used. Mathematica was trusted in being able to automatically choose the best method.

Sometimes local search methods are not good enough. What we really want to find are global minimums. Mathematica also has facilities for *global* minimization techniques, including *simulated annealing* and *random search*. Results are also shown for both of these methods. These global techniques were adjusted specifically for the different situations. See the individual table captions.

Table B.1 to B.3 show the single point minimization results achieved when using image point $\mathbf{p}_{11,136}$ defined in the numerical example in Section 3.4. Table B.4 to B.6 does the same for image point $\mathbf{p}_{11,359}$. The results in bold, are results which were within the correct value, with an error of $\pm 10^{-2}$.

See Section 3.4 for more information and comments related to the individual results.

B.2. Tables

Vars	P	Q	Z	U	V	W	Residual
1	-1.3594×10^{-8}	–	–	–	–	–	8.5455×10^{-9}
1	–	-5.4376×10^{-8}	–	–	–	–	8.5455×10^{-9}
1	–	–	-6.9704×10^{1}	–	–	–	9.0259×10^{-9}
1	–	–	–	6.2398×10^{-10}	–	–	5.7956×10^{-9}
1	–	–	–	–	-1.9574×10^{-10}	–	8.7359×10^{-9}
1	–	–	–	–	–	6.6667	9.0259×10^{-9}
2	-1.2794×10^{-8}	-3.1986×10^{-9}	–	–	–	–	8.5455×10^{-9}
2	-2.0666×10^{-7}	–	-6.9704×10^{1}	–	–	–	2.5692×10^{-10}
2	-4.3694×10^{-12}	–	–	6.2392×10^{-10}	–	–	5.7948×10^{-9}
2	-4.5775×10^{-12}	–	–	–	-1.9564×10^{-10}	–	8.7357×10^{-9}
2	-2.0666×10^{-7}	–	–	–	–	6.6667	2.5692×10^{-10}
2	–	-8.2663×10^{-7}	-6.9704×10^{1}	–	–	–	2.5692×10^{-10}
2	–	-1.0924×10^{-12}	–	6.2398×10^{-10}	–	–	5.7956×10^{-9}
2	–	-1.1449×10^{-12}	–	–	-1.9572×10^{-10}	–	8.7359×10^{-9}
2	–	-8.2663×10^{-7}	–	–	–	6.6667	2.5692×10^{-10}
2	–	–	-6.9704×10^{1}	1.6935×10^{-9}	–	–	6.9722×10^{-10}
2	–	–	-6.9704×10^{1}	–	-3.0665×10^{-10}	–	8.4926×10^{-9}
2	–	–	0.	–	–	0.	0.
2	–	–	–	6.9976×10^{-10}	-2.0944×10^{-10}	–	5.0756×10^{-9}
2	–	–	–	1.6935×10^{-9}	–	6.6667	6.9722×10^{-10}
2	–	–	–	–	-3.0678×10^{-10}	6.6667	8.4926×10^{-9}
3	-1.945×10^{-7}	-4.8625×10^{-8}	-6.9704×10^{1}	–	–	–	2.5692×10^{-10}
3	-4.3694×10^{-12}	-1.0923×10^{-12}	–	6.2392×10^{-10}	–	–	5.7948×10^{-9}
3	-4.577×10^{-12}	-1.1443×10^{-12}	–	–	-1.9561×10^{-10}	–	8.7357×10^{-9}
3	-1.945×10^{-7}	-4.8625×10^{-8}	–	–	–	6.6667	2.5692×10^{-10}
3	1.8766×10^{1}	–	-1.1382×10^{1}	3.5524×10^{-1}	–	–	4.1168×10^{-24}
3	-2.0353×10^{-7}	–	-6.9704×10^{1}	–	-1.3172×10^{-10}	–	1.6054×10^{-10}
3	0.	–	0.	–	–	0.	0.
3	-4.9×10^{-12}	–	–	6.9969×10^{-10}	-2.0942×10^{-10}	–	5.0747×10^{-9}
3	-1.4786×10^{-7}	–	–	5.2689×10^{-10}	–	6.6667	1.6054×10^{-10}
3	-2.0353×10^{-7}	–	–	–	-1.3172×10^{-10}	6.6667	1.6054×10^{-10}
3	–	-5.9145×10^{-7}	-6.9704×10^{1}	5.2689×10^{-10}	–	–	1.6054×10^{-10}
3	–	-8.1412×10^{-7}	-6.9704×10^{1}	–	-1.3172×10^{-10}	–	1.6054×10^{-10}
3	–	0.	0.	–	–	0.	0.
3	–	-1.2251×10^{-12}	–	6.9976×10^{-10}	-2.0944×10^{-10}	–	5.0755×10^{-9}
3	–	-5.9148×10^{-7}	–	5.2684×10^{-10}	–	6.6667	1.6054×10^{-10}
3	–	-8.1412×10^{-7}	–	–	-1.3172×10^{-10}	6.6667	1.6054×10^{-10}
3	–	–	-6.9704×10^{1}	1.9264×10^{-9}	3.4987×10^{-10}	–	1.6054×10^{-10}
3	–	–	0.	0.	–	0.	0.
3	–	–	0.	–	0.	0.	0.
3	–	–	–	1.9264×10^{-9}	3.4987×10^{-10}	6.6667	1.6054×10^{-10}
4	1.5575×10^{1}	1.2764×10^{1}	-9.4509	4.2803×10^{-1}	–	–	4.9582×10^{-24}
4	2.4654×10^{1}	-2.3548×10^{1}	-1.5392×10^{1}	–	-2.8311×10^{-1}	–	7.0987×10^{-24}
4	0.	0.	0.	–	–	0.	0.
4	-4.9×10^{-12}	-1.225×10^{-12}	–	6.997×10^{-10}	-2.0942×10^{-10}	–	5.0747×10^{-9}
4	-1.3274×10^{-1}	5.3097×10^{-1}	–	5.2689×10^{-10}	–	6.6667	1.6054×10^{-10}
4	2.2119×10^{-1}	-8.8478×10^{-1}	–	–	-1.3172×10^{-10}	6.6667	1.6054×10^{-10}
4	1.8766×10^{1}	–	-1.0277×10^{1}	3.5524×10^{-1}	1.48×10^{-1}	–	0.
4	0.	–	0.	0.	–	0.	0.
4	0.	–	0.	–	0.	0.	0.
4	3.2915×10^{-2}	–	–	3.1208×10^{-4}	7.802×10^{-5}	6.6725	1.5998×10^{-10}
4	–	-5.9811×10^{1}	-8.9562×10^{1}	-1.0121×10^{-1}	-2.5302×10^{-2}	–	8.557×10^{-10}
4	–	0.	0.	0.	–	0.	0.
4	–	0.	0.	–	0.	0.	0.
4	–	-6.8812×10^{-1}	–	-1.6135×10^{-3}	-4.0336×10^{-4}	6.6364	1.635×10^{-10}
4	–	–	0.	0.	0.	0.	0.
5	2.0678×10^{1}	-7.6451	-1.1313×10^{1}	4.0212×10^{-1}	2.1558×10^{-1}	–	5.6543×10^{-23}
5	0.	0.	0.	0.	–	0.	0.
5	0.	0.	0.	–	0.	0.	0.
5	3.44×10^{-2}	-6.9245×10^{-1}	–	-1.3033×10^{-3}	-3.2581×10^{-4}	6.6422	1.6293×10^{-10}
5	0.	–	0.	0.	0.	0.	0.
5	–	0.	0.	0.	0.	0.	0.
6	0.	0.	0.	0.	0.	0.	0.

Table B.1.: Single point results in $\mathbf{p}_{11,136}$ from numerical example in Section 3.4. Automatic local search method used.

