Imperial College London
Department of Computing

Towards Real-time Lighting-aware Scene Capture from a Moving Camera

Jan Jachnik

July 2016

Supervised by Prof. Andrew Davison

Submitted in part fulfilment of the requirements for the degree of PhD in Computing and the Diploma of Imperial College London. This thesis is entirely my own work, and, except where otherwise indicated, describes my own research.

Abstract

The level of complexity of maps created by monocular SLAM is on the rise. Increases in computational power have taken us from sparse feature maps to fully dense 3D reconstructions. Still, none of these are making full use of the wealth of information available from a live, monocular video feed. Aside from geometry there are the effects of lighting, reflection and shadow which are often ignored but give us vital clues into the types of surfaces being observed.

We take some steps to extend the maps generated by monocular SLAM by considering real-time acquisition of surface reflectance and lighting information. In robotics, such information could be used to help determine materials, aiding object detection and semantic understanding, thus enabling better interaction with the environment. In augmented reality lighting and reflectance information is essential to make virtual objects blend seamlessly into the real world.

In this thesis we will demonstrate real-time capture of planar surface light-fields, a convenient representation to infer lighting and reflectance information. On a tangent we then investigate how sculptors manipulate geometry to alter the effects of illumination and reflectance, changing our perception and enhancing details; a technique required to bring a piece of work to life when sculpting in a medium of constant albedo. We attempt to apply some of these sculptors techniques in a mesh editing tool we call *sculptural stylisation*. Finally, we investigate methods for recovering the geometry of surfaces from a monocular camera in an attempt to extend our work on planar surface light-fields to work in 3D: we present various ways to generate depth-maps from a monocular video stream and detail a system to fuse them together into a consistent 3D model.

Acknowledgements

Firstly, I'd like to thank Andrew Davison for being an excellent supervisor and taking me on, even though I came to him with zero experience in computer vision. I was looking at PhD positions in a broad range of subjects when I first met Andy and he showed me a selection of demos from his group at the time - I said to myself "that looks awesome, I want to do that". *Nothing beats a good demo!* I didn't know that computer vision was such a rapidly growing field and one of the most sought after skills in the tech industry. Working with Andy has provided a great stepping stone for my career.

Thanks to all of the other lab members in both the robot vision group and Dyson robotics lab. It was a great help when formulating ideas to bounce them off other people, they were always there to listen and provide their feedback. It has been a huge asset to have their support when working through some of the really tough problems. I hope I managed to helped them as much as they helped me.

I'd like to thank Dan Goldman, Linjie Luo and Nathan Carr of Adobe Research for supporting me on my two internships and for the ongoing collaborations which led to a significant part of my thesis. It was a great experience to work with other groups and get a glimpse of life as a researcher in industry.

I owe a huge amount to my wonderful wife, Connie, who has put up with me for all this time and was hugely supportive when the going got tough. Thank you to my Mum and Sister for always encouraging me even though they had no idea what I was doing. Finally, a huge thank you to my Dad, also Jan, for always believing I could do it even when I didn't. He was always enthusiastic and helped me feel positive about my PhD even when I was really struggling. His own interest in the subject made discussions very useful and he was never short of ideas.

Contents

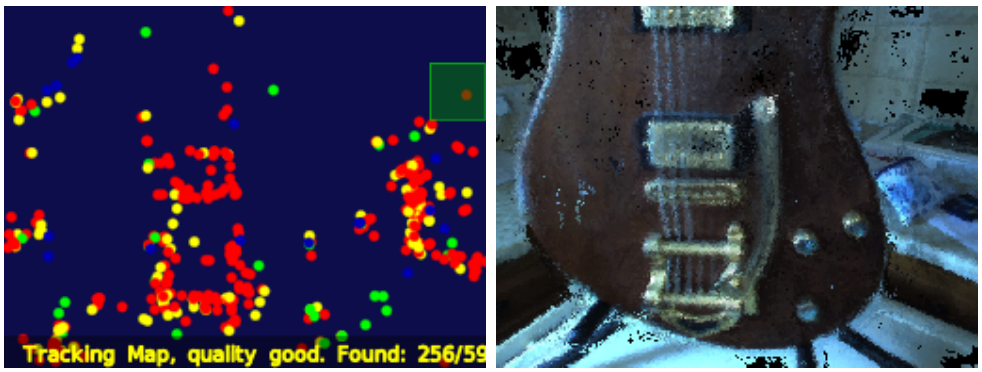
1

Introduction

Poor scene perception is one of the biggest factors limiting the way computers can interact with the world around us. For years computers have had the ability to *see* the world, following the invention of the digital imaging sensor, but the ability to interpret the raw pixel signals into a meaningful and rich understanding of the environment around them is an ongoing challenge.

As a human looking at a scene we unconsciously understand the geometry of the world, we can interpret lighting, shadow and reflectance, we recognise objects and materials, and we know how to interact with what we see. The goal of this thesis is to take some steps to help improve the ability of a computer to perceive these same properties. Our human perception can be divided into two classes: raw observations and semantic understanding. The raw observations are purely physical, such as the interaction of light with surfaces and the geometry of the world. Our semantic understanding is higher level and allows us to identify objects and materials. We take the stance that a more thorough interpretation of raw observations, such as scene geometry, lighting and reflectance, can aid semantic understanding further down the line. Indeed, there is current research using full 3D models to aid in object detection and recognition [132] instead of the more traditional computer vision problem of recognition from single images. Therefore, the goal of this thesis is to improve the ability of a computer to capture and interpret these raw observations.

One of the classic problems of scene perception is Simultaneous Localisation And Mapping (SLAM). For a camera to know where it is in the world it needs a map for reference, but to build the map of the world the camera needs to know its position. One problem cannot be solved without the other so they must be estimated *simultaneously*. Visual SLAM started off using sparse image features with early work of Davison *et al*. [31, 32]. Feature-based SLAM represents the world as a sparse set of points in 3D space. It is very hard



(a) Sparse SLAM from PTAM [68] **(b)** Dense SLAM from Chapter 6

Figure 1.1: *The reconstructed map of a guitar from a sparse SLAM system (left) is unrecognisable. The dense SLAM system (right) provides a much richer and meaningful representation of the world.*

---

to recover any meaningful information about surfaces and geometry from this limited representation. Now, following increases in computational power, we have dense SLAM systems (such as [87, 88]) which make use of every pixel of the image. The maps created by such systems are much much richer than the feature-based maps (Fig. 1.1). From this representation we can easily recognise shapes and objects and it's not hard to see how a computer could use this information to start interacting with the world. In this thesis we attempt to further enhance these dense maps by capturing information about lighting and reflection of surfaces within the scene, leading to a truly rich and vivid knowledge of the surroundings.

Extending dense SLAM to capture lighting and reflectance can also help overcome some of the weaknesses of existing SLAM systems. Currently, almost all SLAM systems make the assumption that the world is Lambertian, that is, surface points have the same colour/intensity when viewed from any angle. They can often cope with some deviation from this assumption but too much can lead to tracking failures or inconsistencies in the map. Dense tracking methods are especially susceptible to this since they work on image intensities directly, feature-based methods are often more stable because the descriptors are designed to cope with illumination changes.. The recent work of Alismail *et al.* [3] makes a step towards robustifying dense RGB-D SLAM to intensity changes by using bit-planes. However, the algorithm underneath is still operating on a Lambertian assumption, we can only go so far with this

approach. By combining a full illumination model into the tracking and mapping pipeline it could be possible to not just become robust to non-Lambertian effects but actually *gain* from them. The tracking could become more accurate and the mapping will become richer. Most existing methods texture the map with some kind of average colour over all the view directions from which it has seen. By estimating illumination it could be possible to recover the true albedo and reflectance properties. This representation does not change with varying illumination so could lead to a SLAM system which can recognise a scene under any lighting conditions.

Bespoke hardware solutions already exist to help overcome some of the challenges of scene perception. Laser scanners provide very high quality 3D information but their size and cost make them prohibitive in many situations. The Microsoft Kinect sensor developed by Primesense provided a lower cost entry into 3D sensing. The accuracy was not comparable to laser scanners but the cost was low and the availability was widespread. The sensor works on a structured light technology which operates by triangulating an infra-red dot pattern emitted by a projector. Unfortunately the sensor fails when other sources of IR, such as sunlight, enter the scene. On the extreme high-end are the light stages at UC Berkeley and University of California [1]. These consist of an array of lights and cameras arranged in a sphere. By placing an object in the centre of the sphere the setup allows rapid image capture from multiple viewpoints and from multiple illumination configurations. This highly specialised system can capture incredibly accurate reflectance information about a surface. The stages have been used extensively in film to capture actor's faces enabling realistic virtual renderings in any scene under any illumination. If we truly want the highest quality reflectance information about a surface then a system such as this is surely the way to go. However, it is clearly not suitable for general unconstrained environments. Our focus is on a system which requires no modifications to the environment and so is purely passive. We focus on what can be achieved with a moving monocular camera, a low-cost sensor which is so widespread an estimated 2 billion people worldwide have access to one[2].

The problem of estimating lighting and reflection from images, also known as inverse rendering, is highly ill-conditioned, especially when the number of

[1] http://gl.ict.usc.edu/LightStages/

[2] In 2016 there are an estimated 2.08 billion smartphone users worldwide. Source: http://www.statista.com/statistics/330695

input images is small. Therefore, many works make assumptions and simplifications to make the problem tractable. Hara *et al.* [55] use just a single image as input and make the simplifying assumption of a single point light source. They jointly estimate the light source position and the diffuse and specular reflection components of the surface. Our approach is to stay away from methods which make such simplifying assumptions about the environment. We hope to develop a system which will work in general, unconstrained environmental so any assumption about the type or number of light sources is out of the question. By using a whole sequence of images from a live video stream we massively increase the volume of data making the problem more tractable so we don't need to make as many assumptions. Closer to our approach is the work of Nishino *et al.* [91]. They relax the point light source assumption and consider an illumination hemisphere. They are able to recover reflectance from a sparse set of images for objects with known geometry, obtained using a laser scanner, and use the information for novel view synthesis. While they use just 8 images in a offline optimisation we hope to develop a system which runs online and makes full use of all the data from a live video stream.

1.1 Applications

Augmented reality and robotic perception are two rapidly expanding fields with a heavy dependency on real-time computer vision. One important property they both share is that they need to have a high-quality model of the environment and they need it immediately. For augmented reality this is essential to ensure that virtual objects can be rendered consistently within a real scene, requiring geometry information for collision detection and illumination data for a realistic appearance utilising lighting, reflectance and shadows. For robotics this is needed so that a robot can interact with the environment, be it just for navigation and obstacle avoidance or more complex manipulations. These applications help drive the real-time approach of this thesis, and we hope the work can enhance these two overlapping, exciting areas of research.

Augmented reality generally consists of rendering a virtual or synthetic object into a real-world scene. The scene is therefore viewed through a special viewport, be it a smartphone screen, computer screen or a head mounted display and images need to be rendered in real-time. For a virtual object to seamlessly integrate with the rest of the scene the effects of the world illumination need

to be apparent on the object and the influence of the object on the scene, such as shadows, need to be accounted for. There have been various approches to capturing the illumination of the scene. The simplest involves placing a mirrored ball, known as a light-probe, into the scene; the reflections in the ball allow the camera to observe a hemisphere of illumination directions which can be used for object relighting. Some works make assumptions about the type of illumination; for example, Madsen and Lal [80] recover illumination for outdoor scenes in daylight from a single stereo depth image. They do this by assuming that all lighting comes from just the sky and the sun and that they are able to recover the direction vector to the sun. If the scene is of a smaller scale it could even be possible to directly view and reconstruct the light sources. This is the approach of Meilland *et al.* [82] in their High Dynamic Range (HDR) SLAM system. By using RGB-D SLAM they are able to create a 3D reconstruction of the environment combined with HDR texture obtained by fusing Low Dynamic Range (LDR) images captured at varying exposures. The 3D, HDR model can then be used to create virtual light-probes which can be used for object relighting and shadow generation. In this thesis we hope to infer lighting information without directly viewing the light-sources and to do this without using a depth sensor.

Most previous works in AR don't consider complex lighting effects of the surface on which the virtual object is placed. They will often apply shadows base on recovered illumination but there is no knowledge of the surfaces reflectance properties. We make a first step towards recovering the surface properties and show that we can use this to make a more realistic AR on specular surfaces by considering the effects of specular occlusion and reflections.

In the field of robotic perception it is still a challenge to interact with previously unseen environments. This hindrance is why most modern robotic systems operate in highly constrained environments or require significant human assistance. As well as the obvious knowledge of the scenes geometry required for physical interaction, there are more subtle, semantic cues which a robot needs to safely operate in other environments. Just observing the raw geometry of an object does not give the robot sufficient information to proceed. For example, a cup of liquid needs to be handled in a different way to a cylindrical block of wood, yet their geometries are very similar. There is a huge amount of research going on in the area of object detection and recognition; some of these work purely from RGB images [34, 69] and some are starting to make use of depth

as well [70]. However, object recognition can only help if the object is in the robots past knowledge. Additionally, extreme lighting effects can have a negative effect on recognition rate. One solution is to detect materials – if an object is not recognised but we can determine what it is made of then a robot can still make useful decisions about how to proceed. Material detection could be done using CNNs trained on labelled images, such as [8], or we can take a more data driven approach. If we have been able to capture information about lighting and reflectance of a surface then that information can be used to help determine materials. As a simple example, surface reflectance information could enable us to distinguish between a shiny, wood-effect plastic surface and a real wooden surface where a purely image based approach may fail. Additionally, we will see in this thesis that certain surfaces exhibit high-frequency geometric details which only become visible when specular reflections are observed. Knowledge of the lighting and reflectance information from various viewpoints could be vital to detect such materials.

Another application of this work is the scanning of objects/scenes for analysis, digital archiving or printing/replication. Many museums are starting to create digital archives of their collections, such as Smithsonian X 3D[3] and those involved in the Google Art Project[4]. By doing this, fragile artefacts which may have a limited lifespan can live on forever in digital form. Additionally, pieces of special interest can easily be shared amongst interested researchers as well as digital models being made available online for the general public. The scanning problem could be solved using specialised equipment, such as a system similar to the light stage mentioned earlier. However, there are still a number of situations where this is not an option and a monocular based system has its advantages. Cost is a large factor, being able to this with just a cheap monocular camera makes such a system much more accessible. Size is another; it is sometimes the case that scanning of objects is done at the site of archaeological digs so the equipment must be easy to transport and use in the field. We also believe that any scanning system is best off with real-time or online feedback and the ability to append a scan with more data so any gaps can be filled. This is one of the powerful features of real-time monocular SLAM – the incremental nature of the system is ideally suited to revisiting areas to increase the detail in the map. Whichever method is used to acquire the scans it is clear that the best representation for analysis and archiving should contain as much informa-

[3] http://3d.si.edu/

[4] https://www.google.com/culturalinstitute/

tion about the objects surface properties as possible. From a single image or a 3D scan with a single view-independent texture it is difficult to understand the materials and how they react with the illumination around them. By including the lighting and reflectance in the 3D model it could be possible for an observer to see how a piece looks under different lighting conditions and make a much more thorough, in-depth analysis of the piece without seeing it in person.

1.2 Enabling Technologies

The recent growth of general purpose compute power on commodity GPUs has really boosted the field of real-time computer vision. Combined with convenient GPU programming languages, such as CUDA and OpenCL, they really have made the work in this thesis, and much of the work this thesis builds upon, possible. GPU's are incredibly efficient at processing data in parallel and many image processing algorithms transfer to a GPU very easily.

Incredibly powerful and power hungry desktop GPUs (which are now reaching theoretical performance figures of 5,000,000,000,000+ FLoating-point Operations Per Second (FLOPS) on a single GPU[5]) are also complemented by increasingly more powerful and power efficient mobile GPUs. With the ubiquity of the modern smartphone a huge number of people have in their pocket a high-quality camera with processing power capable of running advanced computer vision algorithms. While this has enabled many technologies to run on current mobile hardware, the view of this thesis is that by doing research into algorithms which run in real-time on current desktop hardware it is expected that in the near future the compute power of mobile devices will also be sufficient.

1.3 Thesis structure

Following this introduction this thesis starts with a technical introduction to define some of the common tools and equations used throughout the rest of the thesis.

[5]NVIDIA GeForce GTX TITAN Black has 5120 GFLOPS of single-precision compute. Source: www.wikipedia.org

Our first step towards extending dense reconstruction to include lighting and reflectance information is outlined in Chapter 3. We introduce a novel approach to real-time acquisition of planar surface light-fields. Although we make use of a feature-based camera tracker, by assuming the observed surface is planar we are able easily convert this to a dense representation. During an initial capture stage we observe the surface from multiple viewpoints and record the observed irradiance of the surface from each direction. From the captured information we are able to infer lighting and reflectance information and we demonstrate how this can be used to generate shadows and specular occlusion for more realistic augmented reality on specular surfaces. We extend the work by using the captured data to recover high-frequency normal variations on the surface, further increasing the fidelity of the model.

Continuing the theme of lighting, shadow and reflectance we take a look at how these properties are manipulated by humans in classical sculpture. When sculpting in a medium of constant albedo our entire perception of the object is based on the lighting, shadow and reflectance information. In Chapter 4 we take a look at how sculptors manipulate geometry to enhance or mellow some of these effects in their work. We spoke to professional sculptors about techniques they use when creating human busts and interpret these to form a mathematical model and apply these enhancements to scanned busts. The term *sculptural stylization* is introduced to help define these types of modifications. The outcome often includes increasing angles between regions to increase curvature and, hence, increase the chances of catching a specularity, as well as protruding or recessing areas to create more shadow.

The next logical step in the story is to expand the surface light-field capture work of Chapter 3 to work with general 3D surfaces. The first hurdle to overcome was acquiring the 3D geometry from a moving monocular camera in real-time. In Chapter 6 we describe a system to perform such a task based on fusing depth-maps together incrementally using a surfel-based representation. We demonstrate how to track the camera via direct image alignment to the model and show how to make this robust to irradiance changes due to change of viewpoint. Prior to this in Chapter 5 we take a look at various methods to generate the required depth-maps. We first look at some existing approaches and then focus heavily on a state-of-the-art stereo system, PatchMatch Stereo [12]. While the system was originally presented as an offline method we do an in-depth performance analysis and demonstrate how it can

be modified and accelerated to work in a real-time system.

Finally, in Chapter 7 we conclude the thesis and discuss directions for future work extending what has been presented.

1.4 Publications

The work in this thesis resulted in the following publications:

- J. Jachnik, R. A. Newcombe, and A. J. Davison. Real-Time Surface Light-field Capture for Augmentation of Planar Specular Surfaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2012

- J. Jachnik, D. B. Goldman, L. Luo, and A. J. Davison. Interactive 3D Face Stylization Using Sculptural Abstraction. *arXiv preprint arXiv:1502.01954 [cs.GR]*, 2015

2

Technical Introduction

In this section we will introduce notation and definitions which are used extensively throughout this thesis.

2.1 Notation

Bold notation (e.g. $\mathbf{a}, \mathbf{b}$) will be used to represent vectors. A hat on top of a vector means that it has unit length ($\|\hat{\mathbf{n}}\| = 1$). $\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z$ are unit vectors in the x, y and z directions respectively. Together they form an orthonormal basis of $\mathbb{R}^3$. Matrices will use the following formatting: $\mathtt{A}, \mathtt{R}, \mathtt{M}$. The elements of a vector or matrix will use zero-based numbering:

$$\mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}, \quad \mathtt{M} = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}. \tag{2.1}$$

2.2 Camera Projection

Throughout this thesis there will be continuous references to camera projection and camera transformations. This section is a brief overview of the notation and terminology used.

2.2.1 Homogeneous Coordinates

Very often it will be useful to take a vector $\mathbf{x} \in \mathbb{R}^N$ and add a "1" to the end of it to make a vector in $\mathbb{R}^{N+1}$. We call this *homogeneous coordinates* and it is useful when working with projections and rigid transformations. We will use

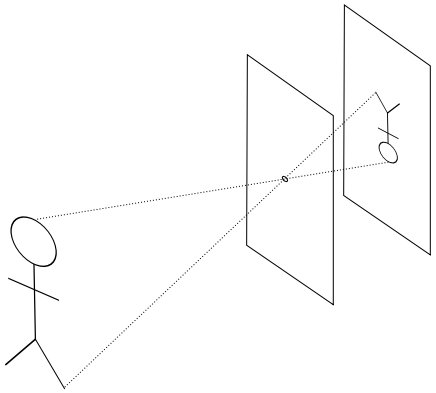


Figure 2.1: *A pinhole camera. Light rays travel in straight lines through an infinitesimally small hole (aperture). The result is an upside down view projected onto the image plane.*

---

the dot notation to represent such a transformation:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{2.2}$$

2.2.2 Camera Projection

In this thesis we will make use of the pinhole/perspective camera model. In a true pinhole camera (Fig. 2.1), the *focal length* is adjusted by varying the distance between the aperture and the image plane. In a conventional camera there is also a lens (or multiple lenses) which has an effect on the focal length. In order to accurately model the image formation we use four parameters to represent the projection of a pin-hole camera: two to represent the focal length in the horizontal (f_x) and vertical (f_y) directions and two to represent to position of the center of projection (u_0, v_0). These are combined into the camera *intrinsic* matrix $\mathtt{K}$:

$$\mathtt{K} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

In general, f_x and f_y will be roughly equal and the centre of projection will be roughly in the centre of the image. The focal length is related to field-of-view of the camera. A smaller focal length has a wider field-of-view.

Given a 3D point $\mathbf{x}$ in the camera frame of reference then the projection

of that point into image coordinates $\mathbf{u} = (u, v)^T$ is given by:

$$\mathbf{u} = \pi\left(\mathtt{K}\mathbf{x}\right), \tag{2.4}$$

where

$$\pi(\mathbf{x}) = \frac{1}{x_2}\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}. \tag{2.5}$$

The projection function $\pi(\cdot)$ is not injective (one-to-one) so it is not invertible. However, we can determine the direction of a ray from the camera using the inverse projection and if given the depth-map $D(\mathbf{u})$ of a pixel we can find the 3D point uniquely:

$$\mathbf{x} = D(\mathbf{u})\mathtt{K}^{-1}\dot{\mathbf{u}}. \tag{2.6}$$

This type of inverse projection will be used extensively in later sections.

2.2.3 Lens Distortion

Real-life cameras with lenses never quite live up to the perfect pin-hole camera model. The way in which the lenses bend the light causes distortions which must be included in our model. The most common and most significant type is radial distortion which is especially prevalent with wide-angle lenses (small focal length).

We use the term *radial* because the distortion acts in the radial direction from a given *centre of distortion* in the image (which we usually assume to be the centre of projection, $\mathbf{u}_0$). Let us note that any pixel $\mathbf{u}$ in an image can be written in terms of its direction from the camera center scaled by its radius from the centre:

$$\mathbf{u} = \frac{\mathbf{u} - \mathbf{u}_0}{|\mathbf{u} - \mathbf{u}_0|}|\mathbf{u} - \mathbf{u}_0| + \mathbf{u}_0 \tag{2.7}$$

$$= \frac{\mathbf{u} - \mathbf{u}_0}{|\mathbf{u} - \mathbf{u}_0|}r(\mathbf{u}) + \mathbf{u}_0. \tag{2.8}$$

Now, radial distortion is modelled by a non-linear function $R : \mathbb{R}_+ \to \mathbb{R}_+$ acting on the radius:

$$\mathbf{u}_d = \frac{\mathbf{u}_u - \mathbf{u}_0}{|\mathbf{u}_u - \mathbf{u}_0|}R\big(r(\mathbf{u}_u)\big) + \mathbf{u}_0, \tag{2.9}$$

where $\mathbf{u}_u$ is the coordinate in the undistorted image and $\mathbf{u}_d$ is the corresponding coordinate in the distorted image. Equation 2.9 is easily inverted:

$$\mathbf{u}_u = \frac{\mathbf{u}_d - \mathbf{u}_0}{|\mathbf{u}_d - \mathbf{u}_0|}R^{-1}\big(r(\mathbf{u}_d)\big) + \mathbf{u}_0, \tag{2.10}$$

which can be useful to create distorted images from undistorted input. For example, in augmented and mixed reality the virtual objects need to be distorted to match the real world view.

There are two common models we make use of for the radial distortion function:

ATAN Camera model This one parameter model gets its name from the arctangent function used to represent the distortion equation. It was introduced by Devernay and Faugeras [36] to model wide-angle and fisheye lenses. It benefits from having an analytical inverse:

$$R(r) = \frac{1}{\omega} \tan^{-1}\left(2r \tan\frac{\omega}{2}\right), \tag{2.11}$$

$$R^{-1}(r) = \frac{\tan(r\omega)}{2\tan\frac{\omega}{2}}. \tag{2.12}$$

Polynomial model This uses a polynomial to approximate the inverse distortion function.

$$R^{-1}(r) = r\left(1 + k_1 r^2 + k_2 r^4 + ... + k_n r^{2n}\right). \tag{2.13}$$

When $n = 1$ the distortion function $R(r)$ can be found by analytically solving a cubic equation. For $n \geq 2$ the inverse of the polynomial cannot be found analytically so an iterative method must be used.

Instead of using the full non-linear model of perspective projection with lens distortion in all of our computations we choose to remove the distortion from the input images as a pre-processing step. Now we only have to deal with the relatively simple perspective projection functions which simplifies much of the computation and speeds up our algorithms.

Regardless of the distortion function, fast computation of the undistorted image is best done using a lookup table. We create lookup table $\mathbf{L}$ such that the undistorted image I_u can be computed from the distorted image I_d using the mapping:

$$I_u(\mathbf{u}) = I_d\big(\mathbf{L}(\mathbf{u})\big). \tag{2.14}$$

Since $\mathbf{L}(\mathbf{u})$ does not lie at an integer pixel position we use bilinear interpolation to evaluate $I_d\big(\mathbf{L}(\mathbf{u})\big)$. The lookup table can be precomputed and then undis-

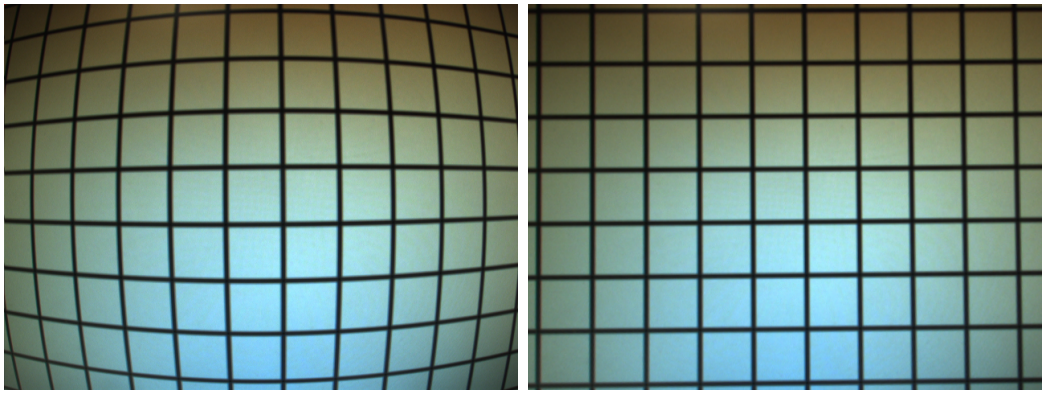


Figure 2.2: *Left: An image of a square grid captured with a wide angle lens showing significant radial distortion. Right: The same image after radial distortion has been successfully removed using the ATAN model.*

---

tortion can be computed very efficiently on the GPU. The undistorted image can now be treated as a traditional pin-hole camera image.

2.3 Transformations

2.3.1 Rigid Transformations

Rigid transformations are called as such because they keep space rigid. I.e. there is no scaling or warping of the space. Rigid transformations are most often used to describe the position and motion of objects in space we will often use them to describe the pose of a camera.

There are two components to build a rigid transformation in 3-dimensional space ($\mathbb{R}^3$): translation (3 degrees of freedom) and rotation (3 degrees of freedom). When dealing the the pose of a camera, it is defined by its position and orientation relative to some other coordinate frame. We use a right-handed coordinate system where, in image space, x goes from left to right, y goes from top to bottom and z points forwards from the camera.

A 3-dimensional rotation can be represented as a matrix from the Special Orthogonal group $\mathrm{SO}(3) = \{\mathtt{R} \in \mathrm{GL}(3) | \mathtt{R}^T\mathtt{R} = \mathtt{R}\mathtt{R}^T = \mathtt{I}, det\mathtt{R} = 1\}$. In text, this is the group of 3×3 invertible matrices such that the inverse of a matrix is its transpose and its determinant is equal to 1. A matrix $\mathtt{R} \in \mathrm{SO}(3)$ rotates all

points $\mathbf{v} \in \mathbb{R}^3$ about the origin $(0,0,0)^T$ by the action of matrix multiplication: $\mathtt{R}\mathbf{v}$.