Vars	P	Q	Z	U	V	W	Residual
1	**-1.3595×10^{-8}**	–	–	–	–	–	8.5455×10^{-9}
1	–	**-5.4382×10^{-8}**	–	–	–	–	8.5455×10^{-9}
1	–	–	**-6.9704×10^{1}**	–	–	–	9.0259×10^{-9}
1	–	–	–	**6.2404×10^{-10}**	–	–	5.7956×10^{-9}
1	–	–	–	–	**-1.9576×10^{-10}**	–	8.7359×10^{-9}
1	–	–	–	–	–	**6.6667**	9.0259×10^{-9}
2	1.7872×10^{-1}	-7.1488×10^{-1}	–	–	–	–	8.5455×10^{-9}
2	**-2.0666×10^{-7}**	–	**-6.9704×10^{1}**	–	–	–	2.5692×10^{-10}
2	**-5.7525×10^{-8}**	–	–	**1.2658×10^{-9}**	–	–	3.6391×10^{-10}
2	**-1.1077×10^{-8}**	–	–	–	**-8.4346×10^{-11}**	–	8.5059×10^{-9}
2	**-2.0666×10^{-7}**	–	–	–	–	**6.6667**	2.5692×10^{-10}
2	–	**-8.2663×10^{-7}**	**-6.9704×10^{1}**	–	–	–	2.5692×10^{-10}
2	–	**-2.301×10^{-7}**	–	**1.2658×10^{-9}**	–	–	3.6391×10^{-10}
2	–	**-4.431×10^{-8}**	–	–	**-8.4346×10^{-11}**	–	8.5059×10^{-9}
2	–	**-8.2663×10^{-7}**	–	–	–	**6.6667**	2.5692×10^{-10}
2	–	–	**-6.9704×10^{1}**	**1.6935×10^{-9}**	–	–	6.9722×10^{-10}
2	–	–	**-6.9704×10^{1}**	–	**-3.0669×10^{-10}**	–	8.4926×10^{-9}
2	–	–	-7.8719×10^{-1}	–	–	7.5288×10^{-2}	1.1511×10^{-12}
2	–	–	–	**6.8736×10^{-10}**	**-3.3283×10^{-10}**	–	4.9484×10^{-9}
2	–	–	–	**1.6935×10^{-9}**	–	**6.6667**	6.9722×10^{-10}
2	–	–	–	–	**-3.0669×10^{-10}**	**6.6667**	8.4926×10^{-9}
3	3.2778×10^{-2}	-1.3111×10^{-1}	**-6.9704×10^{1}**	–	–	–	2.5692×10^{-10}
3	-6.1457×10^{-2}	2.4583×10^{-1}	–	**1.2658×10^{-9}**	–	–	3.6391×10^{-10}
3	-6.1062×10^{-2}	2.4425×10^{-1}	–	–	**-8.4346×10^{-11}**	–	8.5059×10^{-9}
3	1.8422×10^{-1}	-7.3687×10^{-1}	–	–	–	**6.6667**	2.5692×10^{-10}
3	1.8766×10^{1}	–	-1.2592×10^{1}	3.5524×10^{-1}	–	–	0.
3	**-2.0353×10^{-7}**	–	**-6.9704×10^{1}**	–	**-1.3172×10^{-10}**	–	1.6054×10^{-10}
3	1.2669×10^{-1}	–	-9.3821×10^{-11}	–	–	8.9934×10^{-12}	5.9971×10^{-21}
3	**-6.4967×10^{-8}**	–	–	**1.3115×10^{-9}**	**1.9615×10^{-10}**	–	1.6054×10^{-10}
3	**-1.4786×10^{-7}**	–	–	**5.2689×10^{-10}**	–	**6.6667**	1.6054×10^{-10}
3	**-2.0353×10^{-7}**	–	–	–	**-1.3172×10^{-10}**	**6.6667**	1.6054×10^{-10}
3	–	**-5.9145×10^{-7}**	**-6.9704×10^{1}**	**5.2689×10^{-10}**	–	–	1.6054×10^{-10}
3	–	**-8.1412×10^{-7}**	**-6.9704×10^{1}**	–	**-1.3172×10^{-10}**	–	1.6054×10^{-10}
3	–	**-8.2663×10^{-7}**	-7.2198×10^{-6}	–	–	6.9052×10^{-7}	2.7563×10^{-24}
3	–	**-2.5987×10^{-7}**	–	**1.3115×10^{-9}**	**1.9615×10^{-10}**	–	1.6054×10^{-10}
3	–	**-5.9145×10^{-7}**	–	**5.2689×10^{-10}**	–	**6.6667**	1.6054×10^{-10}
3	–	**-8.1412×10^{-7}**	–	–	**-1.3172×10^{-10}**	**6.6667**	1.6054×10^{-10}
3	–	–	**-6.9704×10^{1}**	**1.9264×10^{-9}**	**3.4987×10^{-10}**	–	1.6054×10^{-10}
3	–	–	-5.1862×10^{-1}	**1.2601×10^{-11}**	–	4.9602×10^{-2}	3.8597×10^{-14}
3	–	–	-2.3566×10^{-1}	–	**-1.0369×10^{-12}**	2.2539×10^{-2}	9.7073×10^{-14}
3	–	–	–	**1.9264×10^{-9}**	**3.4987×10^{-10}**	**6.6667**	1.6054×10^{-10}
4	1.5897×10^{1}	1.1476×10^{1}	-9.2954	4.1936×10^{-1}	–	–	2.7254×10^{-25}
4	2.2993×10^{1}	-1.6908×10^{1}	-7.3543	–	-3.9429×10^{-1}	–	2.5035×10^{-23}
4	1.7138	6.512×10^{-1}	9.5722×10^{-10}	–	–	-9.5024×10^{-11}	1.4693×10^{-16}
4	-3.3501×10^{-1}	1.3401	–	**1.3115×10^{-9}**	**1.9615×10^{-10}**	–	1.6054×10^{-10}
4	-4.3321×10^{-1}	1.7329	–	**5.2689×10^{-10}**	–	**6.6667**	1.6054×10^{-10}
4	6.0358×10^{-2}	-2.4143×10^{-1}	–	–	**-1.3172×10^{-10}**	**6.6667**	1.6054×10^{-10}
4	1.7508×10^{1}	–	-8.7599	3.106×10^{-1}	7.765×10^{-2}	–	1.1399×10^{-14}
4	1.9723×10^{-1}	–	-5.7503×10^{-9}	**1.9057×10^{-13}**	–	5.5312×10^{-10}	2.2522×10^{-17}
4	**-2.0353×10^{-7}**	–	8.2276×10^{-1}	–	**1.5548×10^{-12}**	-7.8691×10^{-2}	2.2368×10^{-14}
4	-9.9177×10^{-1}	–	–	**-8.9158×10^{-3}**	**-2.2289×10^{-3}**	6.4993	1.7796×10^{-10}
4	–	-6.4656×10^{1}	-9.0687×10^{1}	-1.0694×10^{-1}	-2.6734×10^{-2}	–	9.4148×10^{-10}
4	–	**-5.9145×10^{-7}**	6.8893×10^{-1}	**-5.2076×10^{-12}**	–	-6.5891×10^{-2}	1.5683×10^{-14}
4	–	**-8.1413×10^{-7}**	3.9891×10^{-1}	–	**7.5382×10^{-13}**	-3.8153×10^{-2}	5.2581×10^{-15}
4	–	-4.2704	–	**-9.5609×10^{-3}**	**-2.3902×10^{-3}**	6.4872	1.7933×10^{-10}
4	–	–	1.5429	**-4.2641×10^{-11}**	**-7.7446×10^{-12}**	-1.4757×10^{-1}	7.8664×10^{-14}
5	2.1089×10^{1}	-9.2918	-1.1871×10^{1}	4.5959×10^{-1}	3.2564×10^{-1}	–	3.4926×10^{-24}
5	2.1933×10^{-1}	-8.7734×10^{-1}	7.3149×10^{-2}	**-5.509×10^{-13}**	–	-6.9961×10^{-3}	1.7681×10^{-16}
5	9.4761×10^{-1}	1.2468×10^{-1}	-5.9097×10^{-11}	–	**2.3894×10^{-14}**	6.0667×10^{-12}	1.1116×10^{-17}
5	-3.7533×10^{-1}	-1.0761	–	**-5.8963×10^{-3}**	**-1.4741×10^{-3}**	6.556	1.7176×10^{-10}
5	2.4898×10^{-1}	–	1.7663×10^{-1}	**-6.0519×10^{-6}**	**-1.513×10^{-6}**	-1.7007×10^{-2}	1.0037×10^{-15}
5	–	4.6294×10^{-1}	3.6867×10^{-1}	**-5.8297×10^{-6}**	**-1.4574×10^{-6}**	-3.537×10^{-2}	4.4359×10^{-15}
6	-3.9035×10^{-1}	4.6241×10^{-1}	5.7377×10^{-1}	**2.1097×10^{-5}**	**5.2742×10^{-6}**	-5.448×10^{-2}	1.1199×10^{-14}

Table B.2.: Single point results in $\mathbf{p}_{11,136}$ from numerical example in Section 3.4. Simulated annealing used as global search method.

Vars	P	Q	Z	U	V	W	Residual
1	**-1.3595×10^{-8}**	–	–	–	–	–	8.5455×10^{-9}
1	–	**-5.4382×10^{-8}**	–	–	–	–	8.5455×10^{-9}
1	–	–	**-6.9704×10^{1}**	–	–	–	9.0259×10^{-9}
1	–	–	–	**6.2404×10^{-10}**	–	–	5.7956×10^{-9}
1	–	–	–	–	**-1.9576×10^{-10}**	–	8.7359×10^{-9}
1	–	–	–	–	–	**6.6667**	9.0259×10^{-9}
2	-1.7368×10^{-1}	6.9474×10^{-1}	–	–	–	–	8.5455×10^{-9}
2	**-2.0666×10^{-7}**	–	**-6.9704×10^{1}**	–	–	–	2.5692×10^{-10}
2	**-5.7524×10^{-8}**	–	–	**1.2658×10^{-9}**	–	–	3.6391×10^{-10}
2	**-1.1077×10^{-8}**	–	–	–	**-8.4346×10^{-11}**	–	8.5059×10^{-9}
2	**-2.0666×10^{-7}**	–	–	–	–	**6.6667**	2.5692×10^{-10}
2	–	**-8.2664×10^{-7}**	**-6.9704×10^{1}**	–	–	–	2.5692×10^{-10}
2	–	**-2.301×10^{-7}**	–	**1.2658×10^{-9}**	–	–	3.6391×10^{-10}
2	–	**-4.4312×10^{-8}**	–	–	**-8.434×10^{-11}**	–	8.5059×10^{-9}
2	–	**-8.2663×10^{-7}**	–	–	–	**6.6667**	2.5692×10^{-10}
2	–	–	**-6.9704×10^{1}**	**1.6935×10^{-9}**	–	–	6.9722×10^{-10}
2	–	–	**-6.9704×10^{1}**	–	**-3.0669×10^{-10}**	–	8.4926×10^{-9}
2	–	–	-9.192×10^{-3}	–	–	8.7915×10^{-4}	1.5696×10^{-16}
2	–	–	–	**6.8736×10^{-10}**	**-3.3283×10^{-10}**	–	4.9484×10^{-9}
2	–	–	–	**1.6935×10^{-9}**	–	**6.6667**	6.9722×10^{-10}
2	–	–	–	–	**-3.0669×10^{-10}**	**6.6667**	8.4926×10^{-9}
3	2.1204×10^{-1}	-8.4814×10^{-1}	**-6.9704×10^{1}**	–	–	–	2.5692×10^{-10}
3	-1.1×10^{-1}	4.3998×10^{-1}	–	**1.2658×10^{-9}**	–	–	3.6391×10^{-10}
3	-8.3037×10^{-2}	3.3215×10^{-1}	–	–	**-8.4329×10^{-11}**	–	8.5059×10^{-9}
3	-1.97×10^{-1}	7.8798×10^{-1}	–	–	–	**6.6667**	2.5692×10^{-10}
3	1.8766×10^{1}	–	-1.2375×10^{1}	3.5524×10^{-1}	–	–	0.
3	**-2.0353×10^{-7}**	–	**-6.9704×10^{1}**	–	**-1.3172×10^{-10}**	–	1.6054×10^{-10}
3	3.2248×10^{-1}	–	5.4465×10^{-12}	–	–	-5.2394×10^{-13}	1.3188×10^{-22}
3	**-6.4967×10^{-8}**	–	–	**1.3115×10^{-9}**	**1.9615×10^{-10}**	–	1.6054×10^{-10}
3	**-1.4786×10^{-7}**	–	–	**5.269×10^{-10}**	–	**6.6667**	1.6054×10^{-10}
3	**-2.0353×10^{-7}**	–	–	–	**-1.3173×10^{-10}**	**6.6667**	1.6054×10^{-10}
3	–	**-5.9147×10^{-7}**	**-6.9704×10^{1}**	**5.2684×10^{-10}**	–	–	1.6054×10^{-10}
3	–	**-8.1412×10^{-7}**	**-6.9704×10^{1}**	–	**-1.3172×10^{-10}**	–	1.6054×10^{-10}
3	–	**-6.7722×10^{-3}**	-3.2926×10^{-10}	–	–	3.149×10^{-11}	1.3129×10^{-23}
3	–	**-2.5987×10^{-7}**	–	**1.3115×10^{-9}**	**1.9614×10^{-10}**	–	1.6054×10^{-10}
3	–	**-5.9145×10^{-7}**	–	**5.2689×10^{-10}**	–	**6.6667**	1.6054×10^{-10}
3	–	**-8.1412×10^{-7}**	–	–	**-1.3172×10^{-10}**	**6.6667**	1.6054×10^{-10}
3	–	–	**-6.9704×10^{1}**	**1.9264×10^{-9}**	**3.4987×10^{-10}**	–	1.6054×10^{-10}
3	–	–	9.5696×10^{-3}	**-2.3249×10^{-13}**	–	-9.1526×10^{-4}	1.3141×10^{-17}
3	–	–	9.5696×10^{-3}	–	**4.2147×10^{-14}**	-9.1526×10^{-4}	1.6007×10^{-16}
3	–	–	–	**1.9264×10^{-9}**	**3.4986×10^{-10}**	**6.6667**	1.6054×10^{-10}
4	1.5388×10^{1}	1.3513×10^{1}	-9.2395	4.3323×10^{-1}	–	–	1.5143×10^{-25}
4	2.2485×10^{1}	-1.4874×10^{1}	-6.6976	–	-4.482×10^{-1}	–	0.
4	5.5834×10^{-1}	-1.0066	-9.3752×10^{-24}	–	–	9.0161×10^{-25}	3.5326×10^{-46}
4	-2.5652×10^{-1}	1.0261	–	**1.3115×10^{-9}**	**1.9614×10^{-10}**	–	1.6054×10^{-10}
4	-3.3502×10^{-1}	1.3401	–	**5.2694×10^{-10}**	–	**6.6667**	1.6054×10^{-10}
4	1.635×10^{-1}	-6.5401×10^{-1}	–	–	**-1.3172×10^{-10}**	**6.6667**	1.6054×10^{-10}
4	1.8766×10^{1}	–	-1.015×10^{1}	3.5524×10^{-1}	-6.0551×10^{-1}	–	0.
4	3.467×10^{-1}	–	-8.9703×10^{-13}	**3.5927×10^{-17}**	–	8.6525×10^{-14}	4.8931×10^{-25}
4	**-2.8258×10^{-7}**	–	-2.7619×10^{-11}	–	**-2.5268×10^{-23}**	2.6415×10^{-12}	2.2251×10^{-34}
4	9.6626×10^{-1}	–	–	**9.6419×10^{-3}**	**2.4105×10^{-3}**	6.8476	1.4444×10^{-10}
4	–	-5.4826×10^{1}	-8.835×10^{1}	-9.5028×10^{-2}	-2.3757×10^{-2}	–	7.7227×10^{-10}
4	–	**-1.4164×10^{-6}**	-3.2344×10^{-10}	**-5.1062×10^{-21}**	–	3.0934×10^{-11}	2.5939×10^{-32}
4	–	**-1.4949×10^{-6}**	-2.2688×10^{-11}	–	**4.747×10^{-24}**	2.1699×10^{-12}	6.3423×10^{-34}
4	–	4.1621×10^{-1}	–	**9.9033×10^{-4}**	**2.4758×10^{-4}**	6.6853	1.5877×10^{-10}
4	–	–	-9.192×10^{-3}	**2.5404×10^{-13}**	**4.6139×10^{-14}**	8.7915×10^{-4}	2.7919×10^{-18}
5	2.0578×10^{1}	-7.2479	-1.1854×10^{1}	1.1473×10^{-1}	-5.9406×10^{-1}	–	0.
5	3.5411×10^{-1}	5.2251×10^{-1}	-2.4392×10^{-11}	**1.4066×10^{-15}**	–	2.3617×10^{-12}	7.6207×10^{-22}
5	-4.3105×10^{1}	1.7473×10^{2}	-3.9579×10^{-12}	–	**1.4334×10^{-17}**	3.8201×10^{-13}	1.5527×10^{-22}
5	8.0854×10^{-1}	3.0529×10^{-2}	–	**8.0762×10^{-3}**	**2.0191×10^{-3}**	6.8182	1.4688×10^{-10}
5	-9.3387×10^{-2}	–	8.7608×10^{-3}	**1.1054×10^{-7}**	**2.7636×10^{-8}**	-8.3583×10^{-4}	2.5614×10^{-18}
5	–	-1.168×10^{-1}	8.6432×10^{-3}	**3.4217×10^{-8}**	**8.5543×10^{-9}**	-8.2601×10^{-4}	2.4762×10^{-18}
6	6.9359×10^{-1}	7.8322×10^{-1}	6.7374×10^{-3}	**-8.5414×10^{-7}**	**-2.1354×10^{-7}**	-6.6041×10^{-4}	1.3611×10^{-18}

Table B.3.: Single point results in $\mathbf{p}_{11,136}$ from numerical example in Section 3.4. Random search used as global search method.