It is often useful to calculate the rotation matrix from one direction vector to another. The rotation matrix from vector $\hat{\mathbf{v}}_0$ to vector $\hat{\mathbf{v}}_1$ (both unit vectors) is given by:

$$\mathtt{R}_{\hat{\mathbf{v}}_0 \to \hat{\mathbf{v}}_1} = \begin{bmatrix} \hat{\mathbf{v}}_1 & \hat{\mathbf{n}} & \hat{\mathbf{n}} \times \hat{\mathbf{v}}_1 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{v}}_0 & \hat{\mathbf{n}} & \hat{\mathbf{n}} \times \hat{\mathbf{v}}_0 \end{bmatrix}^T, \text{where} \quad \hat{\mathbf{n}} = \frac{\hat{\mathbf{v}}_0 \times \hat{\mathbf{v}}_1}{\|\hat{\mathbf{v}}_0 \times \hat{\mathbf{v}}_1\|}. \quad (2.15)$$

Note that this formula is only valid if $\|\hat{\mathbf{v}}_0 \times \hat{\mathbf{v}}_1\| \neq 0$. This matrix could also be derived from the Rodrigues' formula which computes the rotation matrix from the rotation axis $\hat{\mathbf{n}}$ and rotation angle $\theta = \cos^{-1}(\hat{\mathbf{v}}_0 \cdot \hat{\mathbf{v}}_1)$:

$$\mathtt{R} = \cos\theta \mathtt{I} + \sin\theta [\hat{\mathbf{n}}]_\times + (1 - \cos\theta)\hat{\mathbf{n}} \otimes \hat{\mathbf{n}}, \quad (2.16)$$

where

$$[\hat{\mathbf{n}}]_\times = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}, \quad \hat{\mathbf{n}} \otimes \hat{\mathbf{n}} = \begin{bmatrix} n_x^2 & n_x n_y & n_x n_z \\ n_x n_y & n_y^2 & n_y n_z \\ n_x n_z & n_y n_z & n_z^2 \end{bmatrix}. \quad (2.17)$$

A 3-dimensional translation (position) is simply a vector $t \in \mathbb{R}^3$ and transforms points $\mathbf{v} \in \mathbb{R}^3$ by addition: $\mathbf{v} + \mathbf{t}$.

A 3-dimensional rigid transformation is an element of the Special Euclidean group $\mathrm{SE}(3) = \mathrm{SO}(3) \ltimes \mathbb{R}^3$, where $\ltimes$ represents a semi-direct product [60]. An element $\mathtt{T}$ of $\mathrm{SE}(3)$ consists of a rotation $\mathtt{R} \in \mathrm{SO}(3)$ and a translation $\mathbf{t} \in \mathbb{R}^3$. The easiest way to represent and use these is the 4×4 homogeneous matrix representation:

$$\mathtt{T} = \begin{bmatrix} \mathtt{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \quad (2.18)$$

although, for a more compact notation we will often write Eq. 2.18 as

$$\mathtt{T} = [\,\mathtt{R} \,|\, \mathbf{t}\,]. \quad (2.19)$$

Composition of two transformations is done by matrix multiplication:

$$\mathtt{T}^a\mathtt{T}^b = \begin{bmatrix} \mathtt{R}^a & \mathbf{t}^a \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathtt{R}^b & \mathbf{t}^b \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathtt{R}^a\mathtt{R}^b & \mathtt{R}^a\mathbf{t}^b + \mathbf{t}^a \\ \mathbf{0} & 1 \end{bmatrix}. \quad (2.20)$$

The inverse of a 3D rigid transformation can be derived using the multiplication rules above and setting the right-hand side to the identity matrix:

$$\mathtt{T}^{-1} = \begin{bmatrix} \mathtt{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathtt{R}^T & -\mathtt{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.21}$$

To transform a 3D point $\mathbf{x}$ we use the homogeneous representation and then it is simply matrix-vector multiplication:

$$\mathtt{T}\dot{\mathbf{x}} = \begin{bmatrix} \mathtt{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathtt{R}\mathbf{x} + \mathbf{t} \\ 1 \end{bmatrix}. \tag{2.22}$$

Alternatively, we may drop the "dot" notation a simply write $\mathtt{T}\mathbf{x} = \mathtt{R}\mathbf{x} + \mathbf{t}$.

In many cases there will be more than one transformation to keep track of and each of them transforms between two specific coordinate frames. To make it easier to keep track we will use superscripts to denote in which space a point/vector lies and between which spaces a transformation operates. For example, let $\mathbf{x}^w$ be a point in the world frame of reference and let $\mathtt{T}^{cw}$ be the transformation from world coordinates to camera coordinates. Now, the point $\mathbf{x}^c$ in the camera frame of reference is given by $\dot{\mathbf{x}}^c = \mathtt{T}^{cw}\dot{\mathbf{x}}^w$ (note the use of homogeneous coordinates). If we are given a direction vector $\mathbf{v}^w$ (e.g. a normal vector) then we just need to apply a rotation to change coordinate frames: $\mathbf{v}^c = \mathtt{R}^{cw}\mathbf{v}^w$.

Note To represent the pose of a camera we will use $\mathtt{T}^{wc}$ (rather than $\mathtt{T}^{cw}$). This is convenient because in the camera frame of reference the center of the camera lies at the origin $\mathbf{0} = (0, 0, 0)^T$ and, hence, in world coordinates the camera origin lies at $\mathtt{T}^{wc}\mathbf{0} = \mathbf{t}^{wc}$.

2.3.2 Affine transformations

Affine transformations allow a much broader range of transformations than SE(3) such as scaling, reflections and shearing. Using the homogeneous representation, as used in Eq. 2.18, a general affine transformation can be represented by a linear transformation $\mathtt{A}$ and a translation $\mathbf{b}$:

$$\begin{bmatrix} \mathtt{A} & \mathbf{b} \\ 0 & 1 \end{bmatrix}. \tag{2.23}$$

The difference from a rigid transformation is that $\mathtt{A}$ can be any 3×3 real-valued matrix. Therefore, the space of affine transformations includes all rigid transformations. An affine transformation is invertible if and only if the matrix $\mathtt{A}$ is invertible, that is $\mathtt{A} \in GL(3)$. The set of invertible affine transformations forms the semi-direct product Lie group $GL(3) \ltimes \mathbb{R}^3$.

2.3.3 Lie Algebras and Lie Group Generators

In some cases we may wish to compute derivatives with respect to a transformation (e.g. a rotation matrix). Given, that a rotation matrix has only 3 degrees of freedom, it is not obvious how to do this. We make use of the Lie algebra and exponential map as a way of parametrizing a rotation with only 3 variables and computing derivatives with respect to these variables.

First recall that the exponential function of a matrix is written as:

$$\exp(\mathtt{X}) = \sum_{k=0}^{\infty} \frac{\mathtt{X}^k}{k!}. \tag{2.24}$$

By using the exponential map it is possible to show that for any anti-symmetric matrix $\mathtt{M}$ ($\mathtt{M}^T + \mathtt{M}{=}0$) then $\exp(\mathtt{M}) \in \mathrm{SO}(3)$. An anti-symmetric matrix $\mathtt{M}$ has only 3 degrees of freedom and can be written as:

$$\mathtt{M} = \begin{bmatrix} 0 & m_2 & -m_1 \\ -m_2 & 0 & m_0 \\ m_1 & -m_0 & 0 \end{bmatrix}. \tag{2.25}$$

We denote the space of anti-symmetric matrices as $\mathfrak{so}(3)$ and it is known as the Lie algebra of SO(3). Equation 2.25 reveals a natural representation of an element of $\mathfrak{so}(3)$ as a vector $\mathbf{m} = (m_0, m_1, m_2) \in \mathbb{R}^3$. We introduce the wide hat operator to go from the vector representation to the matrix representation in Eq. 2.25: $\widehat{\mathbf{m}} = \mathtt{M}$. Note that this vector representation also allows us to interpret matrix multiplication as a vector cross product: $\mathtt{M}\mathbf{v} = \mathbf{m} \times \mathbf{v}$.

Now, given a rotation matrix $\mathtt{R} = \exp(\mathtt{M})$ we can attempt to compute derivatives. When working with scalars we are familiar with the derivative of the exponential function:

$$\frac{d}{dx} \exp(f(x)) = \exp(f(x)) f'(x), \tag{2.26}$$

which makes use of the chain rule and the fact that x and $f'(x)$ commute. Matrices don't always commute so we can't use the same equation. To overcome this, we can look at the derivative when $\mathtt{R} = \exp(\mathtt{M})$ is close to the identity matrix, that is, $\mathtt{M}$ is close to the zero matrix. In this case we can just look at the first few terms of the Taylor expansion of $\exp(\mathtt{M})$:

$$\mathtt{R} = \mathtt{I} + \mathtt{M} + \frac{\mathtt{M}^2}{2} + ... \tag{2.27}$$

We can compute the derivative of the first-order approximation to be:

$$\frac{d}{dx}\mathtt{R}(x) \approx \frac{d\mathtt{M}}{dx}. \tag{2.28}$$

Now, suppose $\mathtt{M} = \widehat{\mathbf{m}}$ and $x = m_i$, computing the derivatives we end up with the *generators* of the Lie group SO(3):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.29}$$

The generators form a basis for the Lie algebra $\mathfrak{so}(3)$. In theory, any basis could be used as generators. This particular basis arises from the vector representation we introduced in Eq. 2.25.

The same ideas can be applied to the Lie group SE(3) and its corresponding Lie algebra $\mathfrak{se}(3)$. The generators of SE(3) are listed in Appendix B.

We will see in later sections that this first-order approximation of the derivative is useful to do optimisation with Lie groups. For a more in-depth derivation and explanation of Lie groups, Lie algebras and the exponential map see [67].

2.4 Multi-view Geometry

Stereo vision relies on the fact that given multiple views of an object it is possible to find correspondences between views and use this to infer geometry. In this section we will give an overview of the geometry involved and the transformations needed.

2.4.1 Transforming between two cameras

To transform a pixel in one camera to a pixel in another camera we must use a combination of projection, inverse projection and rigid transformations. Let $\mathbf{u}^l$ be a pixel in the *live* camera with depth $D(\mathbf{u}^l)$. This pixel can be un-projected using Eq. 2.6 to obtain a vertex in the frame of reference of the live camera. We can then transform this vertex into the frame of reference of the *reference* camera and find the corresponding pixel $\mathbf{u}_r$ in this image using Eq. 2.4. The full transformation can be written as follows:

$$\dot{\mathbf{u}}^r = \pi\left(\mathtt{K}\mathtt{T}^{rl} D(\mathbf{u}^l)\mathtt{K}^{-1}\dot{\mathbf{u}}^l\right) \tag{2.30a}$$

$$= \pi\left(\mathtt{K}\mathtt{R}^{rl} D(\mathbf{u}^l)\mathtt{K}^{-1}\dot{\mathbf{u}}^l + \mathtt{K}\mathbf{t}^{rl}\right) \tag{2.30b}$$

$$= \pi\left(\mathtt{K}\mathtt{R}^{rl}\mathtt{K}^{-1}\dot{\mathbf{u}}^l + \frac{1}{D(\mathbf{u}^l)}\mathtt{K}\mathbf{t}^{rl}\right), \tag{2.30c}$$

where $\mathtt{T}_{rl}$ is the rigid transformation between the coordinate frames of each image. Equation 2.30c arises due to the fact that the π operator is invariant to scaling: $\pi(\alpha\mathbf{x}) = \pi(\mathbf{x}), \forall\alpha \neq 0$. As discussed in Section 2.4.2 it is more convenient to use inverse depth when doing stereo and this form of the warp can be more computationally efficient by removing the need to constantly convert between inverse depth and depth.

2.4.2 Inverse Depth

If we vary $D(\mathbf{u}_l)$ in Eq. 2.30 for a fixed $\mathbf{u}_l$ we see that $\mathbf{u}_r$ moves along a straight line in the reference image. This line is known as the *epipolar line*. When performing stereo matching we would like to sample depth values $D^i(\mathbf{u}_l)$ so that the corresponding points $\mathbf{u}_r^i$ are evenly sampled along the epipolar line. An even sampling in $D^i(\mathbf{u}_r)$ does not correspond to an even sampling along the epipolar line. Instead, we use *inverse* depth, which does.

The fact that inverse depth is directly related to the pixel distance along the epipolar line is also convenient as a way to measure the accuracy at which the depth can be resolved, as formalized in Section 2.4.3.

2.4.3 (Inverse) Depth Uncertainty

When generating depth-maps from stereo vision it is useful to have an idea of the uncertainty of that estimate. For example, we will use it for fusing multiple depth maps in Chapter 6. In this section we will actually describe the uncertainty in inverse depth and then show how to convert this to uncertainty in depth.

Our model of inverse depth uncertainty is based on the derivations by Engel *et al*. [41]. They describe three terms which contribute to the uncertainty.

The geometric disparity error is derived from possible errors in camera calibration and pose estimation.

$$\sigma_g^2 \propto \frac{1}{\hat{\mathbf{g}} \cdot \hat{\mathbf{l}}}, \tag{2.31}$$

where $\hat{\mathbf{g}}$ is the normalized image gradient and $\hat{\mathbf{l}}$ is the normalized epipolar line direction. Intuitively this shows that if the camera motion is perpendicular to the image gradient then the uncertainty is high.

The photometric disparity error is based on possible noise in the intensity values of an image.

$$\sigma_p^2 \propto \frac{1}{\mathbf{g} \cdot \hat{\mathbf{l}}}, \tag{2.32}$$

where $\mathbf{g}$ is the non-normalized image gradient. I.e. the denominator is the image gradient along the epipolar line. We see that in order to have a low uncertainty there must be significant gradient along the epipolar line.

The final uncertainty term is dependent on the conversion from pixels to inverse depth. We approximate the inverse depth to be proportional to the disparity. We introduce the ratio α which is the length of the inverse depth range divided by the length of the corresponding epipolar line. If the epipolar line is longer, there will be more resolution over which to recover an accurate inverse depth and, hence, the uncertainty goes down.

The three uncertainty terms are combined in [41] as follows:

$$\sigma^2 = \alpha^2 \left(\sigma_g^2 + \sigma_p^2\right). \tag{2.33}$$

When using multiple images in the stereo data-term we will combine them

with an inverse sum:

$$\frac{1}{\sigma^2} = \sum_i \frac{1}{\sigma_i^2}, \tag{2.34}$$

where each individual σ_i^2 is given be Eq. 2.33. Using this approach is equivalent to combining several measurements at the same inverse depth within a Kalman filter style update.

For speed and ease of implementation we compute all quantities in the reference image. The image gradient is independent of the other frames but there will be a different epipolar line direction and length for each frame used in the data-term.

Depth Uncertainty Let I represent inverse depth. Given the current inverse depth estimate $\bar{I}$ and uncertainty σ^2 we can write that $I = \bar{I} + \eta$ where η is a normally distributed random variable with mean 0 and variance σ^2. We then write the depth, D, in these terms:

$$\begin{aligned} D &= \frac{1}{\bar{I} + \eta} && \text{(2.35a)} \\ &= \frac{1}{\bar{I}} \frac{1}{1 + \frac{\eta}{\bar{I}}} && \text{(2.35b)} \\ &= \frac{1}{\bar{I}} \left(1 - \frac{\eta}{\bar{I}} + O\left(\frac{\eta^2}{\bar{I}^2}\right)\right) && \text{(2.35c)} \\ &\approx \bar{D} - \bar{D}^2 \eta. && \text{(2.35d)} \end{aligned}$$

Hence, using standard properties of the variance, we can say that D is distributed with mean $\bar{D} = \bar{I}^{-1}$ and variance $\sigma^2 \bar{D}^4$.

2.4.4 Normals from depth-map

We often need to compute normals from a depth-map. To do this, we first create the vertex map $\mathbf{v}(\mathbf{u})$ using Eq. 2.6. Then, the normal at pixel $\mathbf{u}$ is given by:

$$\hat{\mathbf{n}}(\mathbf{u}) = \frac{\mathbf{d}_x \times \mathbf{d}_y}{|\mathbf{d}_x \times \mathbf{d}_y|}, \tag{2.36}$$

where

$$\begin{aligned} \mathbf{d}_x &= \mathbf{v}(u+1, v) - \mathbf{v}(u-1, v), && \text{(2.37a)} \\ \mathbf{d}_y &= \mathbf{v}(u, v+1) - \mathbf{v}(u, v-1). && \text{(2.37b)} \end{aligned}$$

2.5 Image Denoising

Here we give a brief introduction to variational image denoising methods. These algorithms are used in this thesis to denoise depth-maps.

2.5.1 ROF Denoising

Total Variation (TV) image denoising using the methods outlined in [52, 100] is becoming a commonly used tool in real-time computer vision due to its highly parallel and efficient GPU implementation. *Total Variation* refers to the L_1 norm on the gradient in the energy (Eq. 2.38). This norm is preferable to an L_2 norm because it disfavours noise and has no bias against discontinuities [100], better modelling real images. Rudin, Osher and Fatemi (ROF) [109] define an image denoising formulation using TV which can be represented as minimizing the following energy:

$$E_{ROF} = ||\nabla u||_1 + \frac{\lambda}{2}||u-g||_2^2 = \sum_x |\nabla u(x)| + \frac{\lambda}{2}\big(u(x)-g(x)\big)^2. \tag{2.38}$$

We wish to find the image $u(x)$ which minimizes this energy given a noisy image $g(x)$. The first term in the energy is the regularisation term, this is to generate a smooth, noise-free image. The second term is the data term, this makes sure the generated image is close to the original, noisy image. The parameter λ determines the ratio of the data to the regularisation. A smaller λ means the image will be smoothed more, a larger λ means that the image will fit the data better - and hence show more of the original noise. The result is an image denoising framework which preserves edges, due to the L_1 norm (Fig. 2.3).

ROF proposed the following gradient descent method for minimizing the energy given above:

$$u^{n+1} = u^n - dt\left[-\nabla\cdot\left(\frac{\nabla u}{|\nabla u|}\right) + \lambda(u-g)\right], \tag{2.39}$$

with a given step size dt (on which there is an upper bound to ensure convergence). However, the equations become degenerate as $|\nabla u(x)| \to 0$. To overcome this problem we formulate the dual problem [100]. To do this we re-write the Total Variation term, $|\nabla u|$, using the Legendre-Fenchel transform, as detailed in [52]:

$$|\nabla u| = \max\big(\langle \nabla u, p\rangle, \text{ for } |p| \leq 1\big). \tag{2.40}$$

Figure 2.3: *An image (left) had Gaussian noise added to it (middle). We then use the ROF denoising model using a primal-dual approach to remove the noise (right).*

This transforms the energy minimisation problem:

$$\min_u \sum_x |\nabla u(x)| + \frac{\lambda}{2}\big(u(x) - g(x)\big)^2 \tag{2.41}$$

into a saddle point problem:

$$\min_u \max_p \sum_x \big\langle \nabla u(x), p(x) \big\rangle - \delta_{|p|\leq 1} + \frac{\lambda}{2}\big(u(x) - g(x)\big)^2, \tag{2.42}$$

where the indicator function $\delta_{|p|\leq 1}$ is defined as follows:

$$\delta_{|p|\leq 1} = \begin{cases} 0, & |p| \leq 1, \\ \infty, & \text{otherwise.} \end{cases} \tag{2.43}$$

To solve this saddle point problem we alternate between iterations of gradient ascent to maximise in p and gradient descent to minimise in u:

$$\frac{p^{n+1} - p^n}{\sigma} = \frac{\partial E_{ROF}}{\partial p} = \nabla u^n, \text{ such that } |p| \leq 1, \tag{2.44a}$$

$$\frac{u^n - u^{n+1}}{\tau} = \frac{\partial E_{ROF}}{\partial u} = -\operatorname{div} p^{n+1} - \lambda g, \tag{2.44b}$$

which results in the following iterative update scheme:

$$p^{n+1} = \frac{p^n + \sigma \nabla u^n}{\max(1, |p^n + \sigma \nabla u^n|)}, \tag{2.45a}$$

$$u^{n+1} = \frac{u^n + \tau \operatorname{div} p^{n+1} + \tau\lambda(u^{n+1} - g)}{1 + \tau\lambda}, \tag{2.45b}$$

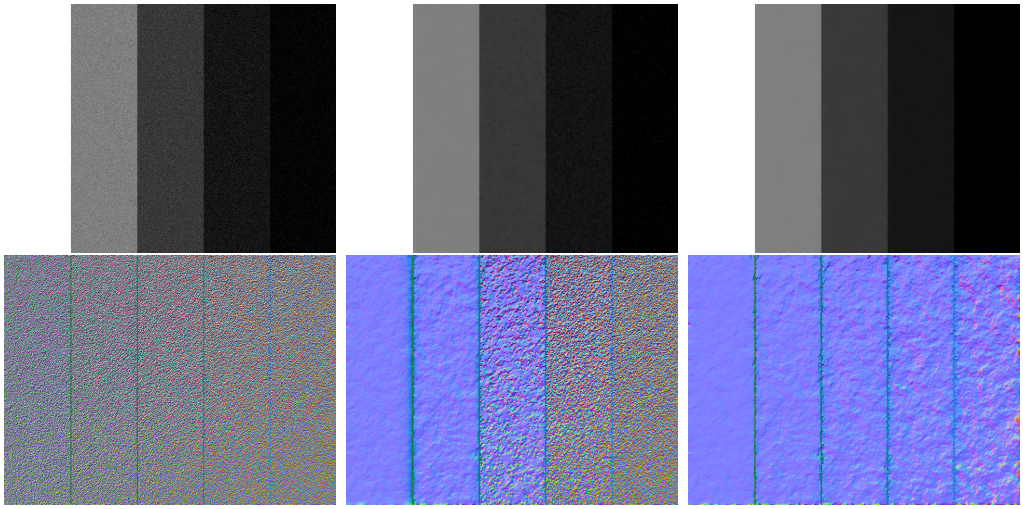


Figure 2.4: *Gaussian noise is added to a depth-map with variance as defined in Section 2.4.3 (left). We use variational denoising parametrised in depth (middle) and inverse depth (right). Note that the inverse depth parametrisation has a more even effect over the range of depths. The top row represents inverse depth, the bottom row represents the normals computed from the depth-map.*

---

where the normalisation term in Eq. 2.45a is so that p remains within the bounds of the unit ball $|p| \leq 1$.

Figure 2.4 shows the effect of ROF regularisation applied to a depth map, as it will be used in this thesis. We formulate the problem in both depth and inverse-depth to demonstrate that the ROF model fits the data better when the noise is well modelled by a Gaussian distribution (as it is for inverse-depth in the figure).

2.5.2 General Form

The standard image denoising problem consists of finding the denoised image u^* from a noisy input image g. It can be formulated as en energy minimisation problem:

$$u^* = \arg\min_u \sum_{x \in \Omega} \mathcal{R}\big(\nabla u(x)\big) + \lambda \mathcal{D}\big(u(x) - g(x)\big), \tag{2.46}$$

where $\mathcal{R}$ is a *regularization* term to ensure smoothness in the result, $\mathcal{D}$ is the *data* term to ensure the solution still represents the original noisy input and λ is a parameter used to adjust the trade-off between smoothness and data fidelity.

	$\mathcal{R}(\mathbf{x})$	$\mathcal{R}^*(\mathbf{y})$
L_2	$\frac{1}{2}\left\|\mathbf{x}\right\|^2$	$\frac{1}{2}\left\|\mathbf{y}\right\|^2$
L_1	$\left\|\mathbf{x}\right\|$	$\delta_{\left\|\mathbf{y}\right\|\leq 1}$
Huber	$\left\|\mathbf{x}\right\|_\alpha$	$\frac{\alpha}{2}\left\|\mathbf{y}\right\|^2 + \delta_{\left\|\mathbf{y}\right\|\leq 1}$

Table 2.1: *Some functions and their conjugates. See Appendix C for the definition of the Huber norm and a derivation of its conjugate.*

There are several common choices for both of these functions, some examples of which can be seen in Table 2.1. For example, in the ROF method we use an L_2 norm on the data term and L_1 regularisation. We may decide to use a more robust norm, such as L_1, on the data term if we know that there are outliers in the data, as is often the case with stereo generated depth-maps. An L_1 data term has the interesting property of being able to remove structures of a certain scale, specified by λ, independent of contrast. We refer you to [100] for a more in-depth analysis of the impact of this term.

The general approach to solving is the same as with the ROF model. We convert the minimisation problem into a saddle point problem by re-writing $\mathcal{R}(x)$ in terms of its conjugate function:

$$\mathcal{R}(x) = \max_{y\in\mathcal{Y}} \langle x, y\rangle - \mathcal{R}^*(y) \tag{2.47}$$

where the Legendre-Fenchel transform is used to compute the conjugate function:

$$\mathcal{R}^*(y) = \max_{x\in\mathcal{X}} \langle x, y\rangle - \mathcal{R}(x) \tag{2.48}$$

Some examples can be seen in Table 2.1 with derivations of these and others in [52]. The final saddle point problem becomes:

$$\min_u \max_p \sum_{x\in\Omega} \langle \nabla u(x), p(x)\rangle - \mathcal{R}^*\big(p(x)\big) + \lambda\mathcal{D}\big(u(x) - g(x)\big), \tag{2.49}$$

which is solved via gradient ascent/descent as demonstrated in the previous section.

Chambolle and Pock [17] formalize and prove the convergence of this class of algorithms and provide values for the step sizes to give optimal convergence speed.

2.5.3 A Useful Tip

We use finite differences to compute the gradient and divergence operators and one may naively decide to use forward differences, say, for both. This does not work. There is a fixed relationship between the gradient and divergence operator defined by the following equation:

$$\langle \nabla u, p \rangle = \langle u, -\operatorname{div} p \rangle, \tag{2.50}$$

and this relation must be obeyed. Therefore by specifying that we use backward difference for the gradient, say, then the finite difference divergence operator is defined by the above equation.

We will demonstrate this for the 1-dimensional case. The easiest way is see this is to use the matrix notation for the finite difference operator. We stack all pixels into a single vector $\mathbf{u}$ so that the gradient operator is a matrix multiplication:

$$\nabla \mathbf{u} = \mathtt{D}\mathbf{u} \tag{2.51}$$

$$= \begin{bmatrix} 0 & 0 & 0 & & 0 \\ -1 & 1 & 0 & & 0 \\ 0 & -1 & 1 & & 0 \\ & & & & \\ 0 & 0 & & -1 & 1 \end{bmatrix} \mathbf{u}. \tag{2.52}$$

Then, using Eq. 2.50, we can write:

$$\langle \nabla \mathbf{u}, \mathbf{p} \rangle = \langle \mathtt{D}\mathbf{u}, \mathbf{p} \rangle \tag{2.53}$$

$$= \langle \mathbf{u}, \mathtt{D}^T \mathbf{p} \rangle \tag{2.54}$$

$$= -\langle \mathbf{u}, \mathrm{div}\mathbf{p} \rangle, \tag{2.55}$$

and, hence, the finite difference operator for div can be represented by the matrix $-\mathtt{D}^T$. If you work it through it turns out that the divergence must be a forward difference. In the case of a two-dimensional image the divergence is

given by:

$$(\text{div}\mathbf{p})_{\text{i,j}} = \begin{cases} p^0_{1,j} & \text{if } i = 0 \\ p^0_{i+1,j} - p^0_{i,j} & \text{if } 0 < i < N-1 \\ -p^0_{N-1,j} & \text{if } i = N-1 \end{cases} + \begin{cases} p^1_{i,1} & \text{if } j = 0 \\ p^1_{i,j+1} - p^1_{i,j} & \text{if } 0 < j < M-1 \\ -p^1_{i,M-1} & \text{if } j = M-1 \end{cases} \quad (2.56)$$

2.6 Light-fields

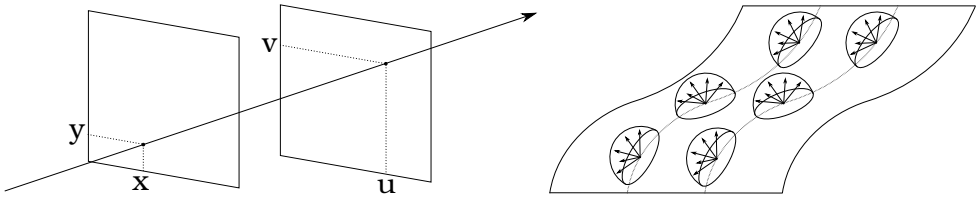
A light-field (also know as a 'Lumigraph' [48]) is a 4-dimensional function representing the intensity of light travelling along all rays in free-space. It is derived from the 5-dimensional plenoptic function. The plenoptic function of Adelson and Bergen [1] represents the intensity of light travelling in every direction (2-dimensions) at every point in space (3-dimensions). The light-field comes about with the assumption that the intensity of light does not change when travelling along a ray through free-space, hence the plenoptic function is reduced by one dimension. The light-field contains all the information needed to render a scene from any position and any orientation within the free-space constraints.

Note In these definitions we are assuming that the scene is static. For dynamic scenes we must add an extra dimension to the definition of the plenoptic function and light-field to represent time.

2.6.1 Light-field Representations

While the 5D plenoptic function has an obvious representation on $S^2 \times \mathbb{R}^3$ (a unit sphere at every point in space), the removal of one dimension is not obvious. This leads to various different representations of the light-field.

Two-plane representation Fig. 2.5a. By specifying a coordinate on each of two parallel planes, a line joining the point on one plane to the point on the other now has a position and direction. This representation is most com-



(a) Two-plane **(b)** Surface

Figure 2.5: *Two common light-field representations. The two-plane representation is the common output from light-field cameras and does not need any knowledge of the scene's geometry. The surface representation is useful when estimating surface properties such as reflection and illumination.*

---

mon, possibly due to the close relationship to the geometry of light-field cameras (Section 2.6.2). This format can also be considered as a set of traditional camera images with the camera centres positioned on the vertices of a 2D grid; one plane represents the camera centres, the other represents the image plane. This is the format used by the Stanford Light Field Archive [118] and is also known as the 'light slab' representation [74].

Surface light-fields Fig. 2.5b. Only when the geometry of a surface is known can we use this representation. This is a useful representation because it represents the irradiance emanating from the actual surface and, hence, is useful in determining lighting and reflectance information. A discrete set of points is evenly sampled from the surface (2D) and a sphere (or hemisphere aligned with the surface normal) is used at each point to represent the ray direction. Wood *et al*. [131] refer to these spheres as *lumispheres* (luminance + sphere) and this is the terminology used in this thesis.