Vars	P	Q	Z	U	V	W	Residual
1	$\mathbf{1.1525\times10^{-8}}$	-	-	-	-	-	7.466×10^{-7}
1	-	$\mathbf{-8.9643\times10^{-9}}$	-	-	-	-	7.466×10^{-7}
1	-	-	$\mathbf{-6.9704\times10^{1}}$	-	-	-	7.4929×10^{-7}
1	-	-	-	$\mathbf{-5.4344\times10^{-9}}$	-	-	5.2567×10^{-7}
1	-	-	-	-	$\mathbf{-3.6138\times10^{-9}}$	-	6.4352×10^{-7}
1	-	-	-	-	-	**6.6667**	7.4929×10^{-7}
2	$\mathbf{4.3442\times10^{-9}}$	$\mathbf{-5.5854\times10^{-9}}$	-	-	-	-	7.466×10^{-7}
2	$\mathbf{1.7023\times10^{-7}}$	-	$\mathbf{-6.9704\times10^{1}}$	-	-	-	7.2105×10^{-7}
2	$\mathbf{3.754\times10^{-11}}$	-	-	$\mathbf{-5.4335\times10^{-9}}$	-	-	5.2561×10^{-7}
2	$\mathbf{3.4964\times10^{-11}}$	-	-	-	$\mathbf{-3.6103\times10^{-9}}$	-	6.4356×10^{-7}
2	$\mathbf{1.7023\times10^{-7}}$	-	-	-	-	**6.6667**	7.2105×10^{-7}
2	-	$\mathbf{-1.324\times10^{-7}}$	$\mathbf{-6.9704\times10^{1}}$	-	-	-	7.2105×10^{-7}
2	-	$\mathbf{-4.8265\times10^{-11}}$	-	$\mathbf{-5.4335\times10^{-9}}$	-	-	5.2557×10^{-7}
2	-	$\mathbf{-4.4912\times10^{-11}}$	-	-	$\mathbf{-3.6069\times10^{-9}}$	-	6.4359×10^{-7}
2	-	$\mathbf{-1.324\times10^{-7}}$	-	-	-	**6.6667**	7.2105×10^{-7}
2	-	-	$\mathbf{-6.9704\times10^{1}}$	$\mathbf{-5.8203\times10^{-9}}$	-	-	5.0627×10^{-7}
2	-	-	$\mathbf{-6.9704\times10^{1}}$	-	$\mathbf{-5.8043\times10^{-9}}$	-	5.8663×10^{-7}
2	-	-	0.	-	-	0.	0.
2	-	-	-	$\mathbf{-4.8243\times10^{-9}}$	$\mathbf{-1.0863\times10^{-9}}$	-	5.1889×10^{-7}
2	-	-	-	$\mathbf{-5.8203\times10^{-9}}$	-	**6.6667**	5.0627×10^{-7}
2	-	-	-	-	$\mathbf{-5.8043\times10^{-9}}$	**6.6667**	5.8663×10^{-7}
3	$\mathbf{6.4164\times10^{-8}}$	$\mathbf{-8.2498\times10^{-8}}$	$\mathbf{-6.9704\times10^{1}}$	-	-	-	7.2105×10^{-7}
3	$\mathbf{3.7533\times10^{-11}}$	$\mathbf{-4.8257\times10^{-11}}$	-	$\mathbf{-5.4326\times10^{-9}}$	-	-	5.2551×10^{-7}
3	$\mathbf{3.4897\times10^{-11}}$	$\mathbf{-4.4868\times10^{-11}}$	-	-	$\mathbf{-3.6034\times10^{-9}}$	-	6.4364×10^{-7}
3	-4.042	5.1969	-	-	-	1.1499×10^{-18}	5.2684×10^{-23}
3	-1.0724×10^{1}	-	-1.3311	-6.2168×10^{-1}	-	-	0.
3	$\mathbf{3.3759\times10^{-7}}$	-	$\mathbf{-6.9704\times10^{1}}$	-	$\mathbf{-7.3549\times10^{-9}}$	-	4.8718×10^{-7}
3	**0.**	-	0.	-	-	0.	0.
3	$\mathbf{5.6457\times10^{-11}}$	-	-	$\mathbf{-4.8266\times10^{-9}}$	$\mathbf{-1.0837\times10^{-9}}$	-	5.1884×10^{-7}
3	$\mathbf{1.4024\times10^{-7}}$	-	-	$\mathbf{-5.7205\times10^{-9}}$	-	**6.6667**	4.8718×10^{-7}
3	$\mathbf{3.3759\times10^{-7}}$	-	-	-	$\mathbf{-7.3548\times10^{-9}}$	**6.6667**	4.8718×10^{-7}
3	-	$\mathbf{-1.0908\times10^{-7}}$	$\mathbf{-6.9704\times10^{1}}$	$\mathbf{-5.7205\times10^{-9}}$	-	-	4.8718×10^{-7}
3	-	8.3406	-1.0662	-	7.993×10^{-1}	-	0.
3	-	**0.**	0.	-	-	0.	0.
3	-	$\mathbf{-7.2432\times10^{-11}}$	-	$\mathbf{-4.8281\times10^{-9}}$	$\mathbf{-1.082\times10^{-9}}$	-	5.1881×10^{-7}
3	-	$\mathbf{-1.0908\times10^{-7}}$	-	$\mathbf{-5.7205\times10^{-9}}$	-	**6.6667**	4.8718×10^{-7}
3	-	$\mathbf{-2.6257\times10^{-7}}$	-	-	$\mathbf{-7.3549\times10^{-9}}$	**6.6667**	4.8718×10^{-7}
3	-	-	$\mathbf{-6.9704\times10^{1}}$	$\mathbf{-9.7856\times10^{-9}}$	$\mathbf{5.2266\times10^{-9}}$	-	4.8718×10^{-7}
3	-	-	0.	**0.**	-	0.	0.
3	-	-	0.	-	**0.**	0.	0.
3	-	-	-	$\mathbf{-9.7855\times10^{-9}}$	$\mathbf{5.2265\times10^{-9}}$	**6.6667**	4.8718×10^{-7}
4	-2.2802×10^{1}	-9.3941	-6.1621	-2.9237×10^{-1}	-	-	2.258×10^{-22}
4	-5.1144	4.3628	-5.2323×10^{-1}	-	1.5281	-	8.0755×10^{-24}
4	**0.**	**0.**	0.	-	-	0.	0.
4	$\mathbf{5.615\times10^{-11}}$	$\mathbf{-7.2192\times10^{-11}}$	-	$\mathbf{-4.8304\times10^{-9}}$	$\mathbf{-1.0794\times10^{-9}}$	-	5.1876×10^{-7}
4	-4.0233	5.2114	-	-1.7347	-	6.9793	0.
4	-1.1603	-9.0245×10^{-1}	-	-	$\mathbf{-7.3549\times10^{-9}}$	**6.6667**	4.8718×10^{-7}
4	-1.0724×10^{1}	-	-8.2376	-6.2168×10^{-1}	2.28	-	0.
4	**0.**	-	0.	**0.**	-	0.	0.
4	**0.**	-	0.	-	**0.**	0.	0.
4	4.1234×10^{-1}	-	-	1.151×10^{-2}	-1.4798×10^{-2}	6.5432	5.2536×10^{-7}
4	-	8.3406	-1.485×10^{1}	-1.2819	7.993×10^{-1}	-	0.
4	-	**0.**	0.	**0.**	-	0.	0.
4	-	**0.**	0.	-	**0.**	0.	0.
4	-	-5.5018×10^{-1}	-	1.9235×10^{-2}	-2.4731×10^{-2}	6.4604	5.5357×10^{-7}
4	-	-	0.	**0.**	**0.**	0.	0.
5	-4.4113	4.9097	-1.5762×10^{1}	-6.3713×10^{-1}	7.8542×10^{-1}	-	9.2257×10^{-23}
5	**0.**	**0.**	0.	**0.**	-	0.	0.
5	**0.**	**0.**	0.	-	**0.**	0.	0.
5	4.648×10^{-1}	-6.7547×10^{-1}	-	3.4373×10^{-2}	-4.4193×10^{-2}	6.2981	6.1585×10^{-7}
5	**0.**	-	0.	**0.**	**0.**	0.	0.
5	-	**0.**	0.	**0.**	**0.**	0.	0.
6	**0.**	**0.**	0.	**0.**	**0.**	0.	0.

Table B.4.: Single point results in $\mathbf{p}_{11,359}$ from numerical example in Section 3.4. Automatic local search method used.

Vars	P	Q	Z	U	V	W	Residual
1	**1.1527×10^{-8}**	–	–	–	–	–	7.466×10^{-7}
1	–	**-8.9652×10^{-9}**	–	–	–	–	7.466×10^{-7}
1	–	–	**-6.9704×10^{1}**	–	–	–	7.4929×10^{-7}
1	–	–	–	**-5.435×10^{-9}**	–	–	5.2567×10^{-7}
1	–	–	–	–	**-3.6142×10^{-9}**	–	6.4352×10^{-7}
1	–	–	–	–	–	**6.6667**	7.4929×10^{-7}
2	-1.1525	-8.964×10^{-1}	–	–	–	–	7.466×10^{-7}
2	**1.7023×10^{-7}**	–	**-6.9704×10^{1}**	–	–	–	7.2105×10^{-7}
2	**3.3883×10^{-8}**	–	–	**-5.8563×10^{-9}**	–	–	4.9863×10^{-7}
2	**-3.9273×10^{-8}**	–	–	–	**-4.8606×10^{-9}**	–	6.1803×10^{-7}
2	**1.7023×10^{-7}**	–	–	–	–	**6.6667**	7.2105×10^{-7}
2	–	**-1.324×10^{-7}**	**-6.9704×10^{1}**	–	–	–	7.2105×10^{-7}
2	–	**-2.6353×10^{-8}**	–	**-5.8563×10^{-9}**	–	–	4.9863×10^{-7}
2	–	**3.0546×10^{-8}**	–	–	**-4.8606×10^{-9}**	–	6.1803×10^{-7}
2	–	8.3406	–	–	–	1.7355×10^{-29}	1.9271×10^{-50}
2	–	–	**-6.9704×10^{1}**	**-5.8203×10^{-9}**	–	–	5.0627×10^{-7}
2	–	–	**-6.9704×10^{1}**	–	**-5.8043×10^{-9}**	–	5.8663×10^{-7}
2	–	–	-7.8719×10^{-1}	–	–	7.5288×10^{-2}	9.5563×10^{-11}
2	–	–	–	**-4.8243×10^{-9}**	**-1.0863×10^{-9}**	–	5.1889×10^{-7}
2	–	–	–	**-5.8203×10^{-9}**	–	**6.6667**	5.0627×10^{-7}
2	–	–	–	–	**-5.8043×10^{-9}**	**6.6667**	5.8663×10^{-7}
3	-2.5	-1.9445	**-6.9704×10^{1}**	–	–	–	7.2105×10^{-7}
3	4.7272×10^{-2}	3.6767×10^{-2}	–	**-5.8563×10^{-9}**	–	–	4.9863×10^{-7}
3	-3.0277×10^{-1}	-2.3549×10^{-1}	–	–	**-4.8606×10^{-9}**	–	6.1803×10^{-7}
3	-2.5699	6.3418	–	–	–	-8.1247×10^{-17}	4.3741×10^{-23}
3	-1.0724×10^{1}	–	-3.0215	-6.2168×10^{-1}	–	–	0.
3	**3.3759×10^{-7}**	–	**-6.9704×10^{1}**	–	**-7.3549×10^{-9}**	–	4.8718×10^{-7}
3	**1.7023×10^{-7}**	–	-1.0246	–	–	9.7999×10^{-2}	1.5581×10^{-10}
3	**6.8998×10^{-8}**	–	–	**-7.7856×10^{-9}**	**2.6551×10^{-9}**	–	4.8718×10^{-7}
3	**1.4026×10^{-7}**	–	–	**-5.7204×10^{-9}**	–	**6.6667**	4.8718×10^{-7}
3	**3.3759×10^{-7}**	–	–	–	**-7.3549×10^{-9}**	**6.6667**	4.8718×10^{-7}
3	–	**-1.0908×10^{-7}**	**-6.9704×10^{1}**	**-5.7205×10^{-9}**	–	–	4.8718×10^{-7}
3	–	8.3406	-1.3696	–	7.993×10^{-1}	–	0.
3	–	**-1.3241×10^{-7}**	-6.8931×10^{-1}	–	–	6.5928×10^{-2}	7.0515×10^{-11}
3	–	**-5.3665×10^{-8}**	–	**-7.7856×10^{-9}**	**2.6551×10^{-9}**	–	4.8718×10^{-7}
3	–	8.3406	–	2.3271×10^{-2}	–	-5.1118×10^{-28}	1.6719×10^{-47}
3	–	**-2.6257×10^{-7}**	–	–	**-7.3549×10^{-9}**	**6.6667**	4.8718×10^{-7}
3	–	–	**-6.9704×10^{1}**	**-9.7855×10^{-9}**	**5.2265×10^{-9}**	–	4.8718×10^{-7}
3	–	–	3.1422	**2.6237×10^{-10}**	–	-3.0053×10^{-1}	1.0288×10^{-9}
3	–	–	-1.5458	–	**-1.2872×10^{-10}**	1.4785×10^{-1}	2.8851×10^{-10}
3	–	–	–	**-9.7856×10^{-9}**	**5.2266×10^{-9}**	**6.6667**	4.8718×10^{-7}
4	-1.4745×10^{1}	-3.1275	-3.4555	-4.5214×10^{-1}	–	–	4.2824×10^{-23}
4	1.0832×10^{1}	1.6766×10^{1}	1.1674	–	3.9764×10^{-1}	–	3.9962×10^{-23}
4	-1.3197	-9.1491×10^{-1}	1.0352×10^{-9}	–	–	-9.9445×10^{-11}	4.4552×10^{-18}
4	6.5087×10^{-1}	5.0623×10^{-1}	–	**-7.7856×10^{-9}**	**2.6551×10^{-9}**	–	4.8718×10^{-7}
4	-2.6712	6.263	–	-5.9399×10^{-1}	–	1.5867	6.815×10^{-24}
4	-5.1234	4.3558	–	–	2.6259×10^{-1}	1.1438	0.
4	-1.0724×10^{1}	–	1.5131	-6.2168×10^{-1}	5.4054×10^{-1}	–	0.
4	-2.4538×10^{-2}	–	-1.4793×10^{-9}	**-1.1202×10^{-15}**	–	1.416×10^{-10}	3.3071×10^{-19}
4	1.447	–	-8.903×10^{-10}	–	**-1.7946×10^{-13}**	8.1382×10^{-11}	3.9261×10^{-16}
4	2.8446×10^{-1}	–	–	**8.0322×10^{-3}**	-1.0327×10^{-2}	6.5805	5.1336×10^{-7}
4	–	8.3406	-2.9961	-5.95×10^{-1}	7.993×10^{-1}	–	0.
4	–	2.0508	-1.6177×10^{-10}	**-7.5685×10^{-14}**	–	1.8809×10^{-11}	6.581×10^{-16}
4	–	1.089	-3.0181×10^{-10}	–	**5.07×10^{-14}**	3.0493×10^{-11}	3.7637×10^{-17}
4	–	-1.547	–	4.8632×10^{-2}	-6.2527×10^{-2}	6.1451	6.8465×10^{-7}
4	–	–	3.8652×10^{-1}	**5.4263×10^{-11}**	**-2.8982×10^{-11}**	-3.6968×10^{-2}	1.498×10^{-11}
5	-2.7748	6.1825	-6.804×10^{-1}	-6.2169×10^{-1}	7.9929×10^{-1}	–	3.9482×10^{-24}
5	5.6477×10^{-1}	4.3926×10^{-1}	-1.384×10^{-2}	**-1.1352×10^{-12}**	–	1.3237×10^{-3}	1.9206×10^{-14}
5	2.3131	1.799	1.9993	–	**2.1096×10^{-10}**	-1.9122×10^{-1}	4.0082×10^{-10}
5	9.1791×10^{-1}	-1.1347	–	5.6394×10^{-2}	-7.2507×10^{-2}	6.0619	7.2706×10^{-7}
5	5.2429×10^{-1}	–	5.7406×10^{-2}	**-1.1932×10^{-5}**	**1.5342×10^{-5}**	-5.3625×10^{-3}	3.6353×10^{-13}
5	–	4.5438×10^{-1}	1.2471	**3.2041×10^{-4}**	**-4.1196×10^{-4}**	-1.2271×10^{-1}	1.3941×10^{-10}
6	-2.5979	1.8134	2.2852	**8.6699×10^{-3}**	-1.1147×10^{-2}	-3.1154×10^{-1}	1.5288×10^{-10}