Unstructured A set of images from traditional cameras along with the camera poses also forms an *unstructured* light-field representation. Davis *et al*. [30] used this representation to interactively capture and render light-fields with a handheld camera.

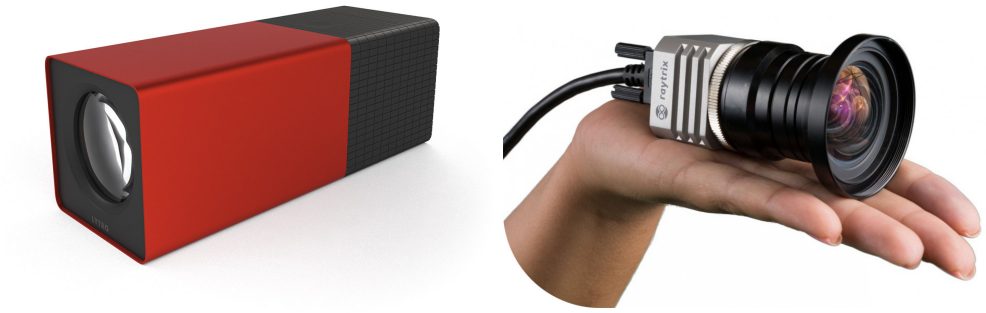


Figure 2.6: *The Lytro light-field camera (left, source: www.lytro.com) and the Raytrix C42 light-field camera (right, source: www.raytrix.de).*

2.6.2 Light-field Cameras

A traditional pin-hole camera can be thought of as sampling the light-field at a single point in space and each pixel corresponds to a different ray direction. A light-field camera tends to capture the light passing through many points in space and in multiple directions. On the simplest level this can be just an array of traditional cameras, such as the Stanford light-field array [118]. Single sensor light-field cameras [79, 103] (see Fig. 2.6) consist of a microlens array between the main lens and the sensor of a traditional camera [89], as seen in Fig. 2.7. This can be thought of as an array of tiny cameras observing the virtual image of the scene created by the main lens. One of the drawbacks of light-field cameras is a much lower spatial resolution because of sampling a 4-dimensional function on a 2-dimensional image sensor. However, there are some advantages such as refocusing after capture, depth estimation, and the ability to synthesize images from varying viewpoints. The interactive gallery on the Lytro website [79] is the best place to see examples of these effects.

2.6.3 Hemisphere Discretization

In this thesis we will make use of the surface light-field representation. To do so, we need a way of sampling the space evenly. This involves discretising over all 4 dimensions of the light-field; the two dimensions representing position on the surface and another two dimensions to represent the viewing direction. We will make use of planar surface light-fields, in which case, evenly sampling points from the surface is trivial; we just create a uniform grid over the plane. For

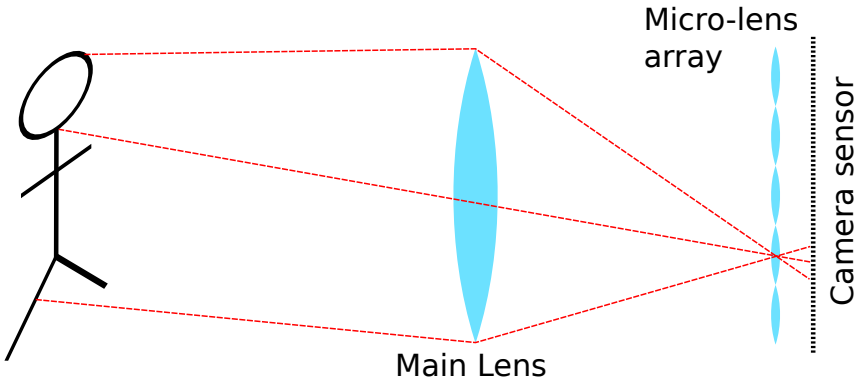


Figure 2.7: *A light-field camera differs from a traditional camera by the addition of a micro-lens array between the main lens and the sensor.*

---

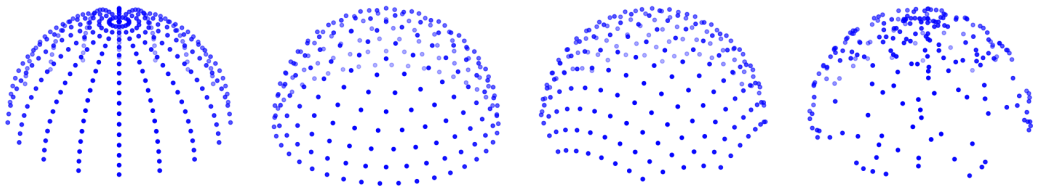
non-planar surfaces the easiest discretisation method is to perform isotropic remeshing using a method such as [123]. This creates a regular mesh with roughly equal sized equilateral triangles. We then use the resulting vertices as the sample points.

Sampling from the space of viewing directions on the lumisphere is not as easy. To represent the radiance function $I(\boldsymbol{\omega})$ on the lumisphere, we discretize the surface of a hemisphere and store a value for each discrete point. The samples on the hemisphere need to be as evenly spaced as possible to efficiently represent the distribution of outgoing light.

Many efficient methods have been explored to distribute points evenly on a sphere. For example, Poisson disc sampling [15, 26] can be adapted to work on a sphere and generate very evenly sampled points[6]. However, random sampling (Fig. 2.8d) is not well suited to interpolation between points, something desired in a surface light-field lumisphere structure.

Using polar coordinates (θ, ϕ) and sampling uniformly in each independently leads to a sampling bias towards the poles of the sphere/hemisphere (Fig. 2.8a). Wood *et al.* [131] use a sampling strategy based on subdividing an octahedron and projecting onto the sphere. We propose a novel discretization based on [131] but allowing a much finer trade-off between resolution and memory requirement; the surface light-field is a four-dimensional structure and

[6]Some excellent examples of random sampling on a sphere:
https://www.jasondavies.com/maps/random-points/



(a) × bad **(b)** ✓ good **(c)** ✓ good **(d)** × bad

Figure 2.8: *Comparison of hemisphere discretization techniques. (a) represents a uniform sampling in θ and ϕ; the points are clearly not evenly distributed. (b) is the method we use, the sampling is uniform. (c) is the distribution obtained by projecting the hemisphere to the plane and sampling on a 2D grid. (d) is a random sampling from a uniform distribution for θ and ϕ.*

---

to store this at an adequate resolution puts tight bounds on memory usage.

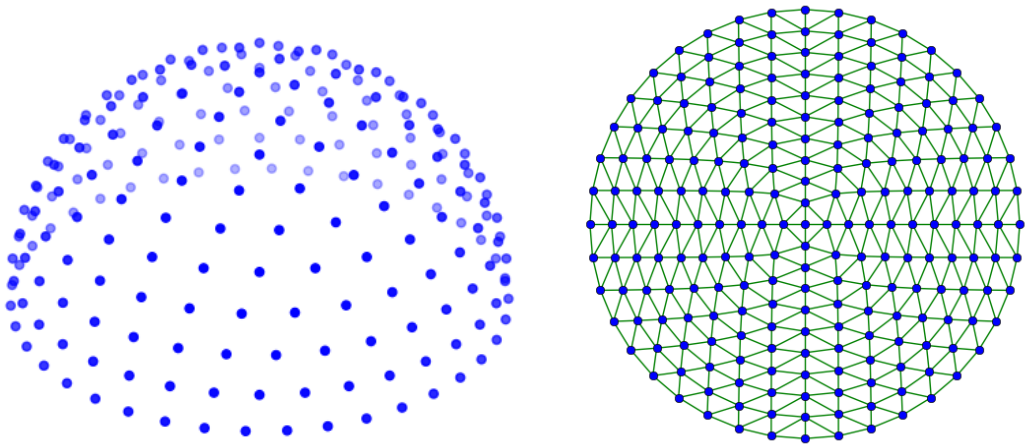
We will define our discretization method on the unit hemisphere. However, it easily generalizes to the unit sphere via symmetry. Let θ be the polar angle and ϕ be the azimuth angle. We define a single integer parameter n which defines a number of discrete values $\{\theta_0, ..., \theta_n\}$ evenly distributed in the range $[0, \frac{\pi}{2}]$. This gives us a set of concentric circles on the surface of the hemisphere. Next, level set θ_i is evenly subdivided into $4i$ values for ϕ on the range $[0, 2\pi]$. Hence, the set of discrete points on the hemisphere can be summarized as follows:

$$(\theta, \phi) \in \mathfrak{L}_n = \left\{ \left(\tfrac{i\pi}{2n}, \tfrac{2j\pi}{i}\right) \;\middle|\; j = 0, ..., i-1, \quad i = 0, ..., n \right\} \tag{2.57a}$$

$$|\mathfrak{L}_n| = 2n(n+1) + 1, \tag{2.57b}$$

with $\phi = 0$ when $i = 0$. Figure 2.8b shows that the spread of the discrete points is relatively even across the surface of the hemisphere. The subdivision exhibits a triangular structure, as can be seen in Fig. 2.9b. To do interpolation between points we find the corresponding triangle and use barycentric coordinates to weight the values at the 3 vertices in the triangle, as described in Section 2.6.4.

Another possible way to discretize a hemisphere is to project that hemisphere onto a plane and then discretize the plane using 2D grid (Fig. 2.8c). The simplest way to project a point (θ, ϕ) to the plane is to directly map these to 2D polar coordinates with radius θ and angle ϕ. All points on the hemisphere now map to a circle of radius $\frac{\pi}{2}$ which we then discretize on a uniform $N \times N$ grid. Bilinear interpolation can be used to lookup values at non-discrete loca-



(a) (b)

Figure 2.9: *Discretization of a hemisphere with $n = 10$. On the left is a 3D view. Note that the points are evenly spread. On the right is the projection onto the plane along with the triangulation used for interpolation.*

---

tions. This discretization scheme is convenient because it allows us to represent a lumisphere as a normal image, ignoring the points which lie outside of the circle. We use this method in Section 3.13 for ease of implementation when computing gradients on the lumisphere. This method does not transfer well to the sphere due to the difficulties of projection of a sphere onto a plane while preserving area.

2.6.4 Lumisphere Interpolation

As in many cases where a continuous space has been discretized, we may wish to perform interpolation on the discrete data to get a value of the light-field at an arbitrary location. To do this we use a natural triangulation of the space (Fig. 2.9b) and use barycentric coordinates to set the interpolation weights. This is the natural choice when we recall that each quadrant of the hemisphere is mapped to a triangle. We have created the natural triangulation on the triangle and mapped it to each quadrant of the hemisphere.

Barycentric coordinates are the most natural way to interpolate values within a triangle and are defined as follows: Given a triangle with vertices $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$, then an point $\mathbf{p}$ inside the triangle can be represented as a unique

linear combination of the three vertices such that:

$$\mathbf{p} = \lambda_0\mathbf{p}_0 + \lambda_1\mathbf{p}_1 + \lambda_2\mathbf{p}_2, \text{ where } \quad \lambda_0 + \lambda_1 + \lambda_2 = 1, \quad \lambda_0, \lambda_1, \lambda_2 \geq 0 \tag{2.58}$$

Note that if one of the barycentric coefficients is zero then $\mathbf{p}$ lies on an edge and if two coefficients are zero then $\mathbf{p}$ lies on a corner.

To compute barycentric coordinates we substitute $\lambda_2 = 1 - \lambda_0 - \lambda_1$ into the barycentric equation to yield:

$$\mathbf{p} = \lambda_0\mathbf{p}_0 + \lambda_1\mathbf{p}_1 + (1 - \lambda_0 - \lambda_1)\mathbf{p}_2 \tag{2.59}$$

Rearranging this we get:

$$\mathbf{p} = \lambda_0(\mathbf{p}_0 - \mathbf{p}_2) + \lambda_1(\mathbf{p}_1 - \mathbf{p}_2) + \mathbf{p}_2 \tag{2.60}$$

Since $\mathbf{p} \in \mathbb{R}^2$ (or $\mathbb{R}^3$) this is a set of simultaneous equations which can be solved to find λ_0, λ_1. We can then compute $\lambda_2 = 1 - \lambda_0 - \lambda_1$. Note that in $\mathbb{R}^3$ there are 3 equations for 2 unknowns, the equations will only be consistent if $\mathbf{p}$ lies in the same plane as the triangle. We can then choose any 2 of the equations to solve for λ_0, λ_1. Barycentric coordinates can also be used to determine whether a point lies inside or outside of a triangle. If a point is inside a triangle then all the constraints in Eq. 2.58 are satisfied. However, we can solve Eq. 2.60 for any point in the same plane as the triangle. If the point lies outside of the triangle then at least one of $\lambda_0, \lambda_1, \lambda_2$ will be negative.

When sampling viewing directions, the 3D coordinate of a point lies on the surface of the unit sphere, not on a planar surface described by the triangulation. Hence, in our method, once the relevant triangle has been found (addressed below) we project the point onto the containing plane to correctly compute barycentric coordinates.

Given a random point (θ', ϕ') on a lumisphere it is not trivial how to find the triangle in which it lies. We first find θ_+ and θ_- to be the closest discrete values to the polar angle θ'. We can then easily find 4 bounding vertices $(\theta_+, \phi_{++}), (\theta_+, \phi_{+-}), (\theta_-, \phi_{-+}), (\theta_-, \phi_{--})$ which are at discrete values of the the azimuth angle closest to ϕ'. These 4 points define 2 triangles using the structure in Fig. 2.9b. We project the point onto both planes defined by the triangles, compute 2 sets of barycentric coordinates and use these to determine in which triangle the point actually lies.

2.7 Photometric Calibration and HDR

When capturing an image with a digital camera we may assume that the value of each pixel is a measure of the *irradiance*, that is, the light incident on the camera sensor. It is then common to assume that the scene's *radiance* (the light emitted my the scene) is proportional to the irradiance measured. This is how we can get estimates of the amount of light in a scene and determine quantities like reflectivity and the environments illumination. However, the image formation process can contain many non-linearities which means that pixel values do not map to irradiance in a simple, direct way.

Debevec and Malik [33] show that radiance may map non-linearly to the pixel values given by a camera. In order to get an estimate of the scenes radiance, Debevec and Malik propose a method to find the Camera Response Function (CRF), which is the characteristic function of a specific camera mapping pixel values to irradiance.

The method for finding the CRF is simple, and only needs to be done once for a fixed set of camera settings. The camera position is fixed and images are captured at multiple, known shutter speeds. Pixels are then selected which cover the full dynamic and colour range of the images. The values of these pixels, with their known shutter speeds, are combined into an optimisation problem. Solving this yields the CRF, for which a lookup table can be built for efficient conversion between pixels and irradiance.

Grossberg and Nayar[51] analyse the theoretical space of all camera response functions. They have collected a database of CRF's, the Database of Response Functions (DoRF), for real world cameras. Then, by applying constraints of the theoretical space of CRF's, developed an Emiprical Model of Response (EMoR) and show that this fits well to the cameras in the database. EMoR allows a CRF to be represented via a low parameter model; most curves are well represented by a 3 parameter model.

A current drawback of these models is that all camera settings (aperture, gain, white-balance, etc.) except shutter speed must remain constant for the response function to hold true. If any of these settings were to be changed, a new response function would need to be found. A more global, parametric model which could account for changes in these other parameters would be of

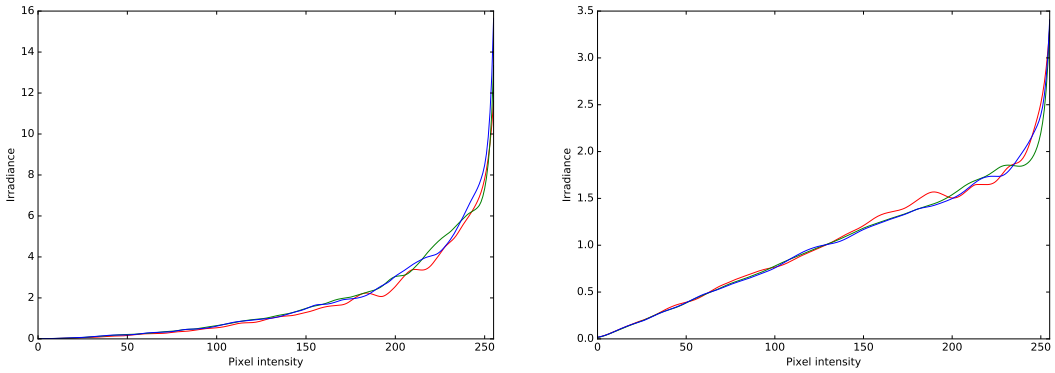


Figure 2.10: *The response function the a consumer DLSR (left) tends to be non-linear due to post-processing. The response function of image sensors tends to be linear (until saturation) and this is seen in the response function of a machine vision camera (right).*

---

great help.

Interestingly, the response function of a typical camera sensor is almost totally linear. However, most modern digital cameras deliberately apply a non-linear mapping to the sensors output to get more aesthetically pleasing images. The reason for this is that photographic film has a non-linear response function [33] and many of the modern digital cameras are trying to simulate this response because it creates more pleasing images. Cameras built for machine vision purposes tend to leave out this non-linear processing because it is unnecessary and often unwanted. Therefore, their response function is almost linear to match the sensor response. Figure 2.10 shows the response function computed using the method of [33] for an Olympus E-520 (a consumer DSLR) and a Point Grey Flea2 (a machine vision camera).

When the response function of a camera is known we can combine images of the same scene at multiple exposures to create a High Dynamic Range (HDR) image. In a traditional Low Dynamic Range (LDR) image there are often under-exposed (black) or over-exposed (saturated) regions where there is a loss of detail and there is not a one-to-one mapping between the pixel value and irradiance. To recover the irradiance in such areas we need to change the shutter speed of the camera. By capturing multiple images at varying shutter speeds and converting each one to irradiance we can combine all the information together to get a true estimate of the irradiance over a dynamic range much

higher than a single image can capture.

2.8 Phong Illumination

The Phong illumination model (or Phong reflection model) was proposed by Bui Tuong Phong in 1975 [96] and is widely used in computed graphics to this day. It provides a local model of the illumination of a surface by considering three different components: ambient, diffuse and specular. The ambient term represents low level scattered light that is present in the entire scene. The diffuse term is a major factor in matt surfaces. It's intensity is dependent on the angle between the surface normal and the direction to a light source. The specular component is dependent on the viewpoint and is what gives surfaces a glossy appearance.

Under Phong illumination, the intensity I_p of a surface point p is given by the following equation:

$$I_p = \underbrace{k_a i_a + \sum_{m \in lights} k_d \left(\hat{\mathbf{l}}_m \cdot \hat{\mathbf{n}}\right) i_{m,d}}_{\text{view-independent}} + \underbrace{k_s \left(\hat{\mathbf{r}}_m \cdot \hat{\mathbf{v}}\right)^{\alpha} i_{m,s}}_{\text{view-dependent}}, \tag{2.61}$$

where,

- k_a, k_d, k_s are the surface's ambient, diffuse, and specular reflection constants, respectively,
- α is the shininess constant for the surface, a large value is more shiny,
- i_a is the ambient illumination,
- $i_{m,d}, i_{m,s}$ are the intensitys of the diffuse and specular illumination from light m, respectively,
- $\hat{\mathbf{l}}_m$ is the unit vector directed from the surface point p to light m,
- $\hat{\mathbf{n}}$ is the unit surface normal,
- $\hat{\mathbf{v}}$ is the unit vector directed from p to the viewpoint (camera),
- $\hat{\mathbf{r}}_m = 2\big(\hat{\mathbf{l}}_m \cdot \hat{\mathbf{n}}\big)\hat{\mathbf{n}} - \hat{\mathbf{l}}_m$.

We will make use of this illumination model in later sections.

2.9 Dense Image Registration

The work in this section is based on the work of Lucas and Kanade [78]. We give an overview of the general method of dense image registration here because it is used a number of times in this thesis.

Suppose we have an image I' which was obtained by warping some image I:

$$I'(\mathbf{u}) = I\big(\mathbf{w}(\mathbf{u};\mathbf{p})\big), \tag{2.62}$$

where $\mathbf{p}$ are parameters of the warping function $\mathbf{w}$. For example, the warp could be due to camera rotation. In this case the parameters $\mathbf{p}$ define a rotation matrix and the warp is given by:

$$\mathbf{w}(\mathbf{u};\mathbf{p}) = \pi\left(\mathtt{KR}(\mathbf{p})\mathtt{K}^{-1}\dot{\mathbf{u}}\right). \tag{2.63}$$

In this section we will give an overview of how to recover the parameters $\mathbf{p}$ of the warp function given the images I and I'. We do this by minimizing a cost function based on the error between pixels:

$$E(\mathbf{p}) = \sum_{\mathbf{u}\in\Omega}\Big(I'(\mathbf{u}) - I\big(\mathbf{w}(\mathbf{u};\mathbf{p})\big)\Big)^2 \tag{2.64a}$$

$$= \frac{1}{2}\mathbf{e}(\mathbf{p})^T\mathbf{e}(\mathbf{p}), \tag{2.64b}$$

where we define the residual vector $\mathbf{e}(\mathbf{p})$ such that $e_i(\mathbf{u}) = I'(\mathbf{u}_i) - I(\mathbf{w}(\mathbf{u}_i;\mathbf{p}))$ and $\mathbf{u}_i$ is the i'th pixel.

To solve this type of equation we first approximate $\mathbf{e}(\mathbf{p})$ with a first order Taylor series about $\mathbf{p} = \mathbf{0}$:

$$\hat{\mathbf{e}}(\mathbf{p}) = \mathbf{e}(\mathbf{0}) + \nabla\mathbf{e}(\mathbf{0})\mathbf{p}. \tag{2.65}$$

The new approximate energy becomes

$$\hat{E}(\mathbf{p}) = \frac{1}{2}\hat{\mathbf{e}}(\mathbf{p})^T\hat{\mathbf{e}}(\mathbf{p}). \tag{2.66}$$

We seek a minimum of this energy in the usual way be setting its derivative equal to zero:

$$\begin{aligned}\mathbf{0} &= \nabla \hat{E}(\mathbf{p}) && (2.67)\\ &= \nabla \mathbf{e}(\mathbf{0})^T \hat{\mathbf{e}}(\mathbf{p}) && (2.68)\\ &= \nabla \mathbf{e}(\mathbf{0})^T \left[\mathbf{e}(\mathbf{0}) + \nabla \mathbf{e}(\mathbf{0})\mathbf{p}\right] && (2.69)\\ &= \mathtt{J}^T \left(\mathbf{e}(\mathbf{0}) + \mathtt{J}\mathbf{p}\right), && (2.70)\end{aligned}$$

where we have introduced the *Jacobian* matrix $\mathtt{J} = \nabla \mathbf{e}(\mathbf{0})$. We see that the solution is obtained when $\mathtt{J}\mathbf{p} = -\mathbf{e}(\mathbf{0})$. If $\mathbf{p} \in \mathbb{R}^N$ and there are M pixels in an image then this is a system of M equations for N unknowns. In the camera rotation example above $N = 3$ and M is on the order of tens of thousands of pixels. Typically it is the case that $M >> N$. We therefore have a massively overdetermined system which we solve in a least squares framework by using the normal equations:

$$\mathtt{J}^T \mathtt{J}\mathbf{p} = -\mathtt{J}^T \mathbf{e}(\mathbf{0}), \tag{2.71}$$

which is solved by inverting the matrix $\mathtt{J}^T\mathtt{J}$:

$$\mathbf{p} = -\left(\mathtt{J}^T \mathtt{J}\right)^{-1} \mathtt{J}\mathbf{e}(\mathbf{0}). \tag{2.72}$$

In practice we don't need to compute the full Jacobian matrix $\mathtt{J}$ (which is a huge $M \times N$ matrix) and instead just compute $\mathtt{J}^T\mathtt{J} \in \mathbb{R}^{N \times N}$ and $\mathtt{J}\mathbf{e}(\mathbf{0}) \in \mathbb{R}^N$.

Due to the first order approximation in Eq. 2.65 this approach will only be valid for $\mathbf{p} \approx \mathbf{0}$. Therefore we redefine the warp function after each iteration so that the parameters $\mathbf{p}$ just represent a small perturbation and our assumptions always hold. For example, the rotation warp in Eq. 2.63 becomes

$$\mathbf{w}^{(k)}(\mathbf{u}; \mathbf{p}) = \pi\left(\mathtt{K}\hat{\mathtt{R}}^{(k)}\mathtt{R}(\mathbf{p})\mathtt{K}^{-1}\dot{\mathbf{u}}\right), \tag{2.73}$$

where k is the iteration count. We then solve the normal equations using this warp function to obtain $\hat{\mathbf{p}}^{(k)}$ and update the estimate of the rotation matrix:

$$\hat{\mathtt{R}}^{(k+1)} = \hat{\mathtt{R}}^{(k)}\mathtt{R}(\hat{\mathbf{p}}^{(k)}). \tag{2.74}$$

Using this approach, we only ever solve for $\mathbf{p} \approx \mathbf{0}$ which is consistent with our first order approximation in Eq. 2.65. Additionally, we only need to compute derivatives of $\mathtt{R}(\mathbf{p})$ about $\mathbf{p} = \mathbf{0}$, which are simply the generators of the Lie group, as stated in Section 2.3.3.

2.9.1 Weighting

In some cases we wish to apply weights to different pixels by considering the following energy:

$$E(\mathbf{p}) = \sum_{\mathbf{u}\in\Omega} w(\mathbf{u})\Big(I'(\mathbf{u}) - I\big(\mathbf{w}(\mathbf{u};\mathbf{p})\big)\Big)^2 \tag{2.75a}$$

$$= \frac{1}{2}\mathbf{e}(\mathbf{p})^T \mathtt{W}\mathbf{e}(\mathbf{p}), \tag{2.75b}$$

where $\mathtt{W}$ is a diagonal matrix, $\mathtt{W}_{ii} = w(\mathbf{u}_i)$, and $w(\mathbf{u}_i) \geq 0, \forall i$. To solve this system we just make a small modification to the normal equations:

$$\mathtt{J}^T\mathtt{W}\mathtt{J}\mathbf{p} = -\mathtt{J}^T\mathtt{W}\mathbf{e}(\mathbf{0}). \tag{2.76}$$

2.9.2 M-Estimators

The quadratic term on the residuals in Eqs. 2.64 and 2.75 works extremely well on clean data or when noise in the data causes the residuals to be well approximated by a zero-mean Gaussian probability function. However, a small number of extreme outliers can have a detrimental effect on the result. We can make use of M-estimators to reduce the impact of such extreme outliers and make the system more robust.

The M-estimator problem is formulated by replacing the sum of squares energy with another function ρ:

$$E(\mathbf{p}) = \sum_{\mathbf{u}\in\Omega} \rho\big(\mathbf{e}_\mathbf{u}(\mathbf{p})\big), \tag{2.77}$$

where $\rho(x)$ is required to be symmetric, positive-definite and have a unique minimum at zero. In the standard least squares setting $\rho(x) = \frac{1}{2}x^2$.

Instead of minimising Eq. 2.77 directly, we transform it to an iteratively re-weighted least squares problem. To do so we introduce the *influence function*, $\psi(x)$, and the *weight function*, $w(x)$:

$$\psi(x) = \frac{d\rho(x)}{dx}, \quad w(x) = \frac{\psi(x)}{x}. \tag{2.78}$$

It can be shown that the minimising Eq. 2.77 is equivalent to solving the following iteratively re-weighted least squares problem:

$$E(\mathbf{p}) = \sum_{\mathbf{u}\in\Omega} w\left(e_\mathbf{u}(\mathbf{p}^{(k-1)})\right)\left(e_\mathbf{u}(\mathbf{p}^{(k)})\right)^2, \tag{2.79}$$

where the bracketed superscript denotes the iteration number. Weights are computed from the result of the previous iteration and the system is solved as in Section 2.9.1.

2.10 Meshes

A polygon mesh consists of *vertices*, *edges*, and *faces*. In this thesis we will only consider triangular meshes (where each face is a triangle). A vertex is a point $\mathbf{p} \in \mathbb{R}^3$ and the geometry of the mesh is defined by the positions of its vertices. The edges and faces define the connectivity of a mesh. An edge joins two vertices and, for a triangle mesh, a face is formed from three vertices and the three edges connecting those vertices.

The meshes we consider in this thesis will be *manifold*. The rigorous mathematical definition of a manifold is complicated so we won't include it here. However, in the scope of triangular meshes we can interpret manifold as meaning that each edge is incident to either one or two faces and the faces surrounding each vertex form either a closed fan or open fan.

In general, a manifold triangle mesh is stored and represented as a list of vertex positions and a list of triplets of indices which form faces. Each triplet references the three vertices which form the corners of the face. Edges don't need to be stored because they are directly inferred from the edges of the faces. In a manifold mesh all edges must be incident to at least one face, so all edges are defined this way.

2.10.1 Mesh Laplacian

Given a mesh consisting of vertices $\{\mathbf{p}_i\}_{i=0}^{n-1}$, let $j \in N(i)$ denote that vertex $\mathbf{p}_j$ is directly connected to $\mathbf{p}_i$ by a single edge. $N(i)$ is the neighbourhood of vertex i. We define the mesh Laplacian as follows:

$$L(\mathbf{p}_i) = \frac{1}{\sum_{j \in N(i)} w_{ij}} \sum_{j \in N(i)} w_{ij} \left(\mathbf{p}_j - \mathbf{p}_i\right). \tag{2.80}$$

We can also define the mesh Laplacian which acts on a function ϕ defined on the mesh:

$$L_\phi(\mathbf{p}_i) = \frac{1}{\sum_{j\in N(i)} w_{ij}} \sum_{j\in N(i)} w_{ij}\big(\phi(\mathbf{p}_j) - \phi(\mathbf{p}_i)\big). \tag{2.81}$$

There are various choices for the set of weights w_{ij}. The two most common weighting schemes are uniform weights ($w_{ij} = 1, \forall i, j$) and cotangent weights (defined in Section 2.10.2). In Appendix D we show how this definition of the mesh Laplacian relates to the common definition of the Laplacian as the divergence of the gradient operator.

The mesh Laplacian is often used to smooth meshes and functions defined on meshes. Smoothing is performed in an iterative fashion. To smooth the vertices directly we use the following update scheme:

$$\mathbf{p}_i^{(k+1)} = \mathbf{p}_i^{(k)} + L^{(k)}(\mathbf{p}_i^{(k)}), \tag{2.82}$$

and the equivalent iterative update for functions on meshes is:

$$\phi^{(k+1)}(\mathbf{p}_i) = \phi^{(k)}(\mathbf{p}_i) + L_\phi^{(k)}(\mathbf{p}_i). \tag{2.83}$$

The mesh Laplacian can also be represented as a discrete linear operator (i.e. a matrix). If we let $\mathbf{P} = (\mathbf{p}_0, ..., \mathbf{p}_{n-1})^T$ be the matrix containing all the points $\mathbf{p}_i$ then the Laplacian matrix $\mathtt{L}$ is defined such that:

$$L(\mathbf{p}_i) = [\mathtt{L}\mathbf{P}]_i \,. \tag{2.84}$$

Using Eq. 2.80 we see that

$$\mathtt{L}_{ij} = \begin{cases} \frac{w_{ij}}{\sum_k w_{ik}}, & i \neq j, \\ -1, & i = j, \end{cases} \tag{2.85}$$

where we set $w_{ij} = 0$ if $\mathbf{p}_i$ and $\mathbf{p}_j$ are not connected by an edge.

2.10.2 Cotangent weights

Cotangent weights are defined on the internal edges of a triangular mesh. They are often used instead of uniform weights when computing the mesh Laplacian because the effects of smoothing is less dependent on the triangulation structure [35]. For example, consider the mesh in Fig. 2.11a. All vertices lie in the

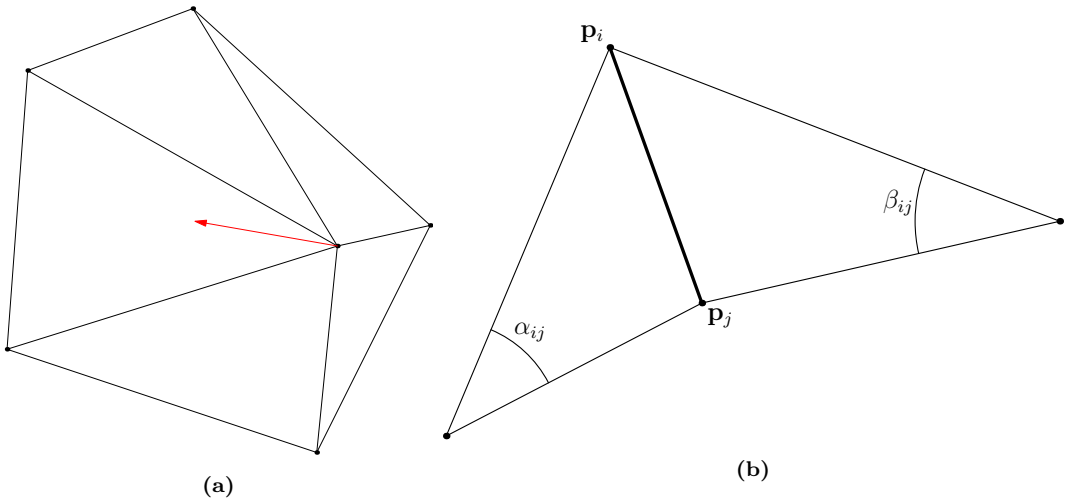


Figure 2.11: *(a) Using uniform weights for Laplacian smoothing causes the vertices to move to a more regular triangular structure despite the fact the mesh is already as smooth as it can be. (b) Cotangent weights are computed from the angles of the two triangles sharing an interior edge of the mesh. When these weights are used for smoothing the triangulation structure is unaffected.*

---

plane so it is a smooth as it can be. However, when smoothing with uniform weights the vertices are moved to make a more uniform triangulation structure. With cotangent weights the positions of the vertices stay fixed, a useful property for meshes with irregular triangulation.

Given an internal edge from vertex $\mathbf{p}_i$ to vertex $\mathbf{p}_j$, the cotangent weight w_{ij} is given by:

$$w_{ij} = \frac{1}{2}\left(\cot \alpha_{ij} + \cot \beta_{ij}\right) \tag{2.86}$$

where α_{ij} and β_{ij} are the angles opposite the edge of the two triangles that share the edge (these triangles always exist provided it is an internal edge), as can be seen in Fig. 2.11b.

2.11 Datasets

In this section we will give a brief overview of some datasets we will use to evaluate stereo and 3D reconstruction.

Figure 2.12: *Left: a view of the geometry from within Blender. Middle: A sample render from the dataset. Right: A depth map from the dataset.*

2.11.1 Basic Blender Sequence

This synthetic dataset was created in Blender[7] and designed to have simple geometry and be free of properties which cause problems for many stereo algorithms. Therefore we do not include any view-dependent lighting effects and make sure the surfaces have plenty of texture so that matching is not ambiguous (see Fig. 2.12). The idea is that this will test the maximum achievable accuracy of the stereo algorithm in absence of these complicated situations. It was also important to include surfaces which were not fronto-parallel to the camera so that we can test the accuracy of reconstructing slanted planes.

Each frame of the dataset consists of an RGB image in PNG format, 32-bit floating-point depth in OpenEXR format and a text file containing the pose of the camera, $\mathtt{T}_{wc}$. The focal length of the lens is 8mm with a sensor size of 6.16×4.62mm. We render at VGA resolution (640×480) leading to a focal length of 831.169 pixels and the camera centre at $(320, 240)$.

We render two trajectories from the dataset. The first is a straight trajectory with constant velocity in the x-direction used to test the performance of stereo algorithms at varying baselines. The second is a circular trajectory which we use to evaluate the full reconstruction system. For example, we can check that the solution converges over time by running it in a loop.

[7] https://www.blender.org/

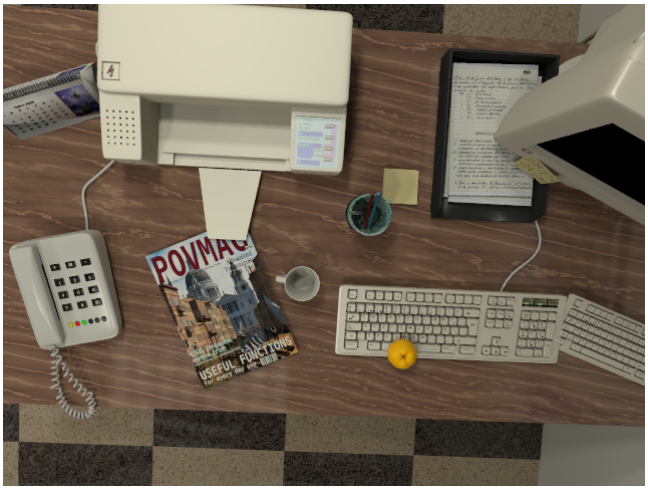


Figure 2.13: *An example image from a POVRay generated sequence of views over a desk.*

2.11.2 POVRay Generated Desk Sequence

This dataset was realistic generated using the POV-Ray tool [93]. We use the same 3D model as used in by Handa *et al*. in [53]. This high-quality rendering pipeline provides non-Lambertian lighting effects, reflections and exhibits textureless regions which can cause further problems for passive vision systems. An example image can be seen in Fig. 2.13.

The trajectory was obtained by logging the tracked camera poses from PTAM [68] during hand-held camera motion over a desk.

We note that we *must* add at least small amount of image noise to the raytraced output of POV-Ray; In texture-less regions there are subtle changes of intensity of just a single quantisation step. The position of these steps form a wave which moves within the texture-less region as the camera moves. Whilst the human eye cannot notice the change, a stereo algorithm looking at pixel intensity differences will notice the change and it can cause artefacts.

3

Real-time Surface Light-field Capture

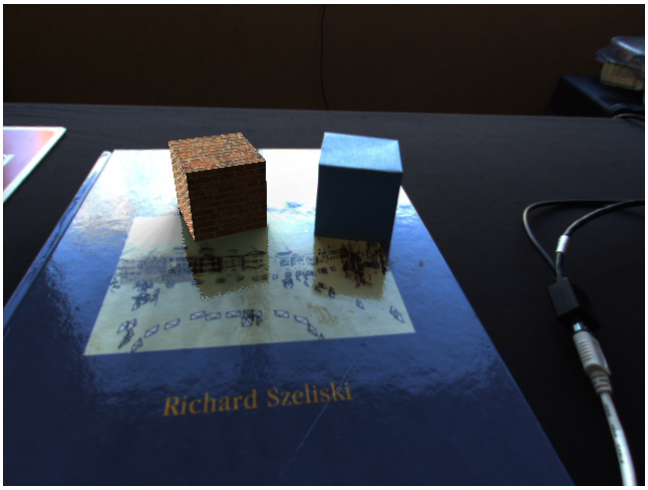


Figure 3.1: *A virtual cube (left) is placed on a planar specular surface next to a real cube (right). We use the captured surface light field to estimate illumination, compute shadows, and, most importantly, occlude the specularities visible on the surface.*

In this chapter we present a system for the real-time acquisition of surface light-fields on planar surfaces. A calibrated camera is tracked relative to a known planar surface, images are acquired from various viewpoints and stored in a surface light-field data structure. We then go on to show how to use this information for light position estimation, environment map estimation, bump-map estimation and to create more realistic AR on specular surfaces.

Shiny/specular surfaces are quite common in indoor environments; for example, books, tables, ceramic and plastic surfaces. Despite this, the realism of current augmented reality systems on such surfaces is limited. Many systems estimate the environment illumination, both with [92] and without [80, 82, 92] light probes. The estimated lighting is used to correctly illuminate the object as well as generate shadows for virtual objects on real surfaces. However, none

seem to attempt to handle specular lighting effects.

Our initial work makes the assumption that the surface is planar. This limitation is still useful in many AR situations (especially indoors) because an uncluttered, planar surface is often favoured as a clean canvas on which to place and interact with virtual objects. Many other works seek planar surfaces for augmented reality applications, such as [18, 68, 92, 98].

A version of the work in this chapter was presented at and published in proceedings of the *International Symposium on Mixed and Augmented Reality 2012* [64].

3.1 Introduction

Augmented Reality (AR) will surely have the potential for world-changing impact when the enabling technologies it relies on come fully together in low-cost, mass-market mobile devices. The more that can be done from the sensors built into these devices, and without the need for infrastructure, the more likely it is that designers will be able to create wide-reaching and generally useful AR applications. In reaction to this, there has been a strong move away from custom hardware in AR research, and interest has been particularly high in what can be done for AR from the video stream from a single moving camera.

Some of the most important steps along this path were taken by work on real-time SLAM using a single camera which was able to build self-consistent maps of features incrementally and use these for long-term, drift-free camera tracking. Davison *et al.*'s early MonoSLAM system [31] using sequential filtering was improved upon by other work such as [37] and most significantly by Klein and Murray's PTAM [68] with a parallel tracking and map optimisation approach which enabled greater tracking accuracy and dynamics of motion.

In the past few years another big advance has been produced by the combination of modern optimisation algorithms and commodity parallel processing resources in the form of GPUs to permit real-time dense reconstruction from a single camera [86, 88, 121]. A dense surface model, generated live, allows AR objects to dynamically interact with the real scene; be occluded by, bounce off or even jump over real objects as shown in [86].

We consider the wealth of information in a standard real-time video stream from a moving camera which is currently still being ignored by most vision algorithms. Specifically, there is the potential to aim towards modelling of the reflectance properties of all scene surfaces and to map the full scene lighting configuration without any of the infrastructure such as light-probes currently needed for such estimation.

The full problem of estimating lighting and reflectance properties for general scenes is surely a long-term one, with difficult joint estimation problems coming into view once issues such as complicated curved geometry and object inter-reflections and considered. In this section we therefore make simplifying assumptions, but focus on an initial demonstration of real-time light-field capture, diffuse and specular lighting estimation and straightforward but highly effective placement of augmentations on specular surfaces — all with a single hand-held camera as the only data source. Specifically, we currently assume a static, planar scene; and static illumination which is well approximated as infinite relative to the camera motion.

3.2 Background

Much work on lighting and reflectance estimation has concentrated on single images, such as [55]. Nishino *et al.* [91] estimated the lighting and reflectance using a sparse set of images. At the other end of the scale [131] used a comparatively dense approach (several hundred images) to capture a view-dependent model of an object, but required a special capture rig. This information was then be used to estimate lighting and reflectance.

None of these approaches have taken advantage, as we do, of the huge amount of information coming from a 30fps live video feed; and in fact the processing and memory resources have only recently become widely available that makes dealing with such a large quantity of data feasible. The huge advantage of a real-time system is the feedback loop it gives to a user and we see in many AR systems which use vision for other capabilities. If processing is done post capture, we may find gaps where we decide that more data is needed, while a real-time user can immediately react to fill in these areas as appropriate.

Coombe *et al.* [27] use online singular value decomposition to capture a

compressed representation of a surface light-field via principal component analysis. Although their method is memory efficient, it is only an approximation to the real light-field. Their system takes about 1 second to incorporate a new image in to the approximation. While interactive, it is not at frame-rate (30fps) and, hence, does not use every available piece of information.

The most closely related work to our own was by Davis *et al.* [30], who captured light-fields in real-time with an unstructured representation and use them for novel view rendering. Their system required only basic knowledge of scene geometry and stored the light-field as a series of keyframes. Novel views are then formed via interpolation of these keyframes. Although this unstructured representation has the advantage that it does not require an accurate 3D model, we believe that we can be more ambitious and that surface light-fields can offer much more information about surface properties, which can be used for BRDF estimation, material based segmentation and object detection. We envision combining the real-time 3D reconstruction algorithms already in existence [86, 87, 88] with real-time light-field capture.

Artificial reality relighting and shadows generally need light-probes or omni-directional cameras to capture the illumination distribution, such as [92]. Some methods aim to work without probes, but these are generally for only specialised cases. For example, Madsen and Lal [80] demonstrated probeless illumination estimation for outdoor scenes by considering the only light sources to be the sun and sky. Our work aims towards a real-time, probeless system which will work in any environment.

3.3 Overview

At the core of our approach is the capture of the light-field emanating from the surface of interest using a tracked single browsing camera. We use the surface light-field representation as described in Section 2.6.1 to store the acquired data. In this work we will concentrate only on planar surfaces ($\hat{\mathbf{n}} = (0, 0, 1)$ for the whole surface). However, most of the concepts can be applied to a general surface as long as a geometric model of the surface, with normals, is known. This model could be captured with existing real-time methods using depth cameras such as [65, 87], or using a monocular camera such as the method discussed in Chapter 6.

Recall that the surface light-field is represented as a 4D function $L(\mathbf{x}, \boldsymbol{\omega})$. In the planar case $\mathbf{x} = (x, y)$ is the 2D position of the texture element in a simple Cartesian coordinate system, and $\boldsymbol{\omega}$ is the viewing direction defined on a hemisphere oriented with the surface normal, in this case, the z-axis. The hemisphere is discretized using the method outlined in Section 2.6.3.

3.4 Camera tracking and initialization

We make use of the freely available "Parallel Tracking and Mapping" (PTAM) program by Klein and Murray [68] to track the pose of the camera relative to the plane. The initialisation stage of PTAM uses feature correspondences from two frames to estimate the camera motion and 3D positions of the features along with a dominant plane estimate using RANSAC. This plane estimate is good enough for the small AR applications initially demonstrated in [68] but it is not always accurate enough for our application. To improve on this estimate we have a simple interface that allows the user to select a rectangular region of interest and then perform RANSAC plane estimation/refinement on the feature points within that region. This step does two things: improves the estimate of the plane parameters by removing feature points which do not lie on the plane, and provide a segmentation of the region of interest. Figure 3.2 shows a screen-shot of this interface in practice. The user has clicked on the four corners of the region of interest and only the relevant feature points belonging to the plane are now visible and used to estimate the plane. We use a rectangular region for ease of implementation but this is easily generalisable to arbitrary shaped regions.

In practice we have found the feature-based tracking to be robust and accurate enough for this application. We experienced that browsing the scene before data capture can increase the tracking accuracy by allowing the bundle adjustment to refine the keyframe poses. One limitation of the feature-based tracker is that the surface must have sufficient texture to have enough features to track from; this restricts the range of surfaces handled by the system. Another possibility for tracking the camera could be to use dense homography based tracking, as outlined in [76]. By using an M-estimator it should be possible to avoid tracking errors caused by the view-dependent specular texture (further discussed in Section 6.4). In the case of feature-based trackers, any features associated with the motion of specularities tend to be unstable and are

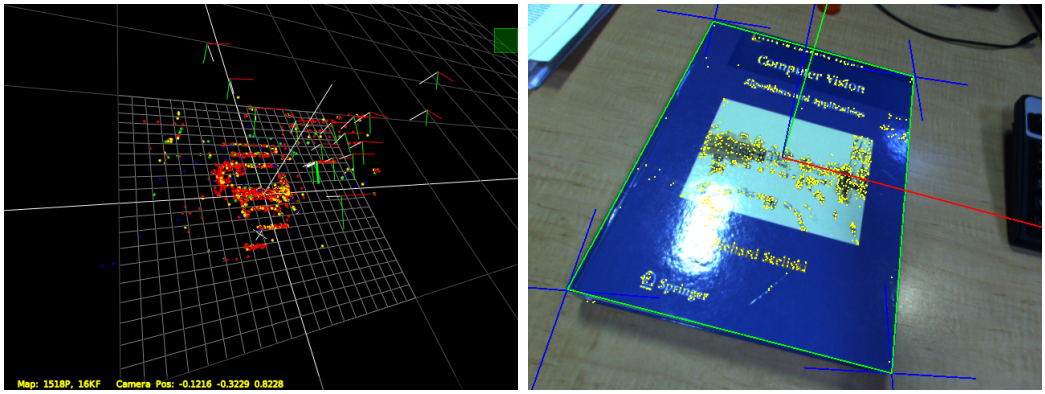


Figure 3.2: *PTAM's map of the world (left) and the plane initialisation interface (right). The user has clicked on the four corners (blue crosses) of the book to mark a rectangular region. A plane is then fitted to the feature points (yellow dots) within the region and the world coordinate frame is aligned with the plane and the rectangle's axis.*

---

naturally filtered out so they do not cause problems with the tracking.

Once the rectangular region has been specified we can build the light-field data structure. The region is discretized on a regular grid of dimension $M \times N$ and each discrete point has its own lumisphere. Each lumisphere will use the hemisphere discretization introduced in Section 2.6.3, with resolution L. Hence, using Eq. 2.57b we get that the total number of data points is $2MNL(L+1) + MN$ (typical values are $M, N, L = 256, 256, 60$ which gives 7321 points per lumisphere and approximately 480 million points in total). Each discrete point on a lumisphere holds a 4 byte value. The first 3 bytes store the RGB values and the last byte is simply used as a binary value which takes the value zero if that viewing direction is unseen. A future extension is to increase the dynamic range of the light-field by using all 4 bytes to represent colour.

3.5 Camera projection

The world coordinate frame is oriented so that the planar surface is the plane $z = 0$ and the normal to the plane points in the positive z direction. The x, y axis are aligned with the rectangular region of interest and centred in the middle of this region. The camera tracking will tell us the camera pose $\mathtt{T}^{wc}$ of

the camera relative to this world coordinate frame. With this configuration it is now simple to project a point $\mathbf{x} = (x, y, 0)$ on the plane into the camera using Eq. 2.4.

In order to write data into the surface light-field we must also know the viewing direction. This is simply the unit vector pointing from the point $\mathbf{x}$ on the surface to the camera position $\mathbf{t}^{wc}$. We then use inverse trigonometric functions to compute the direction in polar coordinates for use in the light-field:

$$\begin{bmatrix}\theta\\ \phi\end{bmatrix} = \begin{bmatrix}\cos^{-1}(d_z)\\ \tan^{-1}\left(\frac{d_y}{d_x}\right)\end{bmatrix}, \text{ where } \quad \mathbf{d} = \begin{bmatrix}d_x\\ d_y\\ d_z\end{bmatrix} = \frac{\mathbf{t}^{wc} - \mathbf{x}}{|\mathbf{t}^{wc} - \mathbf{x}|}. \tag{3.1}$$

In the special case of a Lambertian surface, or when we are only interested in the view-independent component of the radiance, we can avoid storing the entire light-field and just store a running average (e.g. mean or median) of the irradiance measured for each surface point. In this way we get a simple version of planar mosaicing (Fig. 3.3); multiple images are fused together to create a larger view of the planar surface with any view-dependent properties averaged out. Lovegrove [76] demonstrated a more advanced version of planar mosaicing that also estimates the plane orientation online.

3.6 Data Capture

A calibrated camera with fixed exposure, shutter and gain browses a planar scene capturing continuous video from multiple viewpoints. Given a new frame we back-project a ray from each point on the plane into the camera image (Fig. 3.4) and use bilinear interpolation to give a colour value. We use the pose of the camera to calculate the viewing direction for each pixel and calculate the 3 closest points on the lumisphere based on the triangulation in Fig. 2.9b. We then update these 3 points using the barycentric weights (as used for interpolation) and whether the point has been seen before. If the point is unseen, the point is updated with the observed value. If the point has previously been seen and λ is its barycentric weight then the new value I_{new} is given by an incremental weighted average of the old value I_{old} and the observed value I_{obs}:

$$I_{new} = \lambda I_{obs} + (1 - \lambda) I_{old}. \tag{3.2}$$

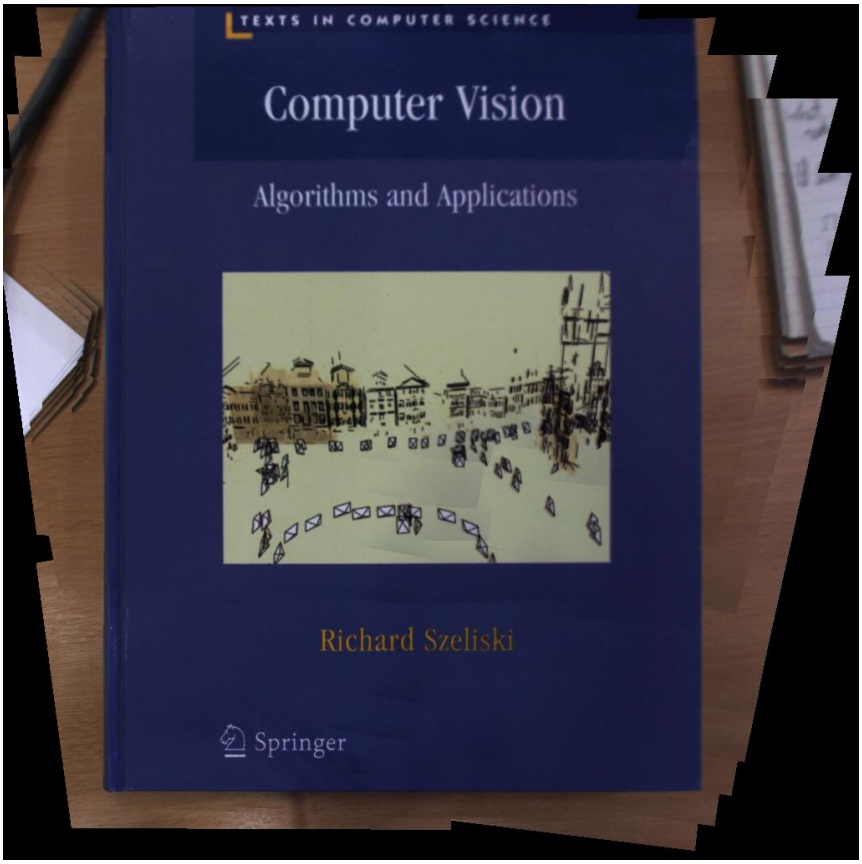


Figure 3.3: *Multiple images from a hand-held camera are stitched together to create a planar mosaic. This approach can create mosaics of higher resolution than that of an individual image. The surface is assumed to be planar so artefacts appear in regions not lying on the estimated plane (see the paper on the left and right of the book).*

PTAM features a way of estimating the reliability of the camera tracking. It outputs three possible states: 'BAD', 'DODGY', or 'GOOD'. We only input data from a frame to the light-field if the tracking state is 'GOOD'.

3.6.1 Feedback Mechanism

For specular surfaces we must use a high resolution lumisphere (at least 7000 discrete points) to give realistic results, since specularities are highly dependent on viewing direction. With each new video frame only one viewing direction is obtained per point on the surface, so it is unrealistic to expect to capture every viewing direction for every pixel. To get some perspective, in the absolute best

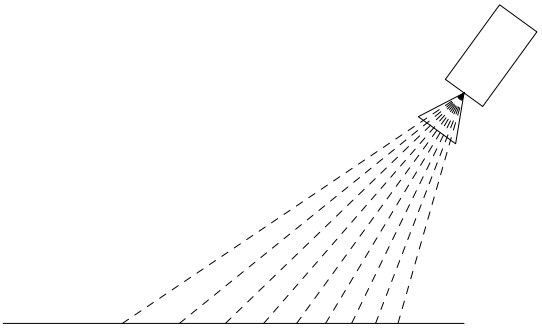


Figure 3.4: *Every point on the planar surface is projected into the camera image. Due to perspective projection, the viewing angle is different for every point on the surface.*

---

case scenario we would capture a new viewing direction for each point on the surface in every frame. That means for a a lumisphere with 7321 data points (a typical value) it would take just over 4 minutes to capture all the data. However, the reality is very different. We don't even get close to the best case and the time and effort needed to capture the complete light-field would make the system unusable.

Given that full capture of the light-field is not feasible we set a new goal to capture every viewing direction from at least one point on the surface. We will see later (Section 3.8) that this is enough information for a variety of applications and we can use global surface and lighting properties to fill in the gaps. Given a camera with a wide angle lens browsing close to the surface, a single frame captures many viewing directions (Fig. 3.4). This means capturing a view from each discrete direction is not time consuming. Our capture times are typically less than 1 minute to obtain all the viewing directions needed while only giving around 10% total coverage of the light-field.

Our system incorporates a visual feedback mechanism (see Fig. 3.5) to assist in getting coverage over the full range of viewing directions. This consists of a lumisphere which is coloured green when that particular viewing direction has been seen somewhere on the surface. The live viewing directions are shaded in red. This provides a feedback loop for the user to move the camera to previously unseen areas and is quite intuitive to use. The hemisphere is displayed to the user via projection onto the unit circle in the plane:

$$(r, \alpha) = \left(\tfrac{2\theta}{\pi}, \phi\right) , \tag{3.3}$$

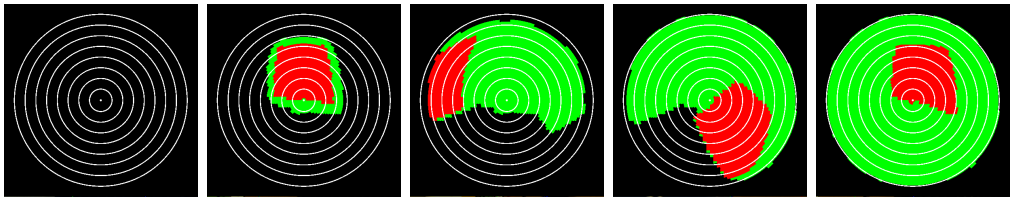


Figure 3.5: *Visual feedback mechanism for light-field capture guidance. Red represents the current view, and green the coverage so far.*

---

where (r, α) are 2D polar coordinates. This projection can also used be for visualisation of the lumispheres, as in Fig. 3.8. Concentric circles are drawn to represent 10 degree intervals of the inclination angle θ. The aim of the user is to fill the biggest circle with green. This means that all viewing directions with an inclination angle of less than 80 degrees have been seen. The last 10 degrees represents viewing directions at grazing angles to the surface. It is hard to capture data at these angles because tracking starts to fail; but we don't consider this a major issue because in our applications the information at these grazing angles is not needed. By using dense homography based tracking (as mentioned in Section 3.4) we might be able to improve the tracking at these grazing angles. The feature descriptors used in PTAM's tracking don't handle massive changes of viewpoint well.

For the feedback mechanism to report that capture is complete it requires a minimum of one observation from each viewing direction from any point on the surface. In reality, the incoming data is so dense that there is many more than one observation from each viewing direction.

3.6.2 Memory Usage

The surface light field is a four-dimensional structure and therefore consumes a vast amount of memory. We typically use a 256×256 grid for the planar surface. A high resolution light-field with 60 levels per lumisphere (7321 data points per lumisphere, 480 million overall) takes up nearly 2GB of memory (4 Bytes per data point).

As mentioned in Section 3.6.1, it is generally not possible to observe viewing directions less than 10 degrees from the horizontal. Therefore it makes sense to ignore this region of the light-field to save memory. Because there are more

data points at grazing angles, the memory saving are quite substantial; for a lumisphere with 60 levels (7321 points), if we remove all points less than 10 degrees from horizontal the structure remaining has the equivalent number of points to a lumisphere with 45 levels (4141 points) which is a 43% reduction in memory. In practice, we use the extra available space to increase the resolution of the light-field.