Table B.5.: Single point results in $\mathbf{p}_{11,359}$ from numerical example in Section 3.4. Simulated annealing used as global search method.

Vars	P	Q	Z	U	V	W	Residual
1	**1.1527×10^{-8}**	–	–	–	–	–	7.466×10^{-7}
1	–	**-8.9652×10^{-9}**	–	–	–	–	7.466×10^{-7}
1	–	–	**-6.9704×10^{1}**	–	–	–	7.4929×10^{-7}
1	–	–	–	**-5.435×10^{-9}**	–	–	5.2567×10^{-7}
1	–	–	–	–	**-3.6142×10^{-9}**	–	6.4352×10^{-7}
1	–	–	–	–	–	**6.6667**	7.4929×10^{-7}
2	-4.684×10^{-1}	-3.6431×10^{-1}	–	–	–	–	7.466×10^{-7}
2	**1.7023×10^{-7}**	–	**-6.9704×10^{1}**	–	–	–	7.2105×10^{-7}
2	**3.3884×10^{-8}**	–	–	**-5.8563×10^{-9}**	–	–	4.9863×10^{-7}
2	**-3.9273×10^{-8}**	–	–	–	**-4.8606×10^{-9}**	–	6.1803×10^{-7}
2	**1.7022×10^{-7}**	–	–	–	–	**6.6667**	7.2105×10^{-7}
2	–	**-1.324×10^{-7}**	**-6.9704×10^{1}**	–	–	–	7.2105×10^{-7}
2	–	**-2.6351×10^{-8}**	–	**-5.8563×10^{-9}**	–	–	4.9863×10^{-7}
2	–	**3.0544×10^{-8}**	–	–	**-4.8605×10^{-9}**	–	6.1803×10^{-7}
2	–	8.3406	–	–	–	-1.7746×10^{-19}	2.0148×10^{-30}
2	–	–	**-6.9704×10^{1}**	**-5.8203×10^{-9}**	–	–	5.0627×10^{-7}
2	–	–	**-6.9704×10^{1}**	–	**-5.8043×10^{-9}**	–	5.8663×10^{-7}
2	–	–	-9.192×10^{-3}	–	–	8.7915×10^{-4}	1.303×10^{-14}
2	–	–	–	**-4.824×10^{-9}**	**-1.0867×10^{-9}**	–	5.1889×10^{-7}
2	–	–	–	**-5.8204×10^{-9}**	–	**6.6667**	5.0627×10^{-7}
2	–	–	–	–	**-5.8043×10^{-9}**	**6.6667**	5.8663×10^{-7}
3	-3.3313×10^{-1}	-2.591×10^{-1}	**-6.9704×10^{1}**	–	–	–	7.2105×10^{-7}
3	-1.9086×10^{-1}	-1.4845×10^{-1}	–	**-5.8562×10^{-9}**	–	–	4.9863×10^{-7}
3	-1.536×10^{-1}	-1.1947×10^{-1}	–	–	**-4.8606×10^{-9}**	–	6.1803×10^{-7}
3	-4.3016	4.9949	–	–	–	4.9699×10^{-21}	1.5803×10^{-33}
3	-1.0724×10^{1}	–	-3.9481	-6.2168×10^{-1}	–	–	0.
3	**3.3759×10^{-7}**	–	**-6.9704×10^{1}**	–	**-7.3549×10^{-9}**	–	4.8718×10^{-7}
3	-5.1033×10^{-2}	–	-6.8521×10^{-12}	–	–	6.5635×10^{-13}	2.4601×10^{-23}
3	**6.8998×10^{-8}**	–	–	**-7.7856×10^{-9}**	**2.6551×10^{-9}**	–	4.8718×10^{-7}
3	**1.4024×10^{-7}**	–	–	**-5.7205×10^{-9}**	–	**6.6667**	4.8718×10^{-7}
3	**3.3759×10^{-7}**	–	–	–	**-7.3549×10^{-9}**	**6.6667**	4.8718×10^{-7}
3	–	**-1.0908×10^{-7}**	**-6.9704×10^{1}**	**-5.7204×10^{-9}**	–	–	4.8718×10^{-7}
3	–	8.3406	-2.0244	–	7.993×10^{-1}	–	0.
3	–	5.7494×10^{-1}	3.4729×10^{-11}	–	–	-3.4019×10^{-12}	1.387×10^{-19}
3	–	**-5.3665×10^{-8}**	–	**-7.7856×10^{-9}**	**2.6551×10^{-9}**	–	4.8718×10^{-7}
3	–	8.3406	–	4.5891	–	-3.7406×10^{-22}	8.9521×10^{-36}
3	–	8.3406	–	–	1.9057	1.5895×10^{1}	0.
3	–	–	**-6.9704×10^{1}**	**-9.7855×10^{-9}**	**5.2265×10^{-9}**	–	4.8718×10^{-7}
3	–	–	9.5696×10^{-3}	**7.9907×10^{-13}**	–	-9.1526×10^{-4}	9.5424×10^{-15}
3	–	–	9.5696×10^{-3}	–	**7.9691×10^{-13}**	-9.1526×10^{-4}	1.1057×10^{-14}
3	–	–	–	**-9.785×10^{-9}**	**5.2259×10^{-9}**	**6.6667**	4.8718×10^{-7}
4	-2.2167×10^{2}	-1.6407×10^{2}	-6.5322×10^{1}	-3.0074×10^{-2}	–	–	0.
4	-5.9695	3.6977	2.5195×10^{-2}	–	1.8029	–	0.
4	1.0409	8.2415×10^{-1}	9.0611×10^{-12}	–	–	-8.6711×10^{-13}	5.7631×10^{-24}
4	8.1982×10^{-2}	6.3764×10^{-2}	–	**-7.7855×10^{-9}**	**2.6551×10^{-9}**	–	4.8718×10^{-7}
4	-4.5999	4.7629	–	-1.2965	–	5.9639	0.
4	-6.6486	3.1695	–	–	3.3838	1.0725×10^{1}	0.
4	-1.0724×10^{1}	–	-9.128	-6.2168×10^{-1}	1.5519	–	0.
4	**1.364×10^{-7}**	–	-4.6215×10^{-12}	**-3.7946×10^{-22}**	–	4.4201×10^{-13}	2.1417×10^{-33}
4	**3.3872×10^{-7}**	–	-9.119×10^{-14}	–	**-9.6287×10^{-24}**	8.7216×10^{-15}	8.338×10^{-37}
4	-3.784×10^{-1}	–	–	-1.1369×10^{-2}	1.4618×10^{-2}	6.7886	4.534×10^{-7}
4	–	8.3406	-2.9188	-2.3358	7.993×10^{-1}	–	0.
4	–	**-1.1752×10^{-7}**	1.0079×10^{-10}	**8.2601×10^{-21}**	–	-9.6394×10^{-12}	1.0188×10^{-30}
4	–	**-2.6935×10^{-7}**	3.1703×10^{-11}	–	**3.3633×10^{-21}**	-3.0321×10^{-12}	1.0079×10^{-31}
4	–	-4.6931×10^{-2}	–	**1.7392×10^{-3}**	**-2.2361×10^{-3}**	6.648	4.9267×10^{-7}
4	–	–	-9.192×10^{-3}	**-1.2904×10^{-12}**	**6.8923×10^{-13}**	8.7915×10^{-4}	8.4721×10^{-15}
5	-2.3523×10^{1}	-9.955	-6.4478×10^{1}	-1.4254	2.6984	–	0.
5	6.8963×10^{1}	5.3601×10^{1}	1.0223×10^{-10}	**5.1914×10^{-16}**	–	-9.7757×10^{-12}	2.1815×10^{-21}
5	-1.249×10^{2}	-9.7143×10^{1}	2.5996×10^{-10}	–	**-5.657×10^{-17}**	-2.4864×10^{-11}	1.4095×10^{-23}
5	-2.1058	-2.2897	–	2.2535×10^{-2}	-2.8973×10^{-2}	6.425	5.6631×10^{-7}
5	-1.7744×10^{-1}	–	8.2072×10^{-3}	**6.1576×10^{-7}**	**-7.9169×10^{-7}**	-7.9156×10^{-4}	6.5323×10^{-15}
5	–	-5.2156×10^{-2}	8.8138×10^{-3}	**-2.4425×10^{-7}**	**3.1404×10^{-7}**	-8.4036×10^{-4}	7.887×10^{-15}
6	6.7962×10^{-1}	5.6179×10^{-1}	6.3935×10^{-3}	**1.1393×10^{-7}**	**-1.4648×10^{-7}**	-6.1271×10^{-4}	4.0661×10^{-15}