Our current method simply allocates all the memory required. However, the captured light-field is highly sparse (typically $\sim$ 10% fill) and would be well suited to a sparse quad-tree structure (for example). This would allow the memory usage to be compressed greatly. Current research hints towards efficient, scalable k-tree data-structures on GPU's with real-time update capabilities [136, 137].

3.7 Reflection/Illumination Model

The objective of capturing a surface light-field is to recover the view-dependent properties of the surface. When we look at standard models of illumination and reflection we see that the observed intensity is the sum of view-dependent terms and view-independent terms. For example, the Phong reflection model (Section 2.8) assumes that the light leaving a surface is the sum of three additive terms: ambient lighting, diffuse lighting and specular reflection. The first two are view-independent, only the specular reflection contribution is view-dependent. In the rest of this section we will refer to the sum of all view-independent terms as the *diffuse* term.

Having captured a whole lumisphere for a surface element, we are first interested in extracting the diffuse part. If we take a closer look at Eq. 2.61 we notice that the view-dependent specular part is a non-negative additive term. We can also note that specular lighting effects are very sensitive to viewing direction and the intensity drops off rapidly when deviating from the most intense viewing direction. Therefore, given a large number of viewpoints (as in a lumisphere) it is highly likely that one or more of those observations have a zero or negligible specular term. Hence, due to the non-negativity, it makes sense to take the minimum observed value to be the diffuse component, and this approach is used by Nishino *et al.* [91]. However, using the minimum value has its drawbacks: it is very susceptible to noise and outliers. Slight

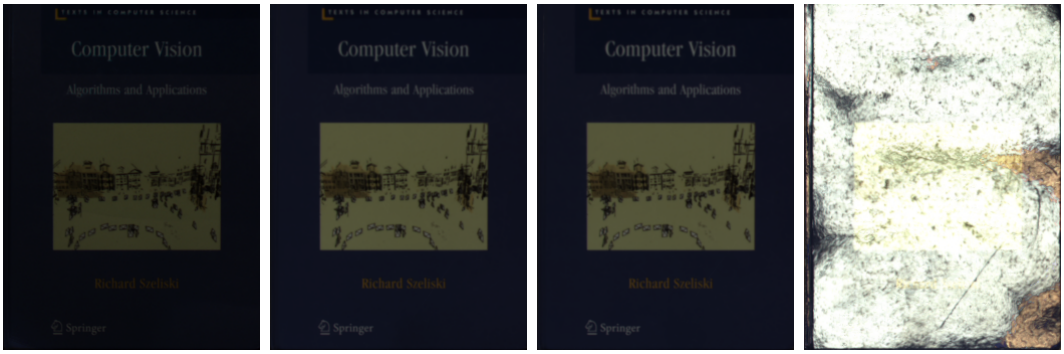
errors in camera calibration or pose estimation can lead to shrinking of brighter regions as the minimum can take the value of neighbouring less bright regions. Also, when a handheld camera is moved over a surface shadows are sometimes created which can mean the minimum does not accurately represent the diffuse term. Therefore, we make use of the median observed value, as used by Wood *et al.* [131]. It is more expensive to compute than the minimum, but it gives much more robustness.

Using the median essentially makes the assumption that less than 50% of the observed samples have been from specular-dominant viewing directions. This assumption appears to be valid in our experiments. During real-time light-field capture, it is possible to update the median of each lumisphere iteratively. To do this we store a histogram of the RGB values for each point on the surface and update this with each new frame. We also keep track of how many samples have contributed to the histogram. Given this information, calculating the median is as simple as finding in which bin the middle value lies. Note that we compute the histogram and median from the values stored in a lumisphere. The alternative would be to add every observation (on every camera frame) to the histogram. However, this would create a bias dependent on the trajectory of the camera.

We can also use the lower quartile as a robust estimate for the minimum. However, in practice we don't notice much difference between using the median or the lower quartile. A comparison is shown in Fig. 3.6.

In practice, we do not decompose the light-field into its diffuse and specular parts on every frame as it is not necessary and requires more processing and memory. However, we do store and update the minimum for each surface element (very efficient and fast to calculate) as a guide to aid capture and then compute the median once capture is complete.

The diffuse terms estimated for all surface elements can be combined into a diffuse texture for the surface, as illustrated in Fig. 3.7. Since the specular radiance component is only visible from angles close to the direct mirror reflection of a light source, the specular lumisphere (obtained by subtracting the diffuse term from the original lumisphere) can be used for estimating the position of light sources. We will discuss this further in Section 3.11.



(a) Minimum **(b)** Lower quartile **(c)** Median **(d)** Upper quartile

Figure 3.6: *Various measures exist to estimate the diffuse component of the surfaces illumination. The minimum (a) yields the darkest result but light coloured, thin structures (like the text) suffer from shrinking. The median (c) is a more robust solution and the result is very similar to using the lower quartile (b). (d) shows the upper quartile containing specular lighting effects.*

3.8 Rendering

Given a full surface light-field we can construct an accurate view-dependent rendering of the surface from any viewing direction. The process of rendering the surface light-field can be posed as a simple ray-casting problem. For each pixel $\mathbf{u}$ in the rendered image we find the intersection of the ray emitted from that pixel with the plane; resulting in a point of intersection $\mathbf{x}^* = (x^*, y^*, 0)$ on the plane (assuming such an intersection exists). We then test if the intersection point is in front of the camera and within the bounds of the region of interest. If a pixel passes both of these tests then we calculate the viewing direction $\boldsymbol{\omega}^*$ using Eq. 3.1. In general, $\mathbf{x}^*$ does not lie at the center of a surface element and $\boldsymbol{\omega}^*$ is not one of the discrete sample points on the lumisphere. To render at these non-discrete locations requires bilinear interpolation on the surface and barycentric interpolation on the lumisphere as described in Section 2.6.4. In practice, we generate a view-dependent texture for the entire region and then render this texture on the plane. For each discrete point $\mathbf{x}_i = (x_i, y_i, 0)$ in the planar region the view-dependent texture can be computed as follows:

$$I(x_i, y_i) = L_i\left(\boldsymbol{\omega}\left(\mathbf{x}_i, \mathbf{t}^{wc}\right)\right), \tag{3.4}$$

where L_i is the lumisphere at $\mathbf{x}_i$, $\boldsymbol{\omega}\left(\mathbf{x}_i, \mathbf{t}^{wc}\right)$ is the direction vector from the point on the surface $\mathbf{x}_i$ to the camera center $\mathbf{t}^{wc}$, and we use barycentric interpolation

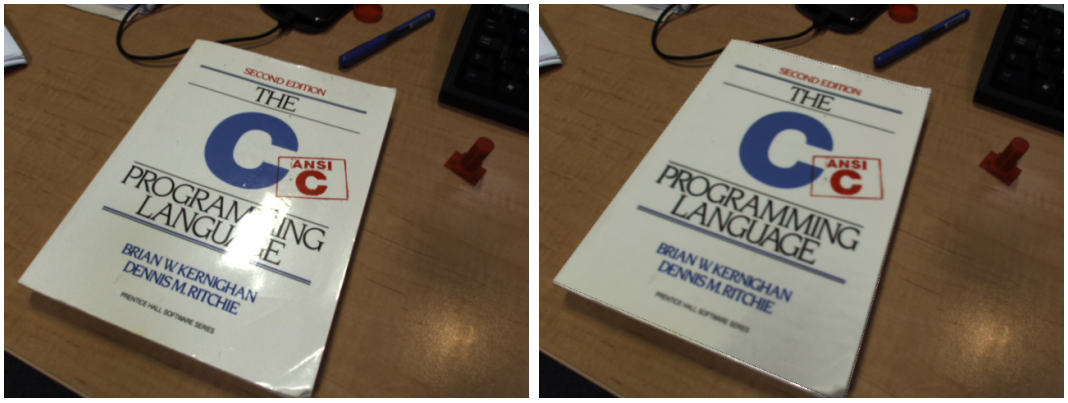


Figure 3.7: *Live camera view (left) and the specularity-free diffuse texture (right) calculated from the medians of the lumispheres.*

(Section 2.6.4) to compute $L_i(\boldsymbol{\omega})$ at non-discrete values of $\boldsymbol{\omega}$.

One big issue, as mentioned earlier, is that the captured lumispheres are very sparse. It is clear that gaps in the data cannot be filled in realistically without some kind of strong prior or global model. We could use the assumption that specularities are sparse so there is a high probability that an unseen viewing direction will not contain a specularity and, hence, just render the diffuse colour. Figure 3.9b shows the artefacts that appear when we try to do this. These artefacts appear because while one surface element may not have data for a particular viewing direction, its neighbour might. This creates large discontinuities in specular regions. From other viewing directions the specularities may not be visible at all, due to no data at those points or their neighbours.

We tried two different ways to combat this problem: 1) Lower the resolution of the lumispheres and spend longer on the capture stage so that they are no longer sparse; 2) Use a global model to fill in the gaps in the data (as hinted at in Section 3.6.1). The first method is quite limited. Small changes in the viewing direction of a specular surface can result in large changes to the observation. A lower resolution means that these small changes are not detected and this results in spreading out of the specularities (Fig. 3.9a). Quantisation effects are also evident when constructing the rendering from a low resolution.

Method two, however, gives good results (see Fig. 3.12). Our global model assumes that the planar surface is of the same material and, although it may have some varying diffuse texture, the specular component is the same across

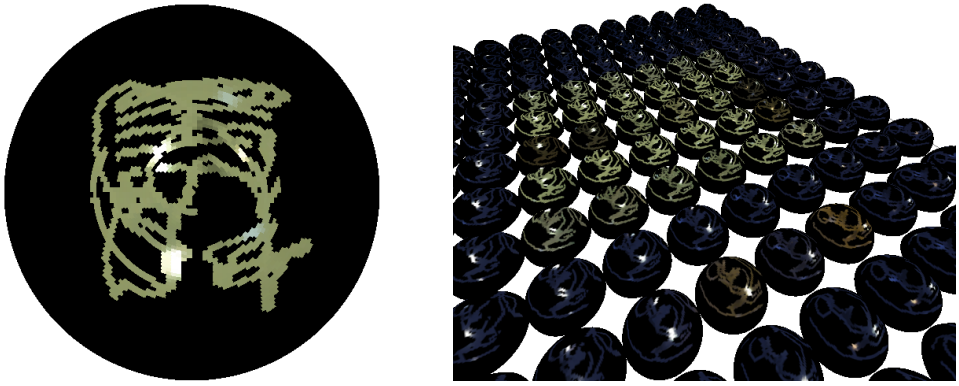


Figure 3.8: *Left: the projection of a captured lumisphere onto the plane. Black represents no data. The specular viewing directions can be easily seen. Right: a 3D rendered view of a small subset of lumispheres distributed over the surface. Notice that the captured data is very sparse for each individual lumisphere.*

---

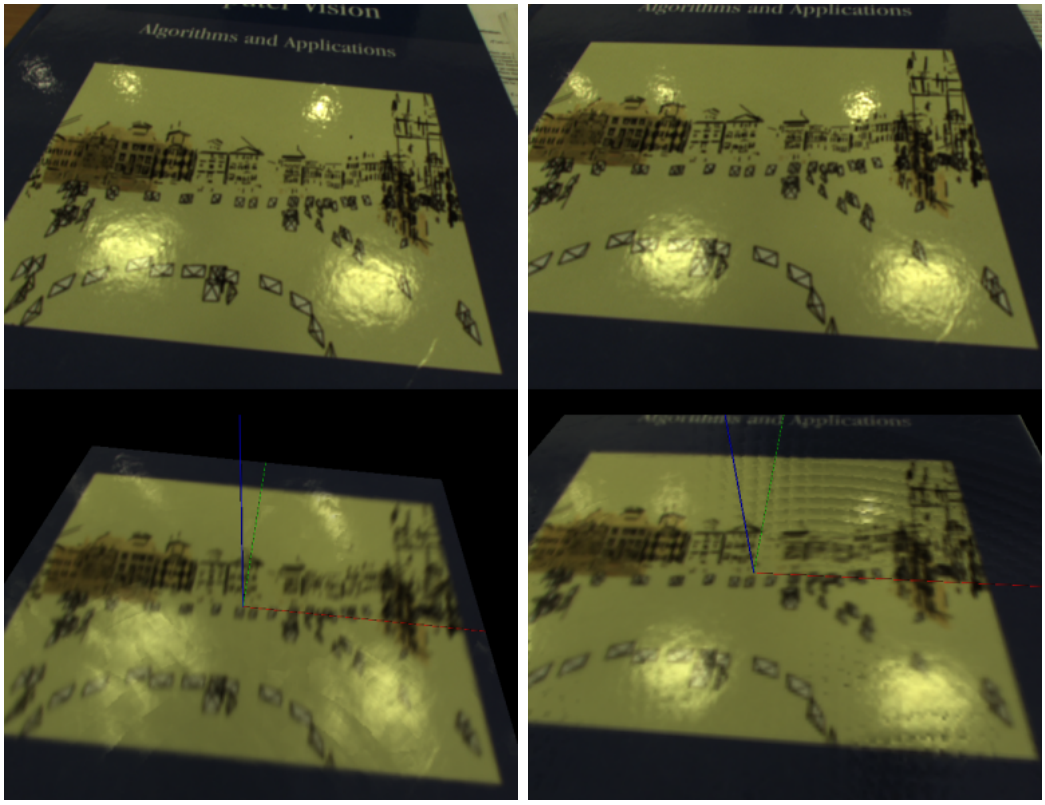
the whole surface - a valid assumption for surfaces with a homogeneous specular BRDF and subject to distant illumination. This means that we can combine all of the sparse specular lumispheres into one complete global specular lumisphere. Our visual feedback system ensures that this global lumisphere is filled. Figure 3.10 shows that the global specular lumisphere does not change significantly with extra data captured on top of what the feedback system deemed necessary. Clearly the feedback system works well.

In the Phong reflectance model (Eq. 2.61) the specular term for a light source can be rewritten as:

$$k_s(\cos\lambda)^\alpha i_s, \tag{3.5}$$

where λ is the angle between the direction of the reflected light and the viewpoint. There are two parameters which are defined by surface properties: k_s and α. The exponent α is considered a property of the material [96] and so can be assumed constant across surfaces of a single material. By assuming k_s is constant and that the direction of the reflected light is constant across the surface, consistent with a planar surface under distant illumination, we see that this whole term is now the same for every point on the surface. This helps to motivate our assumption of globally constant specularities.

The weakest part of our assumption is that k_s is constant across the surface. We might expect, for example, that a blue part of the surface will reflect more blue light than a red part of the surface. However, in practice, our as-



(a) Low resolution $n = 20$ **(b)** High resolution $n = 60$

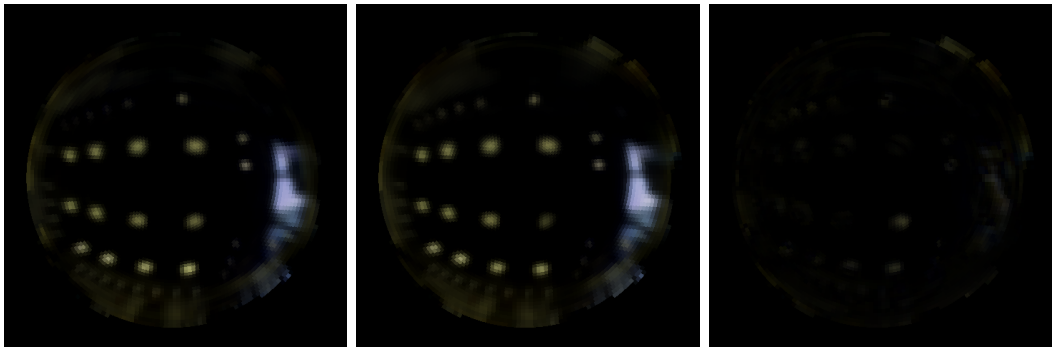
Figure 3.9: *Real camera views (top) and renderings from the light-field (bottom). The low resolution light-field tends to spread out specularities and has visible quantisation effects. The high resolution light-field is sparse and so there are gaps in the data where we can, at best, fill in with the diffuse colour.*

---

sumption gives good results with no apparent skew of colours. There is the further possibility that k_s could be somehow related to the diffuse colour of the surface, which we have already calculated. This is an area which requires further research. The surfaces we consider are of the same material but the colour may vary (e.g. the surface of a book). The distance to light sources is an order of magnitude bigger than the size of the surface so the distant illumination assumption is approximately true.

3.9 Implementation

Camera pose tracking by PTAM is run on the CPU. When a new frame is grabbed, it is sent to PTAM to calculate a pose and then copied to the GPU



(a) 6% Fill **(b)** 20% Fill **(c)** Difference

Figure 3.10: *The first image (a) has 6% fill of the light-field, just enough to cover all viewing directions as indicated by the feedback system. The second image (b) is after further capture to bring it up to 20% fill. (c) is a difference image. The most noticeable difference is that one of the lights is dimmer in the 20% fill image. This is in fact due to the user occluding that light during further capture. There are no major differences which shows that our feedback mechanism works well to obtain just the required amount of information.*

---

for all further processing. The light-field data structure is stored and updated entirely on the GPU. The fact that the problem is highly parallel and the power of GPU's for parallel processing enables our capture and rendering systems to run in real-time. Excluding PTAM, the system takes around 6-7ms per frame to capture or render the light-field using an NVIDIA GeForce GTX580 GPU.

3.9.1 Pixel Values as Irradiance

In our experiments the camera used is a Point Grey Flea2. We have measured it to have a camera response function which is highly linear until close to saturation point. Therefore, by keeping the shutter time constant the direct pixel values we use in the light-field are proportional to irradiance as long as we choose our exposure settings to avoid saturated pixels during light-field capture. If pixels become saturated then the values do not represent irradiance and the decomposition into diffuse and specular terms becomes invalid. This means that, currently, our capture stage is limited by the dynamic range of the camera. A future extension is to increase the dynamic range of the light-field data structure (as hinted at in Section 3.4) and use a dynamically changing shutter speed to capture a wider dynamic range. Preliminary experiments show

Image	RMSE Intensity	Max Grad. Diff.	Saturation
(a)	0.0388	0.735	0.02%
(b)	0.1023	0.957	2.90%
(c)	0.0277	0.126	0.00%

Table 3.1: *Quantitative evaluation of the difference between real camera views and views rendered using the global specular lumisphere modelling using light-field capture, for the five images (a)–(c) shown in Fig. 3.12. We show RMSE image intensity difference, maximum gradient difference ($|\mathbf{g}_{real} - \mathbf{g}_{fake}|$), and the percentage of saturated pixels in each real image. We observe that good general agreement between real and rendered views is obtained but that, with the current method, differences get larger in images with significant saturation. It is also interesting to note that the gradient difference measure reveals something extra, that we get better agreement for image (c) where the surface is smooth and lacks the fine dappled texture of the book used in (a) and (b). Our method currently assumes a perfectly planar surface.*

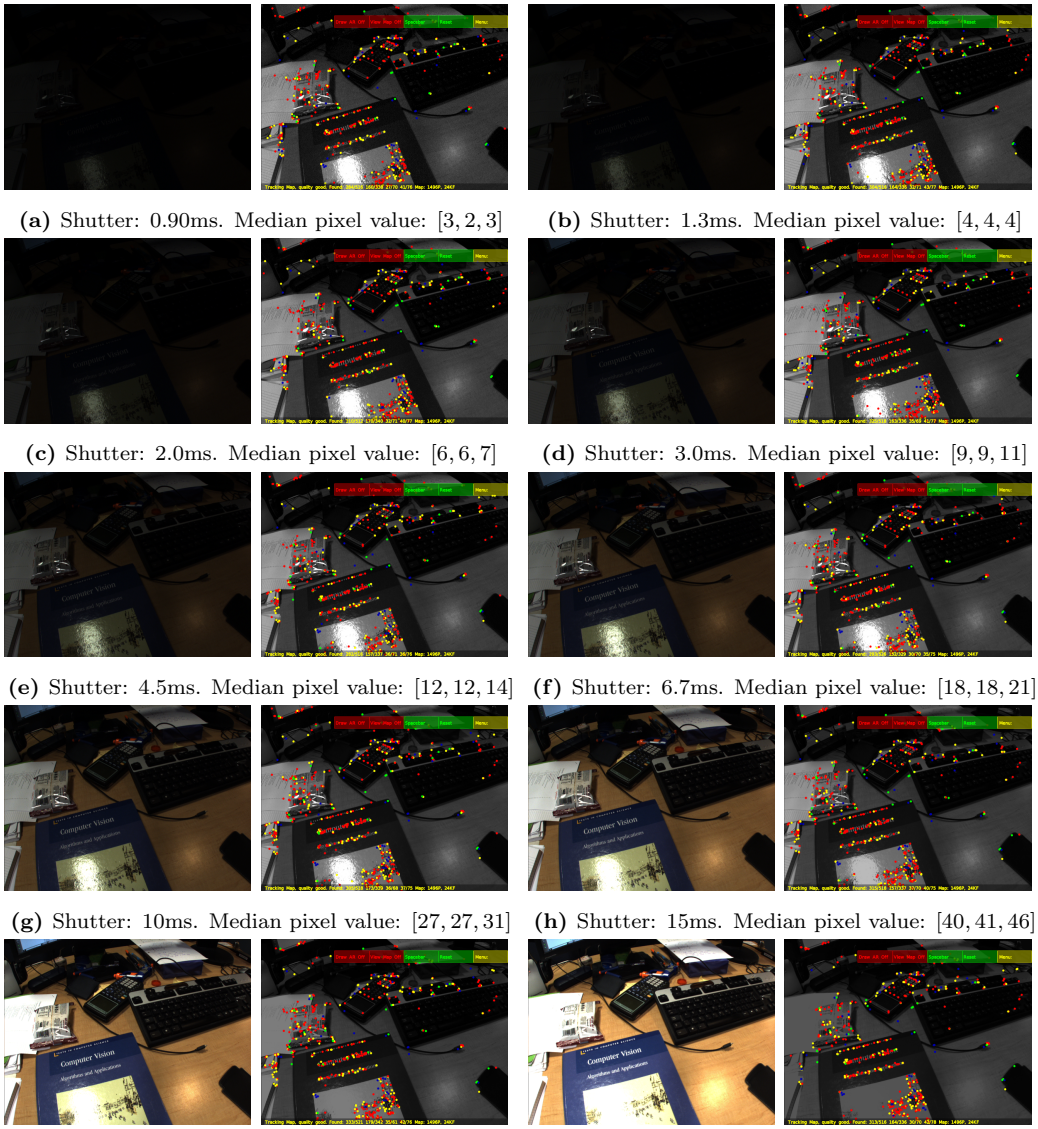
---

that PTAM is still able to track robustly through rapid changes in exposure provided that the images are renormalized to some fixed or slowly varying exposure using the camera response function. An example of this can be seen in Fig. 3.11.

3.10 Results

Figure 3.12 shows a comparison of real camera views with views rendered from the captured light-field with the global model. Table 3.1 shows some quantitative results obtained from these comparisons. We see that we have captured the general size, colour (hue) and position of the major light sources. Due to the averaging over the lumispheres, the fine texture of the surface (in a and b) is not captured. The surface is not truly planar. This fine texture is only visible at specularities so capturing this texture is difficult, and would require high resolution per-surface-element normal estimation (bump-map capture) — something we explore in Section 3.13.

The images show that the intensity of the specularities appear to be underestimated. There are two things which could contribute to this. The first,



(a) Shutter: 0.90ms. Median pixel value: $[3, 2, 3]$ **(b)** Shutter: 1.3ms. Median pixel value: $[4, 4, 4]$

(c) Shutter: 2.0ms. Median pixel value: $[6, 6, 7]$ **(d)** Shutter: 3.0ms. Median pixel value: $[9, 9, 11]$

(e) Shutter: 4.5ms. Median pixel value: $[12, 12, 14]$ **(f)** Shutter: 6.7ms. Median pixel value: $[18, 18, 21]$

(g) Shutter: 10ms. Median pixel value: $[27, 27, 31]$ **(h)** Shutter: 15ms. Median pixel value: $[40, 41, 46]$

(i) Shutter: 23ms. Median pixel value: $[60, 62, 71]$ **(j)** Shutter: 33ms. Median pixel value: $[90, 96, 111]$

Figure 3.11: *The shutter time of a handheld camera is varied on every frame yielding images with massive changes in exposure (left). The images are renormalized to a fixed exposure (right) and PTAM is still able to track the camera robustly. In image (a) a human cannot see any information in the image but when renormalized there is actually plenty of detail for PTAM to track against. The renormalized image has much more noise but still not enough to break tracking. The median pixel value of the raw camera image is provided to give a quantitative measure of exposure.*

(a) **(b)** **(c)**

Figure 3.12: *Comparison of real camera views (top) and views rendered from the light-field (2nd row) using the global specular lumisphere. The 3rd row shows absolute difference images and the bottom row shows difference of gradient images. The position and colour of the specularities is captured well but the intensity is underestimated. The global model smooths the fine texture in (a) and (b) but is not an issue in (c), clearly shown by the difference of gradients. The surface in (a) and (b) has a dappled texture which is not correctly modelled by the planar assumption.*

and most significant, is saturated pixels during capture. Table 3.1 shows what percentage of pixels are saturated in the live image, as a guide. It is clear that the largest errors are in image (b). In this case, not only is the specular component not reproduced accurately but the diffuse component has a larger error than the other examples. The saturation during capture has skewed the decomposition into diffuse and specular components. To avoid these errors, the exposure of the camera must be chosen carefully to avoid saturated pixels. However, in some cases we are limited by the dynamic range of the camera but this can easily be overcome.

The second factor possibly causing the intensity of specularities to be underestimated is using the median to calculate the diffuse component. This will tend to give a diffuse estimate slightly brighter than the real value, dampening the effect of the specular component. While the minimum may give a more realistic estimate of the diffuse component, it is not robust. Most noticeably, when using the minimum, slight errors in camera pose cause visible shrinking of the borders of bright surface regions.

3.11 Environment Map Approximation

We observed that the calculated global lumisphere can be used as an estimate of the environment map and used to predict the positions of light sources. Figure 3.13 compares the global lumispheres we capture with our method with environment maps obtained using the standard reflective spherical 'probe' method (hence the reflection of the cameraman). The light-fields were captured from different surfaces (both shiny books) and in different rooms. Both show good estimates of the environment map, picking up the colours of the lights and windows. It is possible to see some of the green of a tree outside the window. What is most positive about these results is that despite the surfaces having varying colours and texture, the overall colours of the specular component still accurately represent the real values. Note: this will not necessarily be the case for all materials and surfaces. The surfaces we have experimented with here are dielectric, a special class of surface that exhibits this property. This help to affirm that the assumptions we made to combine measurements across the surface into a single global lumisphere hold well.

The accuracy at which we can recover the environment map is dependent

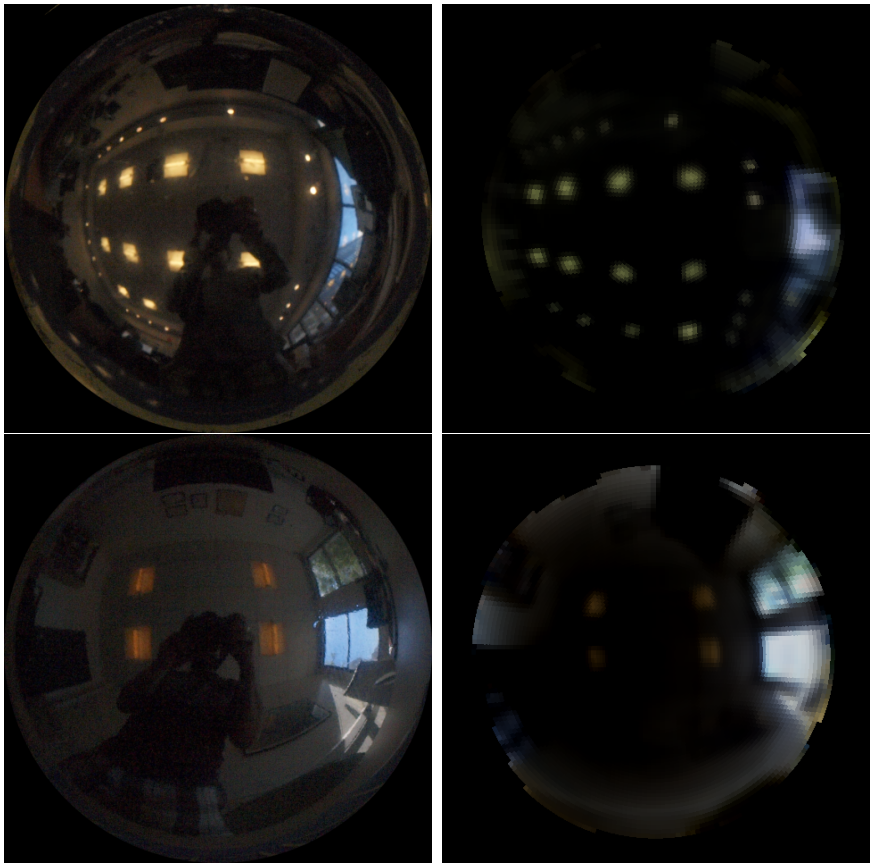


Figure 3.13: *Environment maps for two different rooms captured using a probe (left) and using our method based on specular reflections from a shiny book (right). The direction, intensity and colour of the light sources is captured well. The maps estimated from the light-field appear slightly blurry as they have been convolved with the BRDF of the surface.*

---

on the specular reflection properties of the surface. The α parameter in Eq. 2.61 determines how spread out the specularity is, and this is a material property. The spreading has a similar effect to applying Gaussian blur to the true environment map. A material with a small value of α will lead to a large spread and, hence, a largely blurred environment map approximation. We have also assumed that the intensity of the reflected light relative to the incident light is constant over all angles. The Fresnel equations tell us that this is not true, but it is a valid approximation except at very shallow angles.

Nowrouzezahrai *et al.* [92] used an environment map to apply realistic shadows to augmented reality objects. They applied a spherical harmonic approximation to the environment map to greatly reduce the resolution and enable

effective separation of light sources to cast hard and soft shadows. We believe that our specular lumisphere method captures an approximation of the environment map of sufficient quality which is suitable for such shadow effects, without the need for additional light probes or other capture devices. In essence, we use the existing surface in the scene as the light probe.

3.12 Augmented Reality on Specular Surfaces

One of the key components in producing convincing augmented reality is to make virtual objects fit with the illumination, shadows and reflection of the real objects and surfaces with which they interact. Most previous works have done this by estimating environment illumination and then using this to apply realistic lighting to virtual objects and cast virtual shadows onto real surfaces. However, none that we know have addressed the problem of placing virtual objects onto surfaces with a large specular reflectance component. The most considerable issue with augmented reality on such surfaces is that virtual objects must occlude specularities. The previous works make no attempt to do this yet the appearance of specularities must be handled in order to get a realistic AR view, as can be seen in Fig. 3.14.

Our captured surface light-field gives us the ability to remove the specularities in such a situation. There are two ways of doing this. The first is to extract the specular component from the light-field and subtract this from the incoming video feed. However, we found that our specular model is not yet accurate enough to produce good results with this approach. The second method involves replacing the occluded region with the diffuse term, as calculated from the surface light-field. This gives very convincing results. Figures 3.14 and 3.15 shows how a virtual object occludes the specularities. In the occluded region we have combined the surface's diffuse component and the reflection of the virtual object. The occlusion area is calculated by bouncing a ray off the surface and performing a simple ray-object intersect. This is implemented on the GPU in 5-6ms.

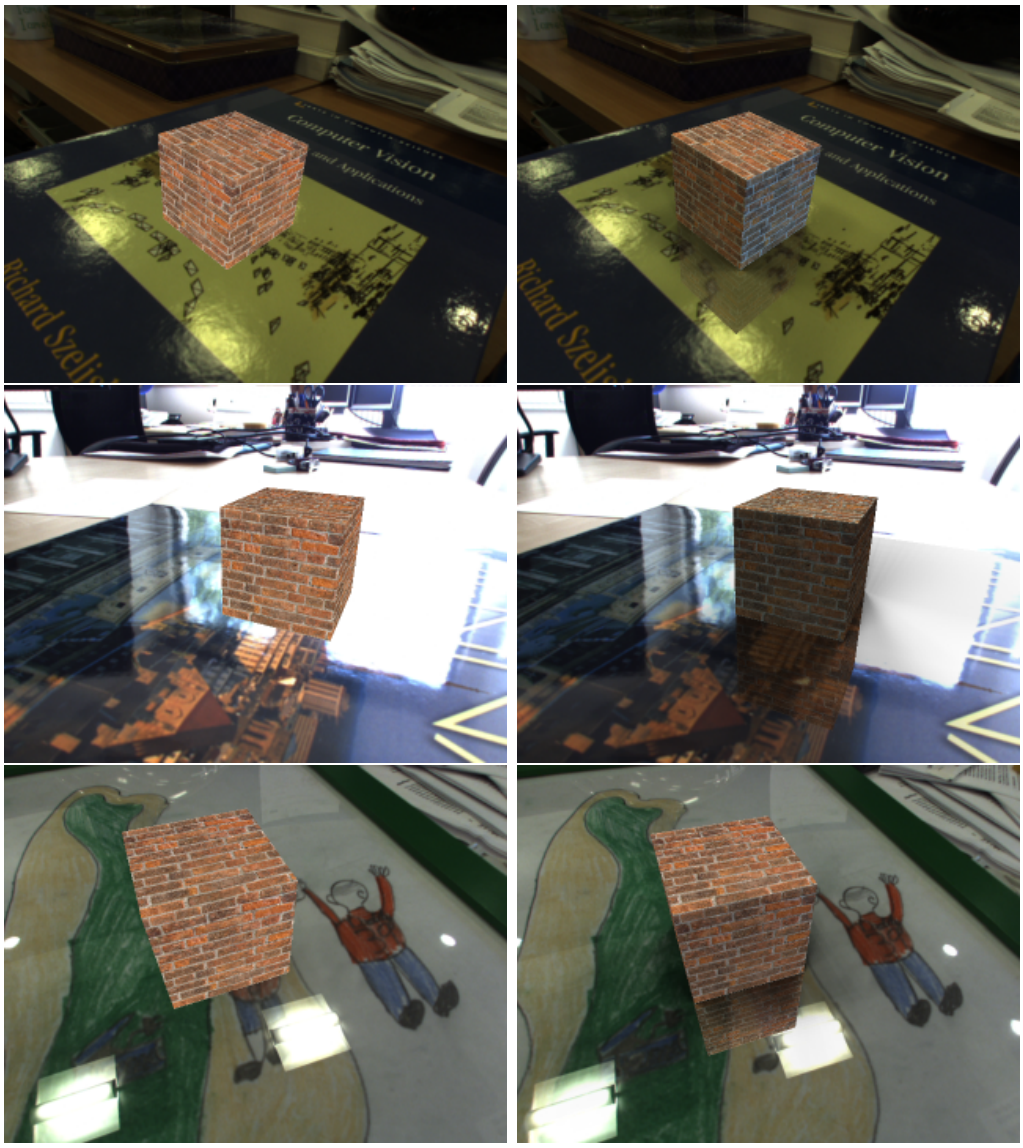


Figure 3.14: *Augmented scenes with (right) and without (left) illumination, shadows, occlusion and reflection. The views with the effects are much more convincing. While augmented reality is generally performed on diffuse surfaces, our method brings realistic AR to specular surfaces. The most obviously important aspect for realism is the occlusion of the specularities by virtual objects.*

Figure 3.15: *Sequence shots from two augmented video sequences. Notice how the object occludes the reflection of the light. The reflection is combined with the diffuse texture so that it joins seamlessly. Please see our video to view these clips, all rendered in real-time.*

3.12.1 Illumination

Illuminating the virtual object is a two-step process. So far we consider only diffuse virtual objects. First we need to calculate the illumination of the object with respect to the environment map. Then we need to calculate the illumination due to the surface. Since the surface is specular, the illumination effects on the object will be direction dependent.

We are yet to implement this full reflectance model (as it was not considered a part of our core research) but we show that even a simple model of the illumination is good enough to provide convincing results. The lighting model we use is as follows: we create a number of directional light sources based on the most intense directions observed in the global specular lumipshere/environment map. Currently, the selection of the light sources from the environment map is manual, however, this could be automated using connected component labelling or other blob detection methods. Once the light source directions are found we use basic diffuse illumination based on surface normals to render the virtual objects. This model is used in the examples and gives satisfactory results.

3.12.2 Shadows

Virtual shadows are added via a shadow map, this is essentially a mostly transparent image layered on top of the planar surface. The areas in shadow are made to be less transparent to darken the resulting image. We describe a few

different methods for computing this shadow map.

We start with a brute-force method. For each point on the surface, we estimate what proportion of radiance from the captured environment map is occluded by the virtual object. The intensity of the shadow is then calculated to be proportional to this value. We compute this by iterating over every viewing direction for every point on the surface. This brute-force approach implemented on the GPU takes 0.3 seconds and so is not suited to dynamic objects.

The preferred method that we use is based on the basic illumination from the previous section. We use a number of dominant light sources extracted from the environment map and generate shadows based on their position. By sampling each light multiple times while slightly perturbing its position we are able to create softer shadows.

Nowrouzezahrai *et al.* [92] provide an alternative method for environment-map based shadows. They pick a dominant light source to give hard shadows and use a spherical harmonic representation of the environment map to calculate soft shadows. This method could easily be applied to our system as well. However, in our experiments there has often been more than one dominant light source.

Note Saturated pixels can cause visual artefacts when applying a shadow map to a video feed - as seen in the middle images of Fig. 3.15. Given that there is no true measure of irradiance at these pixels there is no correct way to apply shadows.

3.12.3 Results

Figures 3.14 and 3.15 show the results of shadows, specular occlusion and basic illumination for a variety of surfaces and environments. These effects clearly make the virtual object look more realistically placed in the scene. For the glass-fronted picture on the far right of Fig. 3.14, the specular reflection is almost mirror-like and this makes the reflection of the object look very realistic. On surfaces with less mirror-like reflections (like the books) the direct reflection looks slightly out of place because it should be diffused by the surface. The information on how this reflection should diffuse is contained in the surface

light-field so we hope to improve this in the future.

Please see Appendix A for a video demonstrating the capture of a surface light-field using this our method and some examples of augmented reality on specular surfaces.

3.13 Bump-map estimation

Figure 3.12 showed that the planar assumption makes renderings overly smooth for a class of surfaces which have locally non-planar normals. To this end we attempt to improve the quality of the renderings by generating a bump-map for the surface: each point on the surface is assigned an independent normal $\hat{\mathbf{n}}$ which is used in rendering the specular component and for computing specular occlusion for AR.

To recover the normals in the bump-map we align each lumisphere L on the surface to the global specular lumisphere L_g by minimising the following error function:

$$E(\hat{\mathbf{n}}) = \sum_{\mathbf{x}\in\Omega} \left\| L(\mathbf{x}) - L_g\big(\mathbf{w}(\mathbf{x};\hat{\mathbf{n}})\big) \right\|^2, \tag{3.6}$$

where Ω is the set of valid measurements in the lumisphere L and $\mathbf{w}(\mathbf{x};\hat{\mathbf{n}})$ is the warp function (defined below) transforming coordinates on one lumisphere into another lumisphere aligned with normal $\hat{\mathbf{n}}$. Note that as seen in Fig. 3.8, the data in an individual lumisphere is sparse, hence it is more effective to project points from the individual lumispheres into the densely filled global lumisphere. For ease of implementation we use a 2D representation of the lumispheres projected onto the plane, as described in Section 2.6.3. In this situation the lumisphere occupies a circular region within the center of a square image of width w. This is now a Lucas-Kanade type image alignment problem as outlined in Section 2.9.

To derive the warp function $\mathbf{w}(\mathbf{x};\hat{\mathbf{n}})$ let us define the lumisphere projection function $\mathbf{p}(\mathbf{v})$ which maps 3D points $\mathbf{v}$ on the hemisphere ($\|\mathbf{v}\| = 1, \mathbf{v}_2 > 0$) to a plane:

$$\mathbf{p}(\mathbf{v}) = \frac{2\cos^{-1}(\mathbf{v}_2)}{\pi\sqrt{\mathbf{v}_0^2 + \mathbf{v}_1^2}} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}. \tag{3.7}$$

In this case, all points on a lumisphere project within the unit circle centred at the origin. The points then undergo a linear transformation to map them to

image coordinates:

$$\boldsymbol{\omega}(\mathbf{v}) = a\mathbf{p}(\mathbf{v}) + \mathbf{b}, \tag{3.8}$$

where $a = \frac{w}{2}$ and $b = [a, a]^T$. We also need the inverse of this projection. To start, we calculate the inverse of $\mathbf{p}(\mathbf{v})$:

$$\mathbf{p}^{-1}(\mathbf{x}) = \begin{bmatrix} \sin\left(\frac{\pi}{2}\|\mathbf{x}\|\right)\frac{\mathbf{x}}{\|\mathbf{x}\|} \\ \cos\left(\frac{\pi}{2}\|\mathbf{x}\|\right) \end{bmatrix}. \tag{3.9}$$

it is easy to verify that $\mathbf{p}^{-1}\big(\mathbf{p}(\mathbf{v})\big) = \mathbf{v}$ for $\|\mathbf{v}\| = 1$. Now the inverse of $\boldsymbol{\omega}(\mathbf{v})$ follows:

$$\boldsymbol{\omega}^{-1}(\mathbf{x}) = \mathbf{p}^{-1}\left(\frac{x-b}{a}\right). \tag{3.10}$$

We now need to write down the transformation between the two lumispheres. Let us recall how the light-fields are built: a lumisphere measures the light *leaving* the surface. For a specular lumisphere this is produced by taking the incoming light and reflecting it through the surface normal. Hence, our captured lumispheres are dependent on the surface normal. The incoming light is not. Therefore, by reflecting a lumisphere back through the surface normal we can get the *incoming* light on the surface. During capture we assumed that the surface normal was $\hat{\mathbf{e}}_z$. Hence, for a direction $\mathbf{v}$ on the lumisphere (outgoing light), then the direction of the incoming light is given by:

$$\mathcal{R}(\hat{\mathbf{e}}_z)\mathbf{v} = \left(\mathtt{I} - 2\hat{\mathbf{e}}_z\hat{\mathbf{e}}_z^T\right)\mathbf{v}, \tag{3.11}$$

where we have defined $\mathcal{R}(\hat{\mathbf{e}}_z)$ to be the transformation matrix created by a reflection in the plane with normal $\hat{\mathbf{e}}_z$. We can then recover the outgoing light direction for the new surface normal $\hat{\mathbf{n}}$ by doing another reflection. The overall transformation is denoted by $\mathtt{R}_{\hat{\mathbf{n}}}$:

$$\begin{aligned} \mathtt{R}_{\hat{\mathbf{n}}} &= \mathcal{R}(\hat{\mathbf{n}})\mathcal{R}(\hat{\mathbf{e}}_z) \\ &= \left(\mathtt{I} - 2\hat{\mathbf{n}}\hat{\mathbf{n}}^T\right)\left(\mathtt{I} - 2\hat{\mathbf{e}}_z\hat{\mathbf{e}}_z^T\right). \end{aligned} \tag{3.12}$$

It is simple to verify that the composition of these two reflections creates a rotation matrix, hence the notation. We can now write down the full warp function:

$$\mathbf{w}(\mathbf{x}; \hat{\mathbf{n}}) = \boldsymbol{\omega}\big(\mathtt{R}_{\hat{\mathbf{n}}}\boldsymbol{\omega}^{-1}(\mathbf{x})\big). \tag{3.13}$$

In words, we take a point on an image, project it onto the lumisphere, transform it to another lumisphere aligned with normal $\hat{\mathbf{n}}$ and then project it back onto an image.

To minimize the energy in Eq. 3.6 we re-write the normal vector as a rotation applied to the z-axis: $\hat{\mathbf{n}} = \mathtt{R}\hat{\mathbf{e}}_z$. Our aim is now to find $\mathtt{R}$ which minimises the energy. We follow the approach described in Section 2.9 and rewrite the energy as a function of generators of the Lie group SO(3). Let us define the following function:

$$f(\mathbf{u};\mathbf{x}) = L(\mathbf{x}) - L_g\Big(\boldsymbol{\omega}\big(\mathcal{R}(\mathtt{R}\hat{\mathtt{R}}(\mathbf{u})\hat{\mathbf{e}}_z)\mathcal{R}(\hat{\mathbf{e}}_z)\boldsymbol{\omega}^{-1}(\mathbf{x})\big)\Big), \tag{3.14}$$

such that the energy to be minimized can be written as

$$E(\mathbf{u}) = \frac{1}{2}\sum_{\mathbf{x}\in\Omega}\big(f(\mathbf{u};\mathbf{x})\big)^2. \tag{3.15}$$

We can compute derivatives of the energy using the chain rule:

$$\begin{aligned}\frac{\partial f(\mathbf{u};\mathbf{x})}{\partial \mathbf{u}}\bigg|_{\mathbf{u}=0} = -\frac{\partial L_g(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{w}(\mathbf{x};\hat{\mathbf{n}})} \frac{\partial \boldsymbol{\omega}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\bigg|_{\boldsymbol{\beta}=\mathtt{R}_{\hat{\mathbf{n}}}\boldsymbol{\omega}^{-1}(\mathbf{x})} \\ \frac{\partial \mathcal{R}(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}}\bigg|_{\boldsymbol{\alpha}=\mathtt{R}\hat{\mathbf{e}}_z} \mathtt{R}\frac{\partial \hat{\mathtt{R}}(\mathbf{u})}{\partial \mathbf{u}}\bigg|_{\mathbf{u}=0} \hat{\mathbf{e}}_z\mathcal{R}(\hat{\mathbf{e}}_z)\boldsymbol{\omega}^{-1}(\mathbf{x})\ .\end{aligned} \tag{3.16}$$

In our case we are representing L_g as an image, so $\frac{\partial L_g}{\partial \mathbf{x}}$ is just an image derivative. $\frac{\partial \hat{\mathtt{R}}}{\partial \mathbf{u}}$ provides the generators of the Lie group SO(3) (Appendix B). Note that when computing derivatives we end up with the product $\mathtt{G}_2\hat{\mathbf{e}}_z$, where $\mathtt{G}_2$ is the third generator of SO(3). It is easy to verify that this product is zero and hence all derivatives w.r.t the third generator are always zero. Hence, we only need to consider the changes with respect to the first two generators of the Lie group. This reduction of one dimension is wanted (and needed) because we are solving for a normal which only has two degrees of freedom. It arises because the axis of rotation must always be perpendicular to $\hat{\mathbf{e}}_z$.

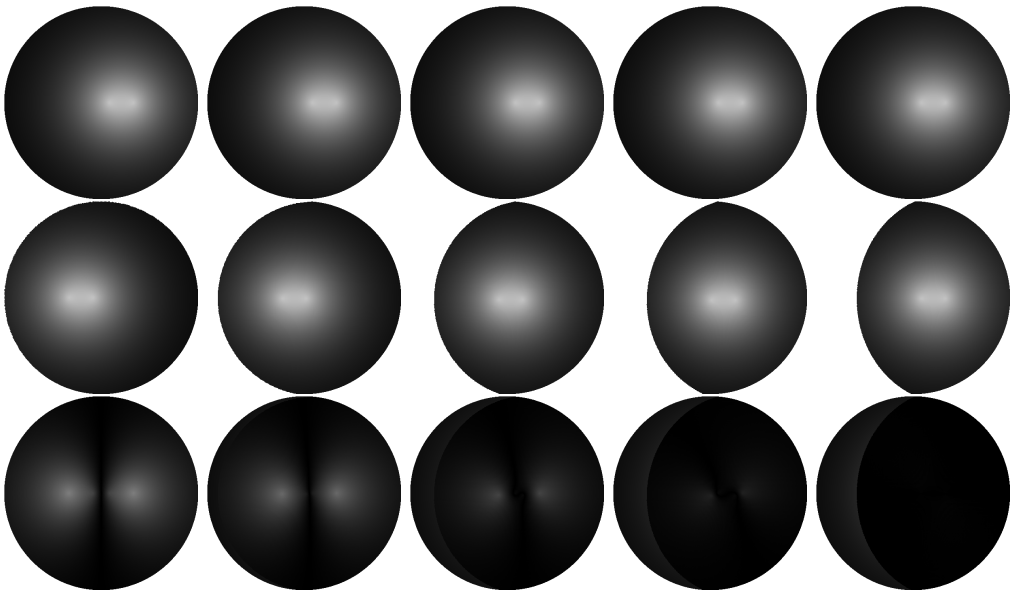
We use Lucas-Kanade image alignment, as detailed in Section 2.9. We compute the products J^TJ and $J^Tf(\mathbf{0})$ using

$$J^TJ = \sum_{\mathbf{x}\in\Omega}\nabla_{\mathbf{u}}f(\mathbf{0};\mathbf{x})^T\nabla_{\mathbf{u}}f(\mathbf{0};\mathbf{x}), \tag{3.17}$$

$$J^Tf(\mathbf{0}) = \sum_{\mathbf{x}\in\Omega}\nabla_{\mathbf{u}}f(\mathbf{0};\mathbf{x})^Tf(\mathbf{0};\mathbf{x}), \tag{3.18}$$

where:

$$\nabla_{\mathbf{u}}f(\mathbf{0};\mathbf{x}) = \frac{\partial f(\mathbf{u};\mathbf{x})}{\partial \mathbf{u}}\bigg|_{\mathbf{u}=\mathbf{0}}. \tag{3.19}$$



(a) 0 iterations **(b)** 2 iterations **(c)** 4 iterations **(d)** 6 iterations **(e)** 15 iterations

Figure 3.16: *A mock example of aligning lumispheres. The top row shows the reference lumisphere. The second row shows the lumisphere which is being aligned, transformed by the current normal estimate. The bottom row shows the error between the two. We see that the method converges quickly and results in a normal rotated by 0.327 Radians (18.7 degrees).*

We are only using the first two generators of SO(3), so $\mathbf{u} \in \mathbb{R}^2$. We work on colour images so $f(\mathbf{u}; \mathbf{x}) \in \mathbb{R}^3$, and $\nabla_{\mathbf{u}} f(\mathbf{0}; \mathbf{x}) \in \mathbb{R}^{3\times 2}$.

Once the sums in Eqs. 3.17 and 3.18 have been computed we use the normal equation (Eq. 2.71) to find the solution $\mathbf{u}^*$, perform the update $\mathtt{R} \leftarrow \mathtt{R}\hat{\mathtt{R}}(\mathbf{u}^*)$ and continue to iterate. Once we have found the final solution $\mathtt{R}^*$ we apply the rotation to $\hat{\mathbf{e}}_z$ to recover the optimal normal $\hat{\mathbf{n}}^* = \mathtt{R}^*\hat{\mathbf{e}}_z$.

Note One may look at Eq. 3.13 and think that we could solve directly for $\mathtt{R}_{\hat{\mathbf{n}}}$. However, in this formulation we still have 3 degrees of freedom, the lumispheres are allowed to rotate about the z-axis, which is inconsistent with the solution we require.

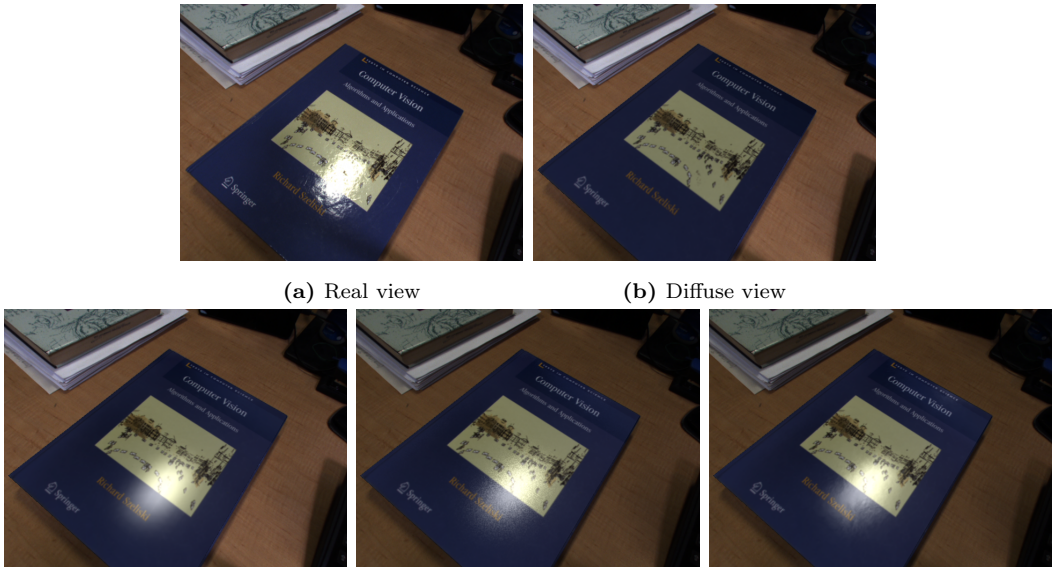
3.13.1 Results

Figure 3.16 shows a synthetic example of aligning 2 lumispheres by minimising the energy in Eq. 3.6. The input data is generated by two Gaussian peaks in intensity placed in different locations on the reference lumisphere and the lumisphere to be aligned. The algorithm is able to quickly converge to the correct solution; we can clearly see that the error has been minimized by looking at the difference images in the bottom row of Fig. 3.16. Note that another possible solution would be to rotate 180 degrees about the center, however, our optimisation does not allow this type of transformation as it does not correspond to a rotation between normal vectors.

Constraining the optimisation As stated earlier, the captured lumispheres are sparse. Additionally, for a large class of surfaces the appearance of specularities is sparse over the range of possible viewing directions so there is a good chance that a lumisphere could contain no specular information. If this is the case there is no way to constrain the optimisation; we are trying to align a textureless region. In reality, slight variations of intensity in the observed diffuse-only directions would lead to the optimisation moving towards an incorrect result. Given the two degrees of freedom in the normal estimation, we need to make sure that each lumisphere observes at least two distinct specularities in order to get a correct result. This is achievable but requires more care and time during light-field acquisition to ensure this is the case. It would be simple to add user-feedback to help achieve this.

Once a surface light-field has been captured, we apply the above optimisation to every lumisphere on the grid, initialised with $\hat{\mathbf{n}} = \hat{\mathbf{e}}_z$. Figures 3.17 and 3.18 shows the effect of the recovered bump-map on the synthetic rendering of a specular surface. We also compare to a bump-map with random normals and to the result without a bump-map. While the result using our estimated bump-map does not exactly match the real view, it does provide a much more convincing rendering than the other two. An alternative method to achieve qualitatively similar results is the work of Wang *et al.* [126] who estimate the parameters of a stochastic model representing the structure of the surfaces bump-map.

Figure 3.19 shows a coloured representation of the estimated surface nor-



(a) Real view **(b)** Diffuse view

(c) $\hat{\mathbf{n}} = \hat{\mathbf{e}}_3$ **(d)** Random $\hat{\mathbf{n}}$ **(e)** Estimated $\hat{\mathbf{n}}$

Figure 3.17: *Synthetic renderings of the surface (b-e) versus the real view (a). The diffuse view (b) is independent of the normals in the bump-map. (c-e) show specular lighting effects with different surface normals.*

---

mals for the same book as in Figs. 3.17 and 3.18. We observe that, while our bump-map estimation captures the local texture of the surface well, it also seems to estimate larger scale deformations which don't reflect the true surface structure. We believe this is due to a lack of data to constrain the optimisation in those areas.

3.14 Conclusion

We have demonstrated a new method for recovering detailed lighting and surface information with a standard single hand-held camera AR configuration. In standard monocular tracking and mapping systems the effects caused by complicated lighting and specular surfaces are ignored or even cause problems. These can, in fact, be used for estimation of properties crucial for realistic AR. Specifically, in this chapter we have demonstrated easy and rapid light-field capture from planar specular surfaces, and the straightforward use of this for reflectance and environment map estimation without the need for any additional probes or hardware. To our knowledge this is the first system to demonstrate using a planar surface as a light probe, rather than a more traditional spherical

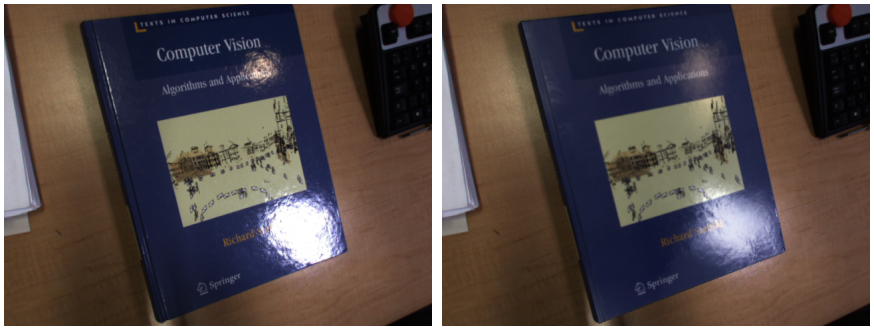


Figure 3.18: *The estimated bump-map of the surface is not absolutely accurate to the original rendering. However, it does provide a much more convincing rendering than the smooth result*

---

probe. We have used the captured data to give convincing real-time demonstrations of the placement of augmentations on specular surfaces with realistically synthesized reflections, shadows, illumination and specularity occlusion.

3.14.1 Further Work

Currently the camera browsing the scene is restricted to have a constant shutter time. For this proof-of-concept application this is adequate, but for real scenes a wider dynamic range is needed. This constraint can easily be relaxed if we know the shutter time and camera response function so that we can convert pixel values to radiance values (as detailed in Section 2.7). Preliminary experiments show that feature-based tracking such as PTAM continues to work robustly through varying exposures if we renormalise the images. Clearly there is a limit on this range; too short an exposure and the renormalized image will just be noise, too long an exposure and too much of the image will be saturated and motion blur becomes more likely.