Table B.6.: Single point results in $\mathbf{p}_{11,359}$ from numerical example in Section 3.4. Random search used as global search method.

Vars	P	Q	Z	U	V	W	Residual
1	8.8753×10^{-9}	–	–	–	–	–	1.2071×10^{-6}
1	–	-4.3343×10^{-9}	–	–	–	–	1.2077×10^{-6}
1	–	–	-6.9704×10^{1}	–	–	–	1.2063×10^{-6}
1	–	–	–	-1.1696×10^{-9}	–	–	1.1679×10^{-6}
1	–	–	–	–	-2.8647×10^{-10}	–	1.2067×10^{-6}
1	–	–	–	–	–	6.6667	1.2063×10^{-6}
2	2.4516×10^{-9}	-3.6697×10^{-9}	–	–	–	–	1.2074×10^{-6}
2	5.6004×10^{-9}	–	-6.9704×10^{1}	–	–	–	1.2055×10^{-6}
2	8.193×10^{-12}	–	–	-1.1697×10^{-9}	–	–	1.1679×10^{-6}
2	7.711×10^{-12}	–	–	–	-2.8654×10^{-10}	–	1.2067×10^{-6}
2	5.5982×10^{-9}	–	–	–	–	6.6667	1.2055×10^{-6}
2	–	-4.439×10^{-9}	-6.9704×10^{1}	–	–	–	1.2046×10^{-6}
2	–	-1.2263×10^{-11}	–	-1.1696×10^{-9}	–	–	1.1679×10^{-6}
2	–	-1.1534×10^{-11}	–	–	-2.8632×10^{-10}	–	1.2067×10^{-6}
2	–	-4.4388×10^{-9}	–	–	–	6.6667	1.2046×10^{-6}
2	–	–	-6.9704×10^{1}	-1.172×10^{-9}	–	–	1.1647×10^{-6}
2	–	–	-6.9704×10^{1}	–	-2.9444×10^{-10}	–	1.2035×10^{-6}
2	–	–	0.	–	–	0.	0.
2	–	–	–	-1.0164×10^{-9}	-2.6455×10^{-10}	–	1.1709×10^{-6}
2	–	–	–	-1.172×10^{-9}	–	6.6667	1.1647×10^{-6}
2	–	–	–	–	-2.939×10^{-10}	6.6667	1.2035×10^{-6}
3	-3.6971×10^{-9}	-6.0102×10^{-9}	-6.9704×10^{1}	–	–	–	1.2045×10^{-6}
3	8.1932×10^{-12}	-1.2264×10^{-11}	–	-1.1697×10^{-9}	–	–	1.1679×10^{-6}
3	7.707×10^{-12}	-1.1537×10^{-11}	–	–	-2.8639×10^{-10}	–	1.2067×10^{-6}
3	-3.699×10^{-9}	-6.0113×10^{-9}	–	–	–	6.6667	1.2045×10^{-6}
3	-2.4254×10^{1}	–	-9.231×10^{1}	-1.602×10^{-1}	–	–	2.5227×10^{9}
3	2.8259×10^{-9}	–	-6.9704×10^{1}	–	-2.7288×10^{-10}	–	1.2033×10^{-6}
3	0.	–	0.	–	–	0.	0.
3	7.1196×10^{-12}	–	–	-1.0164×10^{-9}	-2.6456×10^{-10}	–	1.1709×10^{-6}
3	1.3998×10^{-8}	–	–	-1.2454×10^{-9}	–	6.6667	1.1602×10^{-6}
3	2.8261×10^{-9}	–	–	–	-2.7293×10^{-10}	6.6667	1.2033×10^{-6}
3	–	-2.6057×10^{-9}	-6.9704×10^{1}	-1.1604×10^{-9}	–	–	1.1641×10^{-6}
3	–	-2.1505×10^{-9}	-6.9704×10^{1}	–	-2.3909×10^{-10}	–	1.2032×10^{-6}
3	–	0.	0.	–	–	0.	0.
3	–	-1.0656×10^{-11}	–	-1.0163×10^{-9}	-2.6452×10^{-10}	–	1.1709×10^{-6}
3	–	-2.6062×10^{-9}	–	-1.1604×10^{-9}	–	6.6667	1.1641×10^{-6}
3	–	-2.1505×10^{-9}	–	–	-2.3909×10^{-10}	6.6667	1.2032×10^{-6}
3	–	–	-6.9704×10^{1}	-1.1845×10^{-9}	4.1553×10^{-11}	–	1.1646×10^{-6}
3	–	–	0.	0.	–	0.	0.
3	–	–	0.	–	0.	0.	0.
3	–	–	–	-1.1845×10^{-9}	4.1521×10^{-11}	6.6667	1.1646×10^{-6}
4	-3.3189×10^{1}	-1.4853×10^{1}	-1.0086×10^{2}	-1.6435×10^{-1}	–	–	3.2815×10^{8}
4	-4.8941×10^{-10}	-2.3796×10^{-9}	-6.9704×10^{1}	–	-2.3689×10^{-10}	–	1.2032×10^{-6}
4	0.	0.	0.	–	–	0.	0.
4	7.1188×10^{-12}	-1.0656×10^{-11}	–	-1.0163×10^{-9}	-2.6453×10^{-10}	–	1.1709×10^{-6}
4	3.4002×10^{-8}	1.2236×10^{-8}	–	-1.4049×10^{-9}	–	6.6667	1.1566×10^{-6}
4	-5.044×10^{-10}	-2.3867×10^{-9}	–	–	-2.3667×10^{-10}	6.6667	1.2032×10^{-6}
4	-2.4238×10^{1}	–	-9.1699×10^{1}	-1.5746×10^{-1}	8.7757×10^{-2}	–	1.94×10^{9}
4	0.	–	0.	0.	–	0.	0.
4	0.	–	0.	–	0.	0.	0.
4	1.6569×10^{-8}	–	–	-1.32×10^{-9}	2.032×10^{-10}	6.6667	1.1591×10^{-6}
4	–	-3.9426×10^{-9}	-6.9704×10^{1}	-1.1983×10^{-9}	1.4595×10^{-10}	–	1.1636×10^{-6}
4	–	0.	0.	0.	–	0.	0.
4	–	0.	0.	–	0.	0.	0.
4	–	-3.9426×10^{-9}	–	-1.1983×10^{-9}	1.4595×10^{-10}	6.6667	1.1636×10^{-6}
4	–	–	0.	0.	0.	0.	0.
5	3.3461×10^{-8}	1.1392×10^{-8}	-6.9704×10^{1}	-1.4183×10^{-9}	6.6445×10^{-11}	–	1.1565×10^{-6}
5	0.	0.	0.	0.	–	0.	0.
5	0.	0.	0.	–	0.	0.	0.
5	3.3465×10^{-8}	1.1393×10^{-8}	–	-1.4183×10^{-9}	6.6379×10^{-11}	6.6667	1.1565×10^{-6}
5	0.	–	0.	0.	0.	0.	0.
5	–	0.	0.	0.	0.	0.	0.
6	0.	0.	0.	0.	0.	0.	0.

Table B.7.: Mutiple point results using all 4 image points defined in the numerical example in Section 3.4. Automatic local search method used. All variables considered surface-global.