A clear next step with this work is to seek to combine light-field capture with live dense reconstruction, as discussed in Chapter 6. There are many surfaces in the real world (such as the books we have used in our experiments) which have significant texture but also partial specular reflection characteristics. The shape of such surfaces can be reconstructed in 3D based on their diffuse texture, and then a surface light-field can be defined relative to this shape for the capture of specular lighting effects. The captured light-field could then

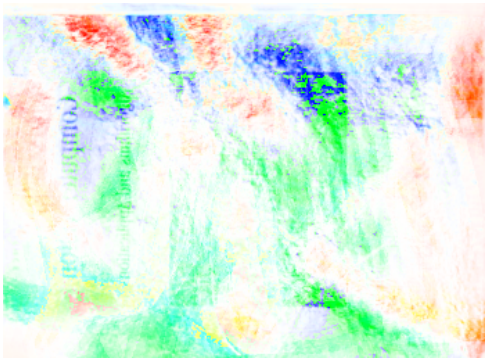


Figure 3.19: *Normals estimated by our bump-map optimisation. The hue represents the normal direction and saturation starts at* 0 *when* $\mathbf{n} = (0, 0, 1)$ *and increases proportionally to the angle of the normal from the z-axis. We see in the high frequencies that the mottled texture of the book has been captured. There are also significant low frequency components but it is not clear if these are part of the true surface or errors in the optimisation.*

---

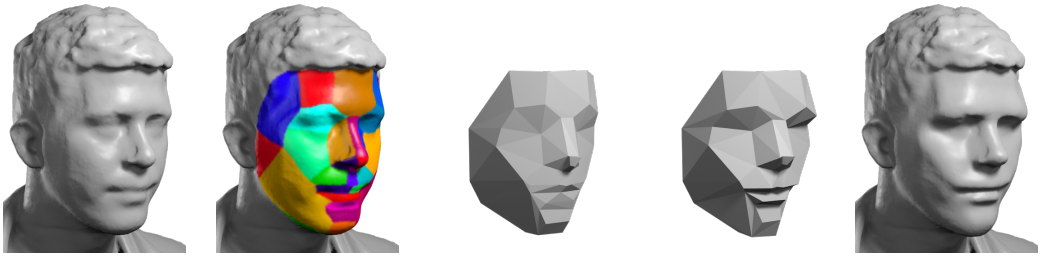
be used to refine the geometry in way similar to Section 3.13. A longer term challenge is to deal with objects whose surfaces are more purely specular, and do not offer enough texture for standard stereo matching and reconstruction. Coping with these will involve substantial future work on the joint estimation of surface shape and reflectance properties.

The method should be easily extendible to surface light-field capture with a plenoptic camera (Section 2.6.2). The amount of data captured on each frame with a light-field camera is far greater than that of a normal camera. This means we can fill the surface light-field structure quicker and more densely. Additionally, it is possible to generate depth-maps from light-field cameras [56, 95]. This could be a step towards simultaneous dense reconstruction and surface light-field capture.

Finally, if we capture an environment map and a surface light-field, we know both the incoming and outgoing light at the surface. This should lead to good BRDF estimation. This could then be used for material based segmentation to aid object recognition. The estimation of a bump-map could also be useful in material detection by recovering characteristic bump-map properties of each material.

4

Sculptural Stylization



(a) Input mesh **(b)** Input segmented **(c)** Abstracted mesh **(d)** Stylized abstracted mesh **(e)** Stylized output mesh

Figure 4.1: *An example of* sculptural stylization *applied to a bust. We transform a scanned input mesh into a stylized output emphasizing anatomy and characteristic features. The input is segmented into* sculptor's planes *via alignment with a template model. The angular definition of the segmented mesh is enhanced in a highly controllable optimization which generates smooth, full resolution output interactively.*

This work was started as part of an internship with Dan B Goldman and Linjie Luo of Adobe Research[8] and continued as a collaboration with Imperial College London. A version of this work is published on arXiv [63].

4.1 Introduction

With the recent advances in 3D scanning using commodity hardware, the ability for consumers to make their own 3D scans of objects and people is closer than ever. Combine this with the recent availability of desktop 3D printers (Fig. 4.2) and 3D printing services [62, 115, 116] and we open up a whole new creative culture where consumers can scan, edit and then print their own 3D models of

[8] http://www.adobe.com/technology.html

Figure 4.2: *A selection of desktop 3D printers. The Makerbot Replicator 2 (left, source: www.flickr.com)) and Ultimaker 2 (middle, source: www.igo3d.com) are ready-to-go complete solutions. The RepRap TriColour Mendel from RepRapPro (right, source: www.reprappro.com) is purchased as a kit and is capable of printing 3 colours at a time. The RepRap [105] project works towards designing 3D printers that are self-replicating.*

---

real world objects or people. Just as photography democratized 2D imagery by unchaining it from the painter's hand, 3D scanning and printing may democratize 3D representations by decoupling them from the sculptor's hand. Such a system could even be envisaged as a "3D fax" that can scan objects of various sizes, transmit them electronically, and reprint facsimiles of those objects at a distant location.

Currently, one of the most common consumer use cases of this 3D scanning and printing pipeline is the creation of figurines of family and friends. There are do-it-yourself solutions mostly using Kinect style RGB-D sensors. FabliTec [42] and ReconstructMe [104] both have the ability to create "3D selfies"; the user mounts the RGB-D sensor on a tripod or desk at eye level and then rotates on a swivel chair in front of the camera. The software then outputs a coloured, water-tight model ready to be sent to an online 3D printing service or to be printed at home on a desktop 3D printer. ReconstructMe also features a hand-held scanning mode to reconstruct arbitrary 3D models. Shapify [117] also provide software for home scanning with a Kinect sensor but additionally have their own scanning booths distributed around various countries. The booths use high-resolution scanning hardware for faster and better quality scans when compared to Kinect style acquisition. Ego3D [38] is an online service which requires just 3 photos of a person to create a high quality bust. However, they make use of a digital sculptor to produce the final model.

(a) textured **(b)** untextured

Figure 4.3: *(a) Commercial 3D printing services offer the ability to produce a textured model [42]. The texture is able to hide inaccuracies and low quality geometry. (b) The majority of consumer 3D printers are only able to print in a single colour, much like traditional sculpture.*

Most scanning services and online printing services will output a textured 3D model. An example from Fablitec is seen in (Fig. 4.3a). The geometry of the scanned model can be of low quality but the texture can still create a visually pleasing result. In contrast, most consumer 3D printers only print in a single colour (Fig. 4.3b). In this case there is no texture to hide low quality geometry and features which may have been visible in the texture could disappear. This provides the motivation for this work: how can we enhance 3D scans of human busts to make them more visually appealing when 3D printed in a single colour. We immediately see a close connection with traditional sculpture where a model is made of a single material of a single colour.

Thus far, research on 3D scanning and printing has rightly focused on geometric accuracy. However, 3D printed human figures often appear lifeless, particularly when generated with commodity scanning and printing. Casting aside the physical limitations, which are already rapidly being eliminated, there is a more artistic method to overcoming this problem: We propose a new, creative approach expanding on these technologies called *sculptural stylization*, which we define to be when a 3D model deviates from geometric accuracy to enhance appearance in a manner analogous in many ways to that of non-photorealistic rendering for 2D imagery. Sculptural stylization could be either for purely aesthetic purposes; or to improve the recognizability of the subject by emphasizing characteristic features [130]; or to fit the limitations or requirements of a given

sculpting medium; or simply to imitate historically popular styles of sculpture. Well known sculptors consistently exaggerate away from geometric accuracy in order to achieve some combination of these aims (a preliminary analysis of how sculpted busts deviate from real human geometry is provided in Section 4.4).

As described in the rest of this section, one important class of stylizations first decomposes a model into masses, planes or other components. These decompositions could be purely geometric, but are often based on higher-level object semantics or a knowledge of underlying anatomy. We use the term *sculptural abstractions* to describe these semantically-based decompositions, using the word *abstraction* in the sense of isolating essential qualities.

We propose an interactive approach to stylization of human faces, founded on a specific sculptural abstraction used by artists called *the planes of the head* — regions of the surface of the face which, although not literally planar, may be grouped together by geometric consistency or separated by underlying anatomy. We generalize this concept to scans of other objects beyond simply faces, and refer to the general concept as *sculptor's planes*. Our method's stylizations emphasize the contrast between these forms by accentuating the angles between them and making the regions within them more planar. We imagine an expert sculptor who first studies his subject and identifies these sculptor's planes, then produces a rough, low-detail model in which the surface angles between planes are exaggerated and deviations within planes are smoothed, and finally introduces fine details of the surface.

Our system, using a 3D scan of the subject as its input, supports an amateur digital sculptor in several stages: First, the sculptor's planes are automatically identified — either by aligning a scan to a generic, pre-segmented head model, or by automatic geometry analysis; and the scan is then simplified to an abstracted mesh using the given segmentation. Second, we construct and optimize an energy function to balance the exaggeration of angles of this abstracted mesh with preservation of the original form and some key individual facial characteristics. The sculptor can adjust both the global scale and local scale of exaggeration as well as fidelity to the facial characteristics. Finally, we map the resulting deformation back to the original full resolution scan, and the sculptor can locally adjust the amount of detail to include from the original scan, as well as the smoothness between sculptors planes. The last two stages are performed continuously at interactive frame rates, providing detailed user

control of both global and local parameters.

4.2 Related Work

The Non-Photorealistic Rendering (NPR) literature features a long history of transforming photographs and 3D models into artistic depictions. In particular, Gao *et al.* [47] used an abstracted model similar to the planes of the head for 2D stylization and exaggeration. Winnemoller *et al.* [130] demonstrated that certain types of abstraction improve human memory tasks, including facial recognition and scene recall. We hypothesize that sculptural abstraction can play a similar role for 3D shape. Furthermore, many papers in NPR explore the depiction of 3D scenes to improve shape understanding. Hertzmann and Zorin developed a hatching technique to emphasize aspects of curved surfaces that are otherwise hard to perceive [59]. Cole *et al.* [21] analyzed where people draw lines when representing 3D surfaces, which might be similar to the features that sculptors exaggerate.

We are unaware of previous work that defines the broader notion of *sculptural abstraction* in 3D geometry. However, the idea of 3D collage [46] seems to fall within this scope, as it attempts to represent the gestalt of a given 3D shape by aggregating several smaller 3D primitives. Bhat *et al.* [10] also applied a method similar to image analogies [58] in the context of 3D geometry. This method applied a learned geometric texture to an entire model rather than local deformations based on existing geometry, and thus could be used as a complement to our sculptural abstraction in order to reproduce the surface texture imposed by various real-world sculpting tools and mediums.

Other existing methods for automatic non-metric mesh enhancement take approaches which have predictable and global effects. Specific to faces, Blanz and Vetter [11] fitted a parametric 3D model built from a large family of faces to a new scan, and could create caricatures by shifting the parameters away from the mean face. This results in a global linear deformation, in contrast to our non-linear template-driven deformation. PriMo [13] is another deformation strategy that has been used to demonstrate caricature, by modelling polygons as prisms of finite thickness and constructing an energy function to increase angles between them. This resembles our exaggeration energy term, but although our system can be used for stylizations such as caricature — e.g. exaggeration

of key individual features — we characterize our goal of abstraction as sharpening visual contrast and form *without* modifying key individual features. This distinction is explored in detail in Section 4.5.3, in which we propose *Lanteri constraints* for maintaining fidelity to these features.

Various signal processing techniques exist for meshes which can exaggerate features. The simplest is the 3D equivalent of the unsharp mask, where a 3D shape is enhanced by smoothing its normals and then exaggerating their angles in the opposite direction [133]. Eigensatz *et al.* [39] operate in the curvature domain, but do not process meshes in real time. More recently, anisotropic smoothing and exaggeration have been implemented in graphics hardware [19]. Although these methods can produce some results similar to ours, they don't offer simple local control.

Some previous works have explored automated abstraction of shapes [81] and shape collections [134] into semantic groupings, though they do not demonstrate stylizations based on these groupings. Similarly, Lee *et al.* [73] decomposed geometry into a base mesh plus displacements not unlike our two-scale decomposition, but their model is not suitable for deformation because it associates each high-resolution vertex with only one base polygon.

4.3 Background

In order to learn what techniques sculptors use in their abstraction, we interviewed two accomplished professionals: Gio Nakpil, a sculptor who works both in physical and digital sculpture, including work for Industrial Light and Magic and Valve; and Mike Magrath, an instructor at Seattle's Gage Academy of Art. Although many techniques are particular to specific styles, we narrowed our focus to abstraction techniques for human busts that span multiple styles.

Two common points of reference between the sculptors emerged: First, both sculptors mentally and visually decompose a subject into component *masses* or *forms* at a variety of scales — starting from larger groupings such as the forehead, and working down to smaller details like the sides of the nose and the folds of the eyelid. And second, in the case of human faces this decomposition is often described in terms of *planes of the head*, as defined in Section 4.1.

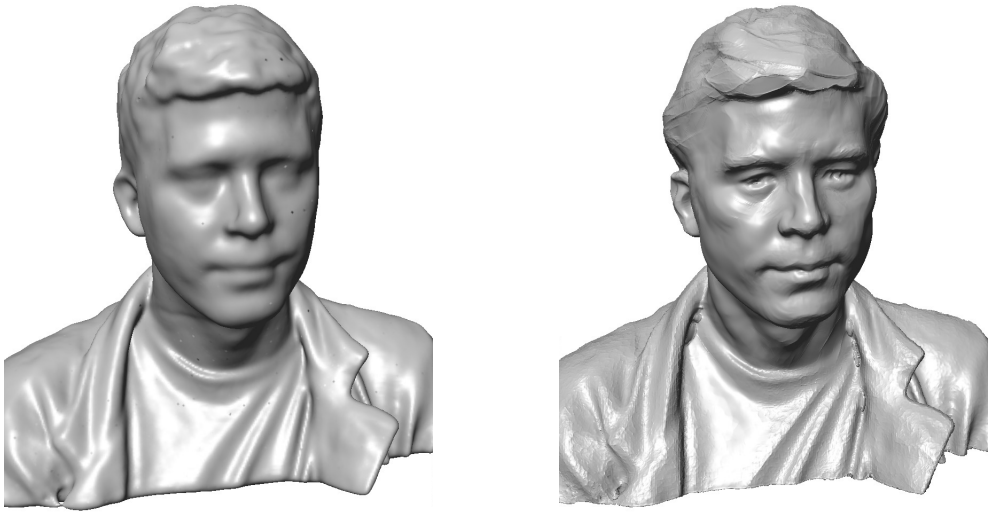


Figure 4.4: *Left: A scan acquired using KinectFusion. Right: The same scan after sculpting by Gio Nakpil. Note exaggerated features such as cheekbones, temples, jawline, chin, and sides of the nose.*

Nakpil emphasized the role of highlight and shadow in sculpting technique; in this context referring to the broad artistic definition of regions brighter or darker than the mean, not only specular highlights and cast shadows. Napkil noted that sculptors view their work under a range of lighting conditions and from many angles to understand and control the shapes and relationships of those highlights and shadows. Since sculptures are typically lit from above, sculptors may tilt surfaces toward the horizontal, exaggerating the contrast between highlights and shadows. Figure 4.4 shows a scan we provided to Nakpil alongside his sculptural interpretation using ZBrush.

Magrath made special note of the exaggeration of "soft" and "hard" forms. This nomenclature refers to the distinction between angular regions with rapidly alternating high and low curvature (e.g. the bridge of a nose) versus rounded, uniform medium-curvature regions (e.g. the forehead). He characterized his abstraction technique as "making the hard forms harder, and the soft forms softer."

The notion of *planes of the head* permeates literature about sculpting, and dates back at least to the turn of the 19th century, when the sculptor Edouard Lanteri (whom Rodin described as "Dear Master" [71]) wrote a seminal textbook. Although much of his work describes technical methods for ensuring

Figure 4.5: *Left: Examples of planar decompositions in painting and sculpture, ©John Asaro 1976, excerpted with permission. Right: A variation of the* Planes of the Head *entitled* Memorized*, intended for student memorization.*

---

metric accuracy, he also suggests a methodical approach to exaggerating these planes and their junctions using side lighting to reveal them more clearly to the sculptor:

> When these divisions of form have been obtained in their proper drawing by studying each separately, the work may appear a little hard. Then it becomes necessary to work by colour — that is to say, by the comparative values of the half-tints, in simplifying or accentuating the surfaces or planes which divide these forms. [72]

The concept of *planes of the head* was further formalized in the late 20th century, most notably by John Asaro, an instructor at Art Center College in Pasadena, California [4]. Asaro hypothesized that decompositions of facial form into masses and planes have been used by painters and sculptors since antiquity, and proposed planar decompositions for works by Cassatt, Rodin, Degas, Rubens, Vermeer, Michelangelo and many others. As an aid to students of sculpting and painting, Asaro developed a set of canonical busts which exaggerate the planes to their logical extreme, reducing a human head to a nearly polygonal object (see Fig. 4.5).

While sculptors use these planes to exaggerate certain forms, Lanteri's

writing also makes it clear that certain aspects of human anatomy and individual facial characteristics are perceptually relevant, and must not be perturbed by these stylizations. He writes voluminously about using calipers to ensure that certain distances, ratios, and angles are preserved in a sculpture of a given model. Here is one such passage:

> From the tip of the nose measure the distance to the most projecting part of the chin, i.e. the subcutaneous mental eminence... To make sure of the correctness of the previous measure, you now take the distance from the ear to the mental eminence (see Fig. 40) in your calipers, and if this measure, taken from both ears of course, coincides with the previously fixed point on the chin, your measure runs a fair chance of being correct. If it is not, you have to... retake both measures, until they agree. [72]

Additional measurements proposed by Lanteri appear in Appendix F. We make use of these in our algorithm to preserve the facial characteristics of the subjects during stylization.

We hypothesize that the *planes of the head* perform a similar function to lines in 2D illustration: Artists abstract detail that isn't important and emphasize features that do matter for recognition and understandability. Although we may not yet know exactly what features are perceived as important for emphasis in 3D, we choose to follow methods employed by professional sculptors because it is reasonable to assume they relate to perceptual importance.

Thus the challenge of sculptural abstraction is to enhance or exaggerate visual contrast between sculptor's planes, while simultaneously preserving certain critical global geometric relationships. Although it may seem these goals are fundamentally incompatible, in the sections that follow we demonstrate a system that accomplishes this automatically, interactively and under flexible user control.

4.4 Analysis of scans of humans vs. sculptures

We observed informally that, in spite of some sculptors' focus on geometric fidelity, some features in realistic classical and modern sculptures appear exaggerated. This is hard to detect when comparing people directly to sculptures,

because their reflectance qualities differ so drastically. However, these differences become more apparent when people and sculptures are scanned. Perhaps most critically for our application, 3D prints of scanned faces often appear flat and lifeless.

An ideal analysis of sculptural stylization would require extensive data collection of existing sculptures and comparisons to human scans of the individual models being sculpted. In the absence of such a dataset, we performed an informal comparison of the depth of the eye sockets between six scans of humans – four from KinectFusion [87] and two using the method of Beeler *et al.* [7] – and five scans of sculptures – three of Rodin's "Burghers of Calais"[9], and two busts from the MIT CSAIL 3D Model Database[10]. First, a number of feature points on the models were identified and manually labeled. Then, we measured the depth of the eye sockets using four separate scale-invariant measures: Measure A takes the distance from the midpoint of the inner corners of the eye to the plane formed by the midpoint of each eyebrow and the tip of the chin, normalized using the distance between the points at the base of the ears. Measure B is the same as measure A, but using the outer corners of the eyes. Measure C takes the distance from the midpoint of the inner corners of the eye to the plane formed by the saddle point at the bridge of the nose and the corners of the mouth, also normalized using the distance between the ears. Measure D is the same as C, but using the outer corners of the eyes. Each measure was aggregated across human and sculpted scans using both the mean and the median. A summary of the results can be found in Table 4.1.

		A	B	C	D
Mean	Human	0.069	0.134	0.094	0.157
	Sculpt	0.112	0.177	0.131	0.193
	% Increase	**63.1**	**32.5**	**39.3**	**23.1**
Median	Human	0.065	0.127	0.091	0.156
	Sculpt	0.091	0.169	0.127	0.183
	% Increase	**40.2**	**33.5**	**40.4**	**17.4**

Table 4.1: *Measures of the depth of the eye sockets on scans of humans vs. sculptures.*

[9] http://www.stanford.edu/~qianyizh/projects/scenedata.html
[10] http://people.csail.mit.edu/tmertens/textransfer/data/

For each aggregate measure, the sculpted scans had significantly deeper eye sockets than the human scans. Note that all fiducial points were identified by hand, the quality and resolution of the scans varies significantly, and the measures we propose are somewhat ad hoc. Furthermore, the size of the sample is much too small to draw concrete conclusions. Nonetheless we see a strong suggestion that sculptors such as Rodin routinely exaggerated features of facial anatomy such as the depth of eye sockets.

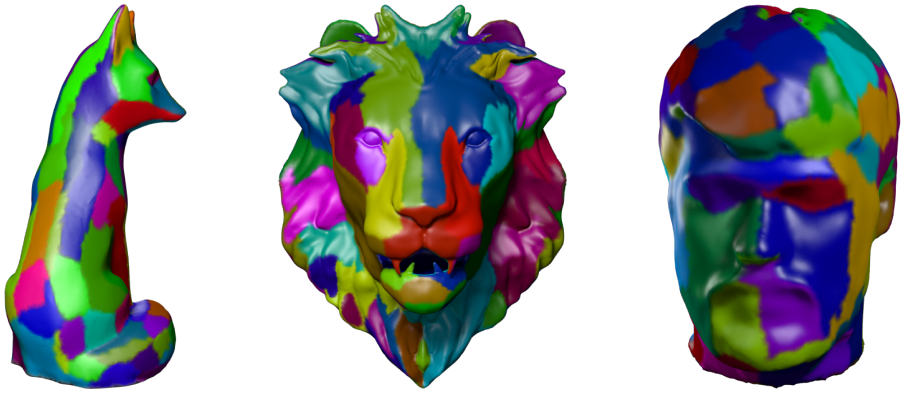
4.5 Interactive sculptural stylization

Our goal is to allow a user to interactively stylize an input mesh guided by sculptural abstraction principles. While the principles we use are applicable to any type of input, we focus mainly on faces because these are more relevant in a sculptural context. Our system first computes an *abstracted mesh* from the input model. The abstracted mesh serves as the *sculptural abstraction framework* for the subsequent stylization steps, and is divided into meaningful regions corresponding to the planes from the template model.

The subsequent sculptural stylization optimization and stylization transfer phases of our system run at real-time rates and are designed for interactive use by a user in control of global and/or local parameter settings. We stylize the abstracted mesh by minimizing an energy function with several terms of various purposes: to exaggerate angles between different regions; to enforce flatness within each segment; to regularize the stylization towards the original shape; and to enforce *Lanteri constraints* that preserve geometric measurements characteristic of a person's identity. The transfer of these stylizations back to the input mesh produces real-time full resolution results for inspection and further adjustment. The amount of smoothing can be controlled here both globally and locally; and the interactive display of the full resolution deformed mesh is crucial in permitting fine user control over subtle deformations.

4.5.1 Abstracted Mesh Generation

We first generate the abstracted mesh $\mathcal{M}_C$ that captures locally meaningful sculptural abstractions from the input mesh $\mathcal{M}$. The process of constructing the abstracted mesh consists of segmenting the input mesh into sculptor's planes



Mesh source: [28] Mesh source: [29] KinectFusion scan

Figure 4.6: *Some examples of segmentation using the method from Variational Shape Approximation [20]. For the human head, the regions are not always aligned with the anatomy which can cause unappealing artefacts during our processing.*

---

and then creating a coarse mesh approximation based on this segmentation.

For general 3D models, we find that Variational Shape Approximation (VSA) [20] often derives plausible geometric abstractions from the input mesh by grouping faces of similar normals into contiguous regions, as seen in Fig. 4.6. However for human faces, due to our heightened perceptions, VSA can cause noticeable artefacts because it does not preserve semantically meaningful segmentations and salient features such as eyes and lips.

Thus, we employ a semantically segmented human head template to generate the abstracted meshes from human face models. More specifically, we use the *Planes of the Head* model (Fig. 4.5) because it takes into account both the principles of sculptural abstraction and facial semantics. To acquire the 3D template, we scanned a Planes of the Head model using KinectFusion [87] and manually segmented the model into sculptor's planes (shown with coloured segments in Fig. 4.7). For each novel input mesh $\mathcal{M}$ we manually pre-align it to the template and then use the non-rigid alignment method of Li *et al.* [75] to refine it. Now, the vertices of the input and the template match very closely so we transfer the segmentation by taking the nearest vertices as correspondences while ensuring the regions are distinct and connected (Fig. 4.8b).

The transferred segmentation divides $\mathcal{M}$ into K regions $\{R_r\}_{r=1}^{K}$ (when

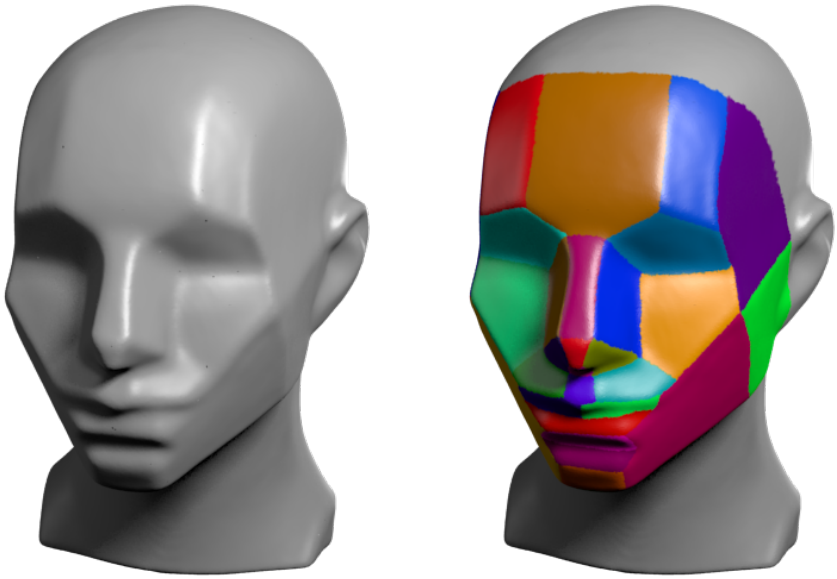


Figure 4.7: Planes of the Head: Memorized *scan and segmentation.*

---

using VSA for segmentation the number of regions can vary, when using the Planes of the Head model then $K = 32$). We generate the abstracted mesh $\mathcal{M}_C$ by approximating each sculptor's plane with a small number of triangles. We follow the approach of Cohen-Steiner *et al.* [20]: anchor vertices are automatically placed along the borders between sculptor's planes and a constrained Delaunay triangulation is used to fill the regions. Note that our algorithm only uses the anchor vertices on the boundaries between regions; the triangulated mesh is used for visualization purposes.

4.5.2 Abstracted Mesh Stylization

After we compute the abstracted mesh $\mathcal{M}_C$ for the input mesh $\mathcal{M}$, we build up a stylization framework on the abstracted mesh in order to eventually transfer the stylization to the input mesh (Section 4.5.5). This yields the stylized abstracted mesh $\mathcal{M}'_C$. The stylization framework includes both *exaggeration* of the angles between the regions and *planarization* of the features within each region, our interpretation of techniques described by professional sculptors in Section 4.3. Here we introduce a few definitions to facilitate our formulation of the stylization framework.

We first define the sculptor's plane π_r approximating each region R_r by the

(a) (b) (c)

Figure 4.8: *The segmentation of the template model is transferred to the input mesh using nonrigid alignment.*

---

normal $\hat{\mathbf{n}}_r$ and centroid $\mathbf{c}_r$. $\mathbf{c}_r$ is computed as the mean position of all vertices in R_r and $\hat{\mathbf{n}}_r$ as the average of the triangle normals weighted by the triangle areas within R_r:

$$\mathbf{c}_r = \frac{1}{|\mathbf{v} \in R_r|} \sum_{\mathbf{v} \in R_r} \mathbf{v} \ , \tag{4.1}$$

$$\hat{\mathbf{n}}_r = \frac{\sum_{t \in R_r} A_t \hat{\mathbf{n}}_t}{\left|\sum_{t \in R_r} A_t \hat{\mathbf{n}}_t\right|} \ , \tag{4.2}$$

where $t \in R_r$ is the set of triangles in region R_r and A_t is the area of triangle t. Since $A_t \hat{\mathbf{n}}_t = \frac{1}{2}(\mathbf{v}_{t1} \times \mathbf{v}_{t2} + \mathbf{v}_{t2} \times \mathbf{v}_{t3} + \mathbf{v}_{t3} \times \mathbf{v}_{t1})$, where $\mathbf{v}_{t1}, \mathbf{v}_{t2}, \mathbf{v}_{t3}$ are the vertices of triangle t, all the terms with respect to the internal edges in Eq. 4.2 cancel out due to the anti-symmetry of the cross product. Thus, we only need to include the terms with respect to each directed boundary edge $e = (\mathbf{v}_{e1}, \mathbf{v}_{e2}) \in \partial R_r$. We therefore use the following formula to compute the normal and area of a region:

$$A(R_r)\hat{\mathbf{n}}_r = \frac{1}{2} \sum_{e \in \partial R_r} \mathbf{v}_{e1} \times \mathbf{v}_{e2} \ , \tag{4.3}$$

where we traverse the boundary in an anti-clockwise direction. To facilitate the stylization transfer in Section 4.5.5, we define an affine transformation $\mathtt{T}_r$ for each region R_r. This will be a composition of two affine transformations:

$$\mathtt{T}_r = \mathtt{T}_r^r \mathtt{T}_r^p \ , \tag{4.4}$$

where $\mathtt{T}_r^r$ defines the rigid transformation implied by the change of $\hat{\mathbf{n}}_r$ to $\hat{\mathbf{n}}'_r$ and

$\mathbf{c}_r$ to $\mathbf{c}'_r$ in the stylization optimization in this section:

$$\mathtt{T}^r_r = [\mathtt{R}(\hat{\mathbf{n}}_r, \hat{\mathbf{n}}'_r) \,|\, \mathbf{c}'_r - \mathtt{R}(\hat{\mathbf{n}}_r, \hat{\mathbf{n}}'_r)\mathbf{c}_r] \ . \tag{4.5}$$

$\mathtt{R}(\hat{\mathbf{n}}_r, \hat{\mathbf{n}}'_r)$ is the rotation matrix transforming $\hat{\mathbf{n}}_r$ to $\hat{\mathbf{n}}'_r$ (see Eq. 2.15). This transformation represents the exaggeration of angles between sculptor's planes.