Vars	P	Q	Z	U	V	W	Residual
1	8.8762×10^{-9}	–	–	–	–	–	1.2071×10^{-6}
1	–	-4.3347×10^{-9}	–	–	–	–	1.2077×10^{-6}
1	–	–	-6.9704×10^{1}	–	–	–	1.2063×10^{-6}
1	–	–	–	-1.1697×10^{-9}	–	–	1.1679×10^{-6}
1	–	–	–	–	-2.865×10^{-10}	–	1.2067×10^{-6}
1	–	–	–	–	–	6.6667	1.2063×10^{-6}
2	7.1879×10^{-9}	-1.3081×10^{-9}	–	–	–	–	1.2071×10^{-6}
2	5.6005×10^{-9}	–	-6.9704×10^{1}	–	–	–	1.2055×10^{-6}
2	1.6124×10^{-8}	–	–	-1.2556×10^{-9}	–	–	1.1608×10^{-6}
2	6.9228×10^{-9}	–	–	–	-2.3692×10^{-10}	–	1.2054×10^{-6}
2	5.6005×10^{-9}	–	–	–	–	6.6667	1.2055×10^{-6}
2	–	-4.4388×10^{-9}	-6.9704×10^{1}	–	–	–	1.2046×10^{-6}
2	–	-2.5024×10^{-9}	–	-1.1585×10^{-9}	–	–	1.1674×10^{-6}
2	–	-2.1052×10^{-9}	–	–	-2.3282×10^{-10}	–	1.2064×10^{-6}
2	–	-4.4388×10^{-9}	–	–	–	6.6667	1.2046×10^{-6}
2	–	–	-6.9704×10^{1}	-1.172×10^{-9}	–	–	1.1647×10^{-6}
2	–	–	-6.9704×10^{1}	–	-2.939×10^{-10}	–	1.2035×10^{-6}
2	–	–	-7.8719×10^{-1}	–	–	7.5288×10^{-2}	1.5384×10^{-10}
2	–	–	–	-1.1844×10^{-9}	4.8916×10^{-11}	–	1.1678×10^{-6}
2	–	–	–	-1.172×10^{-9}	–	6.6667	1.1647×10^{-6}
2	–	–	–	–	-2.939×10^{-10}	6.6667	1.2035×10^{-6}
3	-3.699×10^{-9}	-6.0113×10^{-9}	-6.9704×10^{1}	–	–	–	1.2045×10^{-6}
3	3.0854×10^{-8}	1.0844×10^{-8}	–	-1.3826×10^{-9}	–	–	1.1567×10^{-6}
3	9.6996×10^{-9}	2.4183×10^{-9}	–	–	-2.7869×10^{-10}	–	1.2052×10^{-6}
3	-3.6991×10^{-9}	-6.0113×10^{-9}	–	–	–	6.6667	1.2045×10^{-6}
3	1.3998×10^{-8}	–	-6.9704×10^{1}	-1.2454×10^{-9}	–	–	1.1602×10^{-6}
3	2.8258×10^{-9}	–	-6.9704×10^{1}	–	-2.7288×10^{-10}	–	1.2033×10^{-6}
3	5.6009×10^{-9}	–	-9.3218×10^{-8}	–	–	8.9156×10^{-9}	2.1561×10^{-24}
3	1.8427×10^{-8}	–	–	-1.3352×10^{-9}	2.236×10^{-10}	–	1.1595×10^{-6}
3	1.3998×10^{-8}	–	–	-1.2454×10^{-9}	–	6.6667	1.1602×10^{-6}
3	2.826×10^{-9}	–	–	–	-2.7288×10^{-10}	6.6667	1.2033×10^{-6}
3	–	-2.6057×10^{-9}	-6.9704×10^{1}	-1.1604×10^{-9}	–	–	1.1641×10^{-6}
3	–	-2.1505×10^{-9}	-6.9704×10^{1}	–	-2.3909×10^{-10}	–	1.2032×10^{-6}
3	–	-4.4388×10^{-9}	-3.2259×10^{-5}	–	–	3.0854×10^{-6}	2.5801×10^{-19}
3	–	-3.8968×10^{-9}	–	-1.198×10^{-9}	1.5214×10^{-10}	–	1.1668×10^{-6}
3	–	-2.6057×10^{-9}	–	-1.1604×10^{-9}	–	6.6667	1.1641×10^{-6}
3	–	-2.1505×10^{-9}	–	–	-2.3909×10^{-10}	6.6667	1.2032×10^{-6}
3	–	–	-6.9704×10^{1}	-1.1845×10^{-9}	4.1554×10^{-11}	–	1.1646×10^{-6}
3	–	–	-1.2245×10^{-1}	-2.059×10^{-12}	–	1.1712×10^{-2}	3.5945×10^{-12}
3	–	–	-2.5865	–	-1.0906×10^{-11}	2.4737×10^{-1}	1.657×10^{-9}
3	–	–	–	-1.1845×10^{-9}	4.1549×10^{-11}	6.6667	1.1646×10^{-6}
4	3.4001×10^{-8}	1.2235×10^{-8}	-6.9704×10^{1}	-1.4049×10^{-9}	–	–	1.1566×10^{-6}
4	-4.8941×10^{-10}	-2.3796×10^{-9}	-6.9704×10^{1}	–	-2.3689×10^{-10}	–	1.2032×10^{-6}
4	-2.3862×10^{-9}	-5.4186×10^{-9}	4.3208×10^{-9}	–	–	-4.1325×10^{-10}	4.6281×10^{-27}
4	3.0476×10^{-8}	1.0016×10^{-8}	–	-1.3987×10^{-9}	7.2533×10^{-11}	–	1.1566×10^{-6}
4	3.4×10^{-8}	1.2234×10^{-8}	–	-1.4049×10^{-9}	–	6.6667	1.1566×10^{-6}
4	-4.897×10^{-10}	-2.3799×10^{-9}	–	–	-2.369×10^{-10}	6.6667	1.2032×10^{-6}
4	1.6567×10^{-8}	–	-6.9704×10^{1}	-1.32×10^{-9}	2.0319×10^{-10}	–	1.1591×10^{-6}
4	1.3998×10^{-8}	–	-3.4869×10^{-5}	-6.2301×10^{-16}	–	3.335×10^{-6}	2.9034×10^{-19}
4	2.8259×10^{-9}	–	-4.4586×10^{-5}	–	-1.7455×10^{-16}	4.2644×10^{-6}	4.9234×10^{-19}
4	1.6564×10^{-8}	–	–	-1.32×10^{-9}	2.0313×10^{-10}	6.6667	1.1591×10^{-6}
4	–	-3.9426×10^{-9}	-6.9704×10^{1}	-1.1983×10^{-9}	1.4595×10^{-10}	–	1.1636×10^{-6}
4	–	-2.6057×10^{-9}	-5.3206×10^{-6}	-8.8573×10^{-17}	–	5.0887×10^{-7}	6.7824×10^{-21}
4	–	2.5217×10^{-1}	-9.4525×10^{-9}	–	-8.9511×10^{-13}	9.0494×10^{-10}	7.6868×10^{-14}
4	–	-3.9426×10^{-9}	–	-1.1983×10^{-9}	1.4595×10^{-10}	6.6667	1.1636×10^{-6}
4	–	–	-3.3204×10^{-1}	-5.6426×10^{-12}	1.9795×10^{-13}	3.1757×10^{-2}	2.6427×10^{-11}
5	3.346×10^{-8}	1.139×10^{-8}	-6.9704×10^{1}	-1.4183×10^{-9}	6.64×10^{-11}	–	1.1565×10^{-6}
5	3.3917×10^{-8}	1.2197×10^{-8}	1.0943×10^{-7}	2.2051×10^{-18}	–	-1.0466×10^{-8}	2.8507×10^{-24}
5	-7.6702×10^{-10}	-1.9608×10^{-9}	-1.9256×10^{-9}	–	-6.7168×10^{-21}	1.8417×10^{-10}	9.1822×10^{-28}
5	3.346×10^{-8}	1.1391×10^{-8}	–	-1.4183×10^{-9}	6.6399×10^{-11}	6.6667	1.1565×10^{-6}
5	1.6567×10^{-8}	–	-2.7446×10^{-5}	-5.1974×10^{-16}	8.0007×10^{-17}	2.625×10^{-6}	1.7971×10^{-19}
5	–	-3.9426×10^{-9}	-1.6065×10^{-5}	-2.7619×10^{-16}	3.3639×10^{-17}	1.5365×10^{-6}	6.1812×10^{-20}
6	3.346×10^{-8}	1.139×10^{-8}	-3.8863×10^{-4}	-7.9075×10^{-15}	3.702×10^{-16}	3.7169×10^{-5}	3.5949×10^{-17}

Table B.8.: Mutiple point results using all 4 image points defined in the numerical example in Section 3.4. Simulated annealing used as global search method. All variables considered surface-global.

Vars	P	Q	Z	U	V	W	Residual
1	8.8762×10^{-9}	–	–	–	–	–	1.2071×10^{-6}
1	–	-4.3347×10^{-9}	–	–	–	–	1.2077×10^{-6}
1	–	–	-6.9704×10^{1}	–	–	–	1.2063×10^{-6}
1	–	–	–	-1.1697×10^{-9}	–	–	1.1679×10^{-6}
1	–	–	–	–	-2.865×10^{-10}	–	1.2067×10^{-6}
1	–	–	–	–	–	6.6667	1.2063×10^{-6}
2	7.1879×10^{-9}	-1.3081×10^{-9}	–	–	–	–	1.2071×10^{-6}
2	5.6005×10^{-9}	–	-6.9704×10^{1}	–	–	–	1.2055×10^{-6}
2	1.6125×10^{-8}	–	–	-1.2556×10^{-9}	–	–	1.1608×10^{-6}
2	6.9229×10^{-9}	–	–	–	-2.3692×10^{-10}	–	1.2054×10^{-6}
2	5.5962×10^{-9}	–	–	–	–	6.6667	1.2055×10^{-6}
2	–	-4.4391×10^{-9}	-6.9704×10^{1}	–	–	–	1.2046×10^{-6}
2	–	-2.505×10^{-9}	–	-1.1584×10^{-9}	–	–	1.1674×10^{-6}
2	–	-2.104×10^{-9}	–	–	-2.3282×10^{-10}	–	1.2064×10^{-6}
2	–	-4.4388×10^{-9}	–	–	–	6.6667	1.2046×10^{-6}
2	–	–	-6.9704×10^{1}	-1.172×10^{-9}	–	–	1.1647×10^{-6}
2	–	–	-6.9704×10^{1}	–	-2.9391×10^{-10}	–	1.2035×10^{-6}
2	–	–	-9.192×10^{-3}	–	–	8.7915×10^{-4}	2.0977×10^{-14}
2	–	–	–	-1.1844×10^{-9}	4.8933×10^{-11}	–	1.1678×10^{-6}
2	–	–	–	-1.172×10^{-9}	–	6.6667	1.1647×10^{-6}
2	–	–	–	–	-2.9403×10^{-10}	6.6667	1.2035×10^{-6}
3	-3.699×10^{-9}	-6.0113×10^{-9}	-6.9704×10^{1}	–	–	–	1.2045×10^{-6}
3	3.0854×10^{-8}	1.0844×10^{-8}	–	-1.3826×10^{-9}	–	–	1.1567×10^{-6}
3	9.6946×10^{-9}	2.4163×10^{-9}	–	–	-2.7866×10^{-10}	–	1.2052×10^{-6}
3	-3.6969×10^{-9}	-6.0099×10^{-9}	–	–	–	6.6667	1.2045×10^{-6}
3	1.4001×10^{-8}	–	-6.9704×10^{1}	-1.2454×10^{-9}	–	–	1.1602×10^{-6}
3	2.8259×10^{-9}	–	-6.9704×10^{1}	–	-2.7288×10^{-10}	–	1.2033×10^{-6}
3	-1.538×10^{-9}	–	9.0788×10^{-14}	–	–	-8.6832×10^{-15}	2.0471×10^{-36}
3	1.8424×10^{-8}	–	–	-1.3351×10^{-9}	2.2358×10^{-10}	–	1.1595×10^{-6}
3	1.3998×10^{-8}	–	–	-1.2454×10^{-9}	–	6.6667	1.1602×10^{-6}
3	2.8258×10^{-9}	–	–	–	-2.7288×10^{-10}	6.6667	1.2033×10^{-6}
3	–	-2.6056×10^{-9}	-6.9704×10^{1}	-1.1604×10^{-9}	–	–	1.1641×10^{-6}
3	–	-2.1506×10^{-9}	-6.9704×10^{1}	–	-2.3908×10^{-10}	–	1.2032×10^{-6}
3	–	-8.6554×10^{-8}	9.1731×10^{-12}	–	–	-8.7734×10^{-13}	3.0866×10^{-32}
3	–	-3.8997×10^{-9}	–	-1.198×10^{-9}	1.5219×10^{-10}	–	1.1668×10^{-6}
3	–	-2.6057×10^{-9}	–	-1.1604×10^{-9}	–	6.6667	1.1641×10^{-6}
3	–	-2.1506×10^{-9}	–	–	-2.3909×10^{-10}	6.6667	1.2032×10^{-6}
3	–	–	-6.9704×10^{1}	-1.1845×10^{-9}	4.1552×10^{-11}	–	1.1646×10^{-6}
3	–	–	9.5696×10^{-3}	1.6091×10^{-13}	–	-9.1526×10^{-4}	2.1952×10^{-14}
3	–	–	9.5696×10^{-3}	–	4.035×10^{-14}	-9.1526×10^{-4}	2.2684×10^{-14}
3	–	–	–	-1.1845×10^{-9}	4.1555×10^{-11}	6.6667	1.1646×10^{-6}
4	3.3991×10^{-8}	1.2231×10^{-8}	-6.9704×10^{1}	-1.4048×10^{-9}	–	–	1.1566×10^{-6}
4	-4.8941×10^{-10}	-2.3796×10^{-9}	-6.9704×10^{1}	–	-2.3689×10^{-10}	–	1.2032×10^{-6}
4	-1.0564×10^{-2}	-1.3034×10^{-2}	1.6945×10^{-16}	–	–	-1.6208×10^{-17}	4.2302×10^{-32}
4	3.0476×10^{-8}	1.0016×10^{-8}	–	-1.3987×10^{-9}	7.2533×10^{-11}	–	1.1566×10^{-6}
4	3.4006×10^{-8}	1.2237×10^{-8}	–	-1.4049×10^{-9}	–	6.6667	1.1566×10^{-6}
4	-4.8689×10^{-10}	-2.3792×10^{-9}	–	–	-2.369×10^{-10}	6.6667	1.2032×10^{-6}
4	1.6565×10^{-8}	–	-6.9704×10^{1}	-1.32×10^{-9}	2.0325×10^{-10}	–	1.1591×10^{-6}
4	2.0275×10^{-7}	–	8.4172×10^{-13}	2.6986×10^{-23}	–	-8.0505×10^{-14}	2.8716×10^{-34}
4	7.8937×10^{-8}	–	2.6576×10^{-12}	–	-1.1189×10^{-23}	-2.5418×10^{-13}	1.9325×10^{-33}
4	1.6567×10^{-8}	–	–	-1.32×10^{-9}	2.0319×10^{-10}	6.6667	1.1591×10^{-6}
4	–	-3.9426×10^{-9}	-6.9704×10^{1}	-1.1983×10^{-9}	1.4595×10^{-10}	–	1.1636×10^{-6}
4	–	1.1183×10^{-7}	1.4415×10^{-11}	3.4573×10^{-22}	–	-1.3787×10^{-12}	9.7426×10^{-32}
4	–	-1.0239×10^{-7}	1.309×10^{-11}	–	-4.3492×10^{-22}	-1.2519×10^{-12}	6.5382×10^{-32}
4	–	-3.9422×10^{-9}	–	-1.1983×10^{-9}	1.4594×10^{-10}	6.6667	1.1636×10^{-6}
4	–	–	-9.192×10^{-3}	-1.5621×10^{-13}	5.481×10^{-15}	8.7915×10^{-4}	2.0253×10^{-14}
5	3.3462×10^{-8}	1.1392×10^{-8}	-6.9704×10^{1}	-1.4183×10^{-9}	6.6454×10^{-11}	–	1.1565×10^{-6}
5	1.0098×10^{-1}	4.3868×10^{-2}	1.6958×10^{-12}	1.74×10^{-17}	–	-1.6206×10^{-13}	3.9089×10^{-23}
5	4.6996×10^{-2}	2.1956×10^{-2}	1.0416×10^{-12}	–	3.1107×10^{-18}	-9.9579×10^{-14}	3.7233×10^{-24}
5	3.346×10^{-8}	1.139×10^{-8}	–	-1.4183×10^{-9}	6.6399×10^{-11}	6.6667	1.1565×10^{-6}
5	4.5279×10^{-8}	–	-5.7482×10^{-11}	-1.2821×10^{-21}	3.9857×10^{-22}	5.4978×10^{-12}	7.9946×10^{-31}
5	–	-1.6364×10^{-8}	2.9416×10^{-9}	5.2404×10^{-20}	-2.0039×10^{-20}	-2.8135×10^{-10}	2.0901×10^{-27}
6	-4.8547×10^{-8}	-2.6189×10^{-8}	-2.8894×10^{-10}	-3.6443×10^{-21}	1.0834×10^{-21}	2.7635×10^{-11}	2.0608×10^{-29}

Table B.9.: Mutiple point results using all 4 image points defined in the numerical example in Section 3.4. Random search used as global search method. All variables considered surface-global.

Bibliography

[1] J. Aloimonos. Shape from texture. *Biol. Cybern.*, 58(5):345–360, 1988.

[2] Jens Arnspang. Fundamentals of texture flow equations in vision calculus. *Pattern Recognition Letters*, 12(4):211–218, April 1991.

[3] O. Ben-Shahar and S.W. Zucker. The perceptual organization of texture flow: a contextual inference approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(4):401–417, April 2003.

[4] Jonas Gaarding. Shape from texture for smooth curved surfaces. In *ECCV '92: Proceedings of the Second European Conference on Computer Vision*, pages 630–638, London, UK, 1992. Springer-Verlag.

[5] Jonas Gaarding. Direct estimation of shape from texture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(11):1202–1208, Nov 1993.

[6] James J. Gibson. *The perception of the visual world*. Houghton Mifflin, 1950.

[7] James J. Gibson. The perception of visual surfaces. *The American Journal of Psychology*, 63(3):367–384, jul 1950.

[8] Berthold K. P. Horn. *Robot Vision*. The MIT Press, mit press ed edition, 3 1986.

[9] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, 1981.

[10] S. S. Beauchemin J. L. Barron and D. J. Fleet. On optical flow. In I. Plander, editor, *6th Int. Conf. on Artificial Intelligence and Information-Control Systems of Robots (AIICSR)*, pages 3–14, Bratislava, Slovakia, 1994. World Scientific. Sept. 12-16, 1994, Smolenice Castle, Slovakia.

[11] Ken-ichi Kanatani and Tsai-Chia Chou. Shape from texture: General principle. *Artificial Intelligence*, 38(1):1–48, February 1989.

[12] Angeline Loh. *The Recovery of 3-D Structure Using Visual Texture Patterns*. PhD thesis, The University of Western Australia, 2006.

[13] Angeline Loh and Richard Hartley. Shape from non-homogeneous, non-stationary, anisotropic, perspective texture. In *BMVC05*, 2005.

[14] Angeline Loh and Peter Kovesi. Estimation of surface normal of a curved surface using texture. In Changming Sun, Hugues Talbot, Sébastien Ourselin, and Tony Adriaansen, editors, *DICTA*, pages 155–164. CSIRO Publishing, 2003.

[15] Kent A. Stevens. Surface perception from local analysis of texture and contour. Technical report, Cambridge, MA, USA, 1980.

[16] Kent A. Stevens. The information content of texture gradients. *Biological Cybernetics*, 42(2):95–105, November 1981.

[17] Kent A. Stevens. Slant-tilt: The visual encoding of surface orientation. *Biological Cybernetics*, 46(3):183–195, March 1983.

[18] Yu-Wing Tai, M.S. Brown, and Chi-Keung Tang. Robust estimation of texture flow via dense feature sampling. *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 17-22 June 2007.

[19] Lore Thaler, James T. Todd, and Tjeerd M.H. Dijkstra. The effects of phase on the perception of 3d shape from texture: Psychophysics and modeling. *Vision Research*, 47(3):411–427, February 2007.

[20] Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998.

[21] M. Tuceryan and A. K. Jain. *Texture Analysis*, chapter Chapter 2.1, pages 207–248. World Scientific Publishing Co., 1998.

[22] Chris Urmson and William Whittaker. Self-driving cars and the urban challenge. *Intelligent Systems, IEEE*, 23(2):66–68, March-April 2008.

[23] Andrew P. Witkin. Recovering surface shape and orientation from texture. *Artificial Intelligence*, 17(1-3):17–45, August 1981.

[24] Robert C. Wrede and Murray Spiegel. *Schaum's Outline of Advanced Calculus, Second Edition*. McGraw-Hill, 2 edition, 2 2002.