The transformation $\mathtt{T}^p_r$ scales the affine space in the $\hat{\mathbf{n}}_r$ direction relative to $\mathbf{c}_r$ and represents the planarization of features with a region:

$$\mathtt{T}^p_r(\mu) = \left[\mathtt{I}_3 - \mu\hat{\mathbf{n}}_r\hat{\mathbf{n}}_r^\top \,|\, \mu(\hat{\mathbf{n}}_r \cdot \mathbf{c}_r)\hat{\mathbf{n}}_r\right] \ , \tag{4.6}$$

where $\mu \in [0, 1]$ is a user-defined variable which determines the amount of planarization and $\mathtt{I}_3$ is the 3x3 identity matrix. Under this transformation, all points in space are moved closer to the plane defined by $\hat{\mathbf{n}}_r$ and $\mathbf{c}_r$ along the normal direction. If a point $\mathbf{p}$ is at a distance d from the plane then $\mathtt{T}^p_r(\mu)\mathbf{p}$ will be a distance $(1-\mu)d$ from the plane π_r. Hence, $\mu = 0$ corresponds to the identity transformation and $\mu = 1$ transforms all points in space to lie on the plane.

Note that we can derive all the quantities mentioned above (Eqs. 4.1 to 4.6) directly from vertex positions. Our optimization in the following will be formulated in terms of vertex positions directly or indirectly using the formulas above.

Energy formulation

We pose the stylization of our abstracted mesh as an energy minimization problem. We consider two types of energys in our formulation. The stylization energy exaggerates the angles between adjacent regions while keeping the regions themselves roughly planar; and the regularization energy keeps the solution close to the original abstracted mesh and can also include the Lanteri constraints as suggested by Lanteri [72] and discussed in Section 4.3. The total energy to be minimized is the sum of all stylization and regularization energies:

$$E = E_{style} + E_{reg} \ . \tag{4.7}$$

Stylization energy

The stylization energy consists of two terms for exaggeration between adjacent regions R_i and R_j and planarization within each region:

$$E_{style} = \lambda_d \sum_{R_i \sim R_j} w_{i,j} \hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j + \lambda_f \sum_r \sum_{\mathbf{v} \in R_r} \left(\hat{\mathbf{n}}_r \cdot (\mathbf{c}_r - \mathbf{v})\right)^2 . \tag{4.8}$$

Here, $R_i \sim R_j$ means that region R_i shares a boundary with region R_j. Weights $w_{i,j}$ scale the amount of exaggeration between R_i and R_j. λ_d (d for dihedral angle) controls the overall amount of exaggeration while λ_f controls how flat the regions should be. Minimising the first term of Eq. 4.8 causes exaggeration of the angle between planes since $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j$ decreases as the angle between the vectors (the dihedral angle between planes) increases. The second term minimises the distance of vertices from each region's plane approximation, thereby enhancing the planar abstraction of the mesh.

Initially, the weights $w_{i,j}$ are set as the boundary length between R_i and R_j normalized by the average boundary length. We find that this weighting scheme ensures that each edge's deformation is weighted approximately according to its visual impact on the final model. Our interactive system allows the user to alter these weights by specifying a scale factor $s_{i,j}$ such that: $w_{i,j} \leftarrow s_{i,j} w_{i,j}$.

To make the overall problem easier to solve we can transform the dot product between unit normal vectors $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j$ into an energy with a sum of squares form:

$$\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j = \tfrac{1}{2} \|\hat{\mathbf{n}}_i + \hat{\mathbf{n}}_j\|^2 - 1 \ . \tag{4.9}$$

The resulting sum of squares formulation facilitates the use of efficient solvers such as Levenberg-Marquardt.

Regularization energy

The regularization energy E_{reg} includes the following terms:

$$E_{reg} = E_{area} + E_{edge} + E_{vertex} + E_{normal} + E_{Lanteri}. \tag{4.10}$$

The edge and area terms are as follows:

$$E_{area} = \lambda_a \sum_{i=1}^{K} \left(1 - \frac{A(R_r)}{A^0(R_r)}\right)^2 , \tag{4.11}$$

$$E_{edge} = \lambda_e \sum_{e \in \mathcal{M}_C} \left(1 - \frac{|e|}{|e^0|}\right)^2 , \tag{4.12}$$

where $A(R_r)$ is the area of R_r, $|e|$ is the length of edge e and the sum only includes edges on the borders between regions. The superscript "0" denotes the initial value. These are simple quadratic penalties but by using ratios with the initial value we make the terms scale invariant.

The vertex and normal terms are simple quadratic error metrics:

$$E_{vertex} = \frac{\lambda_v}{2|\bar{e}|^2} \sum_{v \in \mathcal{V}_C} \|v - v_0\|^2 , \tag{4.13}$$

$$E_{normal} = \frac{\lambda_n}{2} \sum_{i=1}^{K} \left\|\hat{\mathbf{n}}_r - \hat{\mathbf{n}}_r^0\right\|^2 , \tag{4.14}$$

where $|\bar{e}|$ is the mean edge length of the mesh used to make the vertex term scale invariant.

All of these regularization terms are needed to avoid various degeneracies of the energy function and maintain closeness to the original mesh, but the results are not critically affected by their weights. We used constant values of λ_v, λ_n, λ_e, and λ_a (given in Section 4.6) for all results shown in this paper.

4.5.3 Lanteri constraints

Based on the writings of Edouard Lanteri, we posit that certain relative measurements on the face are critical to maintain the individual "personality" of the model (Fig. 4.9). Therefore, in an attempt to preserve the recognisability of our results we consider three types of *Lanteri constraints* to formulate the Lanteri energy $E_{Lanteri}$. An absolute position constraint (Eq. 4.15a) ensures that certain points of interest remain at their original positions, a relative position constraint (Eq. 4.15b) ensures that the relative position of two points remains the same, and a relative distance constraint (Eq. 4.15c) ensures a constant distance between two points during stylization. The energy is formulated

(a) Stylized with Lanteri constraints

(b) Original mesh

(c) Stylized without Lanteri constraints

Figure 4.9: *Lanteri constraints help to retain the personality of the original mesh (middle) in the stylized mesh (left). Without the constraints unwanted deformations can occur such as the eyes getting closer together (right).*

---

as a sum of terms of the following form:

$$\sum_{\mathbf{p}} \frac{\lambda_v}{2|\bar{e}|^2} \|\mathtt{T}(\mathbf{p})\mathbf{p} - \mathbf{p}\|^2 \ , \tag{4.15a}$$

$$\sum_{\mathbf{p}_1,\mathbf{p}_2} \frac{\lambda_v}{2} \frac{\|(\mathtt{T}(\mathbf{p}_1)\mathbf{p}_1 - \mathtt{T}(\mathbf{p}_2)\mathbf{p}_2) - (\mathbf{p}_1 - \mathbf{p}_2)\|^2}{\|\mathbf{p}_1 - \mathbf{p}_2\|^2} \ , \tag{4.15b}$$

$$\sum_{\mathbf{p}_1,\mathbf{p}_2} \frac{\lambda_v}{2} \frac{(\|\mathtt{T}(\mathbf{p}_1)\mathbf{p}_1 - \mathtt{T}(\mathbf{p}_2)\mathbf{p}_2\| - \|\mathbf{p}_1 - \mathbf{p}_2\|)^2}{\|\mathbf{p}_1 - \mathbf{p}_2\|^2} \ , \tag{4.15c}$$

where $\mathtt{T}(\mathbf{p})$ is the affine transformation for the region containing $\mathbf{p}$, defined using Eq. 4.4. The specific feature points used to formulate these constraints are given in Appendix F. The positions of these features on the mesh are currently manually annotated by the user. However, this could easily by automated by using the template model.

4.5.4 Energy minimization

To find the minimum of Eq. 4.7 the energy is written as a function of the vertex positions (making use of Eqs. 4.1 and 4.3) and minimized using the Levenberg-Marquardt method in Ceres Solver [2]. Any vertices on an open boundary of the mesh are kept constant during the optimization. Standard thresholds on function tolerance and gradient tolerance are used to determine convergence.

Figure 4.10: *Varying levels of stylization smoothness can enhance or smooth the boundaries between sculptors' planes while the orientation of the planes remains the same. We have deliberately applied excessive planarization to emphasize the effect. The segmentation of the planes of the head model is clearly visible.*

4.5.5 Stylization Transfer

The final step is to transfer the stylization from the abstracted mesh to the high resolution mesh. To do this we make use of affine transformations combined with skinning weights. This approach allows the implementation of Lanteri constraints and planarization terms, as well as allowing interactive use.

When doing the stylization transfer we need to ensure smooth transitions between different regions to avoid any visible artefacts. We define a set of skinning weights $w_{i,r}$ for each vertex $\mathbf{v}_i$ with respect to each region R_r. The final affine transformation $\mathtt{T}(\mathbf{v}_i)$ for each vertex $\mathbf{v}_i$ of the input mesh is defined as the weighted average of all relevant $\mathtt{T}_r$:

$$\mathtt{T}(\mathbf{v}_i) = \sum_{r=0}^{K} w_{i,r}\mathtt{T}_r, \tag{4.16}$$

where we define $\mathtt{T}_0$ to be the identity transformation which is assigned to the region outside of the optimization area to enable smoothing over this boundary. We initialize the skinning weight $w_{i,r}$ to be the number of faces in R_r that contain vertex $\mathbf{v}_i$, and then normalize so that $\sum_r w_{i,r} = 1$. For all vertices not on a region border there will only be a single non-zero weight.

Our system allows the user to adjust the stylization smoothness across different regions. Figure 4.10 shows one example where the user creates differ-

ent stylization smoothness using our interactive user interface. To do this we apply iterations of Laplacian smoothing to the original, un-smoothed skinning weights:

$$w_{i,r} \leftarrow \frac{1}{|\mathbf{v}_i \sim \mathbf{v}_j|} \sum_{\mathbf{v}_i \sim \mathbf{v}_j} w_{j,r}, \tag{4.17}$$

where $\mathbf{v}_i \sim \mathbf{v}_j$ is the set of vertices $\mathbf{v}_j$ adjacent to $\mathbf{v}_i$. The user can increase the amount of smoothing by increasing the number of iterations.

Applying a large number of iterations of Laplacian smoothing on a high resolution mesh can be slow. The naive approach is to recompute the smoothed weights every time the user adjusts the slider controlling the number of iterations. However, this is slow and very wasteful. Therefore, we make use of a stack of weight matrices that represent increasing levels of smoothness and interpolate in between these levels to allow a fast evaluation on a continuous domain of smoothness.

We pre-compute the stack of skinning weights $\{w_{i,r}^l\}_{l=0}^M$ with different levels of smoothing (Fig. 4.11). $w_{i,r}^0$ represents the original, un-smoothed skinning weights and $w_{i,r}^l$ represents 2^l iterations of Laplacian smoothing. We increase the iterations as a geometric sequence because this better represents the visual change of the result. Given an arbitrary smoothing scale s we compute $w_{i,r}^s$ by linearly interpolating between between the two closest layers of skinning weights $w_{i,r}^l$ and $w_{i,r}^{l+1}$ in the stack, where l is the layer with the highest number of smoothing iterations such that $l \leq s$.

We extend the method by allowing the user to interactively specify different amounts of smoothing for different areas of the mesh. The level of smoothing must be continuously varying over the mesh to avoid visual artefacts. To accomplish this at interactive rates, we adapt a method for controlling spatially varying blur from Diffusion Curves [94]: The user selects a boundary $\{i, j\}$ between regions R_i and R_j and assigns a smoothing scale s. All vertices on the boundary $\{i, j\}$ are assigned the smoothing value s. Any other region boundaries emanating from the ends of the $\{i, j\}$ boundary have their values interpolated along the boundary, as in Fig. 4.12. The system then propagates the smoothing scale into all relevant regions by solving a Laplace equation with Dirichlet boundary conditions:

$$\nabla^2 s = 0 \ , \tag{4.18}$$

where the Laplacian operator is discretized by the cotangent weights (Sec-

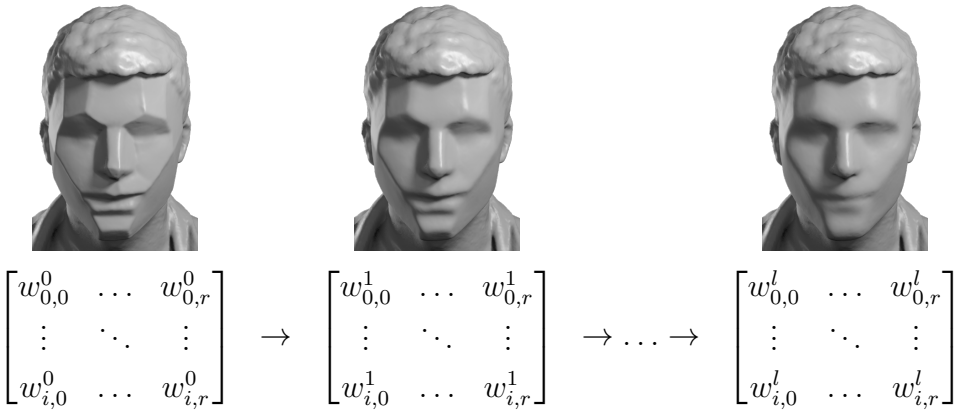


Figure 4.11: *A stack of weight matrices is precomputed to efficiently compute skinning weights for varying levels of smoothness. We use linear interpolation to set the weights for levels of smoothness lying in between layers of the stack. Each layer of the stack has the same amount of smoothing applied to the whole mesh but we allow the smoothness to vary over the mesh, as set by the user.*

---

tion 2.10.2). To do this, we write the Laplace operator as a matrix multiplication and split the set of vertices into those on the border and those inside a region:

$$\mathtt{L}\mathbf{s} = \begin{bmatrix} \mathtt{A} & \mathtt{B} \\ \mathtt{B}^T & \mathtt{C} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \mathbf{0}, \tag{4.19}$$

where $\mathtt{L}$ is the sparse Laplacian matrix consisting of cotangent weights on edges of the mesh, $\mathbf{s}$ is the vector containing the smoothing scale at all vertices in a region. $\mathtt{L}$ is split into a block-matrix where $\mathtt{A}$ represents the edges which are all internal to the region, $\mathtt{B}$ contains edges joining the boundary vertices to internal vertices and $\mathtt{C}$ contains all boundary edges. $\mathbf{s}$ is split into two parts $\mathbf{x}$ and $\mathbf{y}$ representing the smoothing scale at the internal vertices and the boundary vertices respectively. $\mathbf{y}$ represents the Dirichlet boundary conditions and, hence, is fixed. The unknowns are $\mathbf{x}$. To solve we rearrange Eq. 4.19 to obtain the following equation:

$$\mathtt{A}\mathbf{x} = -\mathtt{B}\mathbf{y} \quad \Longrightarrow \quad \mathbf{x} = -\mathtt{A}^{-1}\mathtt{B}\mathbf{y}. \tag{4.20}$$

This system is solved using a sparse Cholesky decomposition of $\mathtt{A}$ and the solution is used to look up the smoothed weights from the skinning weight stack. Since $\mathtt{A}$ is constant for a given mesh we can pre-compute the Cholesky

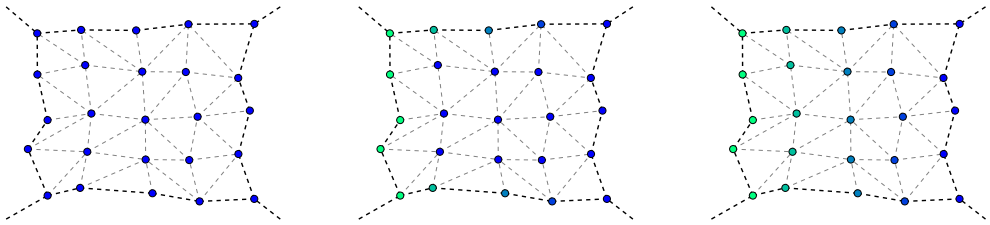


Figure 4.12: *The smoothing weights are generated by solving the Laplace equation with Dirichlet boundary conditions. First (left) the mesh is initialized with all weights the same. Next (middle), the user selects the region boundary on the left side and changes the weight. The weights are linearly interpolated along adjacent region boundaries. Finally (right), we solve the Laplace equation to get weights for the vertices internal to the region. The result is a smooth transition of the weights.*

---

decomposition. It is then very fast to compute the new values $\mathbf{x}$ when the boundary conditions $\mathbf{y}$ are updated by the user.

4.5.6 Interaction

Given a new input mesh, we first do a rough manual alignment with the planes-of-the-head template model which is then refined by the non-rigid alignment of Li *et al*. [75]. The segmentation is transferred from the template to the aligned input mesh on a nearest neighbour basis, after which the abstracted mesh is generated. Lanteri constraint points are manually annotated by the user clicking on specified parts of the input mesh. These pre-processing steps do not run at interactive rates but only need to be done once for a given input mesh. From now on, our full optimization framework for abstracted mesh stylization (Section 4.5.2) and stylization transfer (Section 4.5.5) runs at an interactive frame-rate of 10Hz on a standard desktop workstation. Our GUI (Fig. 4.13) provides users with intuitive controls for the amount of exaggeration, the amount of planarization and transform smoothness. Each of these can be adjusted globally, or for more detailed control, exaggeration and smoothness amounts can be adjusted for each edge (Fig. 4.14) and planarization amounts can be adjusted for each face. These correspond to intuitive visual changes, modifying the visual contrast between regions, displayed live in the form of the

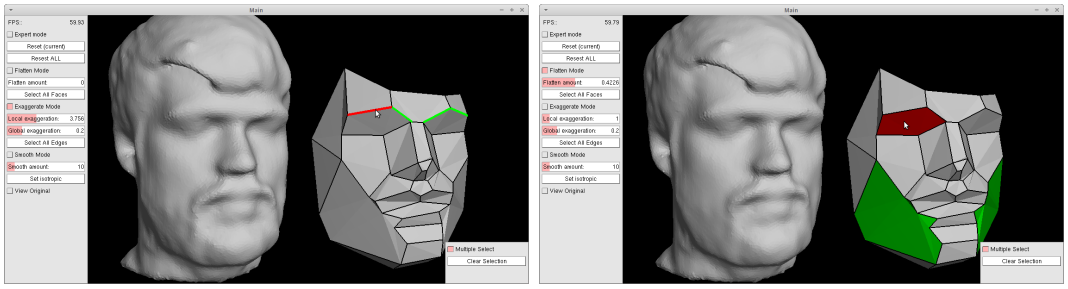


Figure 4.13: *A user can adjust intuitive exaggeration and smoothing terms for specific edges (left) or planarization terms for specific faces (right) of the abstracted mesh. The resulting deformations are shown interactively on the full resolution mesh.*

Figure 4.14: *Our system allows for local as well as global controls. Here, we show increasing local exaggeration of the brow.*

---

full resolution deformation result. Please see Appendix A for a screen-captured demonstration.

4.6 Results

The input triangular mesh can come from any source. The only requirement is that the areas to be abstracted are connected, manifold and contain no holes. We tested our system on meshes from two different sources; scans we made of a number of volunteers using the method of KinectFusion [87], and meshes obtained from a state-of-the-art multi-view stereo reconstruction system [7].

The deformations produced by our method typically enhance cheekbones, deepen eye sockets, and emphasize the brows, chin and lips (see Fig. 4.15). This approach is especially effective for the KinectFusion scans, which are sometimes lifeless and hard to recognize due to their lower fidelity. Further, materials used

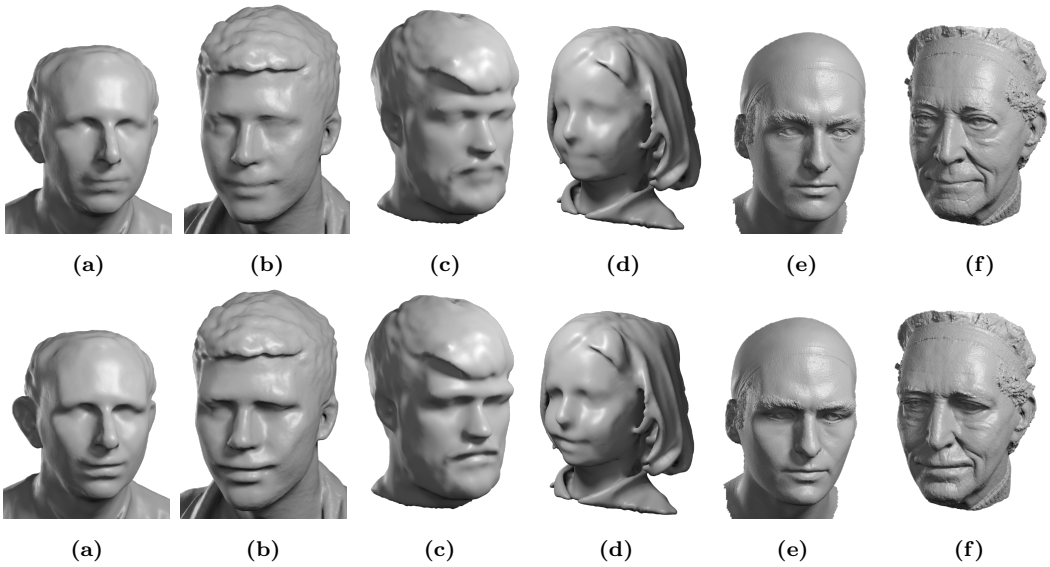


Figure 4.15: *(a)–(d) KinectFusion scans; (e),(f) scans from Beeler* et al. *[7]. Top row: input; bottom row: output.*

---

in 3D printing can be more translucent than skin, giving a blank look to printed busts. Our method gives such busts stronger definition by accentuating facial features, often making them more recognizable (see Fig. 4.16).

Due to the interactive nature of the system, the user is easily able to adjust the amount of stylization (Fig. 4.17), planarization (Fig. 4.18) and smoothing (Fig. 4.10) both globally and locally to their liking. Once these local and global weights have been set by the user they can be stored and transferred to other meshes with the same segmentation structure, as seen in Fig. 4.19. Hence, it would be possible to present the user with a selection of pre-set styles as a starting point for their own adjustments.

We are not aware of any other methods which attempt what we call sculptural stylization. The closest methods we can find to compare against are those which do curvature-aware detail exaggeration/sharpening on meshes. Figure 4.20 shows a comparison against the method of PriMo [13] and exaggeration using the screened Poisson equation [19]. We can see that our method allows a wider range of custom stylizations, better preserves recognisability and, due to our anatomically based model, can apply deformations which are more in-keeping with the human facial structure. The results of the other two methods show quite similar stylizations.

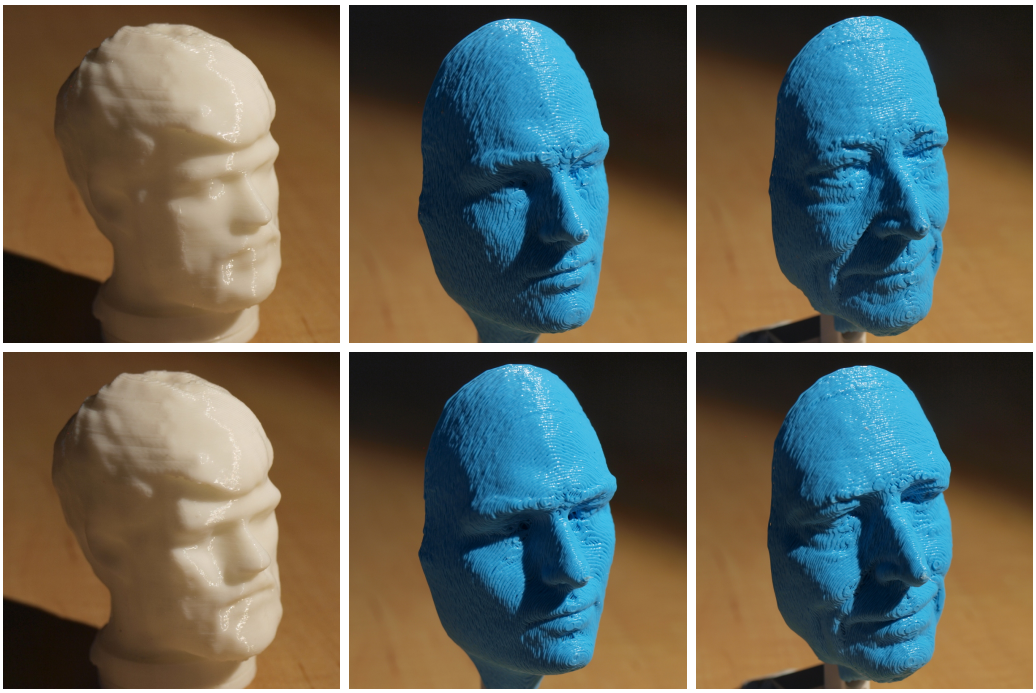
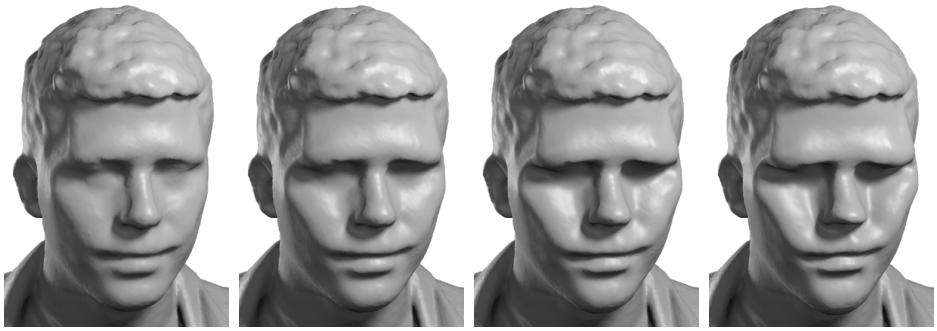


Figure 4.16: *3D printed busts — original (top) vs. stylized (bottom).*



$\lambda_d = 0.5$ $\lambda_d = 1.0$ $\lambda_d = 1.6$ $\lambda_d = 2.3$

Figure 4.17: *Increasing amounts of stylization, from left to right. We intentionally push the exaggeration too far to demonstrate how the Lanteri constraints keep salient features in correct positions.*

Figure 4.18: *Increasing planarization (left to right)*

Figure 4.19: *A style can be transferred to other meshes by using the same set of weights. Here, each row shares the same set of weights.*

Our method can also be applied to models other than faces by using VSA for abstraction. In this case, the models are often thinned, and concavities become more pronounced (see Fig. 4.21). The deformations shown in these figures are subtle, by design, and may be hard to see clearly in the printed thesis. We direct the reader to view the electronic version, and especially to the video (Appendix A), which demonstrates the abstractions more clearly by cutting rapidly between inputs and outputs. One drawback of the VSA approach is that exaggeration edges can be created in areas in which the user does not want any exaggeration. It is then up to the user to manually change the weights of these edges to prevent unwanted deformations.

For the stylization of the abstracted mesh (Section 4.5.2), the optimization is typically performed over a small number ($n < 100$) of vertices on the coarse mesh, and thus takes only a fraction of a second, regardless of the input scan resolution. The stylization transfer step (Section 4.5.5) works with the full resolution mesh, but since only local linear solves and linear operations are required, both steps together can be run interactively. For an input mesh with 30k vertices our system runs at 10Hz on a Linux laptop, including all rendering, with room for further optimisation. For one of the Beeler *et al.* meshes containing 375k vertices the system runs at about 2Hz. Currently the system runs entirely on the CPU, however, the stylization transfer step would be well suited to GPU acceleration.

Most of the weight parameters described in Section 4.5.2 and Section 4.5.5 are held constant for all examples: $\lambda_a = 10$ and $\lambda_e = 4$, $\lambda_v = 60$, $\lambda_n = 1$, and $\lambda_f = 1$. The default exaggeration weights yield pleasing results with the user just specifying $0 \leq \lambda_d < 3$ to control the overall amount of deformation. However, the user can customize the output by specifying per-edge exaggeration and smoothing weights and per-face planarization weights.

A video (Appendix A) and a selection of the results (including original, segmented, abstracted and stylized meshes) are made available on the project webpage:
http://wp.doc.ic.ac.uk/robotvision/project/face-stylization/

4.7 Discussion and Conclusion

Experts in sculpture will note that both the approach and algorithm we have chosen are both oversimplifications of a sculpting process. Our interpretation of the planes of the head, inspired by Lanteri and Asaro, is only one abstraction used by sculptors, and our algorithm is only one possible implementation of this abstraction. We hope and expect that future researchers will explore alternate interpretations and methods with us. We expect that much further insight will be gained by extending the initial comparative study of the geometrical properties of real versus sculpted faces which we present in our supplementary material.

The stylizations achievable on busts is clearly dependent on the segmenta-

tion model used. By using a number of different template models and segmentations it should be possible to obtain a wide variety of abstractions/stylizations.

Both Nakpil and Magrath described working at multiple scales in their sculpting work, from large planes to smaller forms. In this paper we have utilized only two scales, but combining deformations at multiple scales may produce a wide variety of sculptural styles.

One significant limitation of our approach is that it works best on surfaces with a mixture of convex and concave regions. (As a trivial example, it is impossible to uniformly decrease all the angles of a convex polygon, because the sum of the internal angles is constant.) Further, our method is largely *ad hoc*, based on conversations with sculptors and our analytical interpretation of their intent. But we believe that given an appropriate dataset, it may be possible to learn sculptural abstraction in a more principled way from real data.

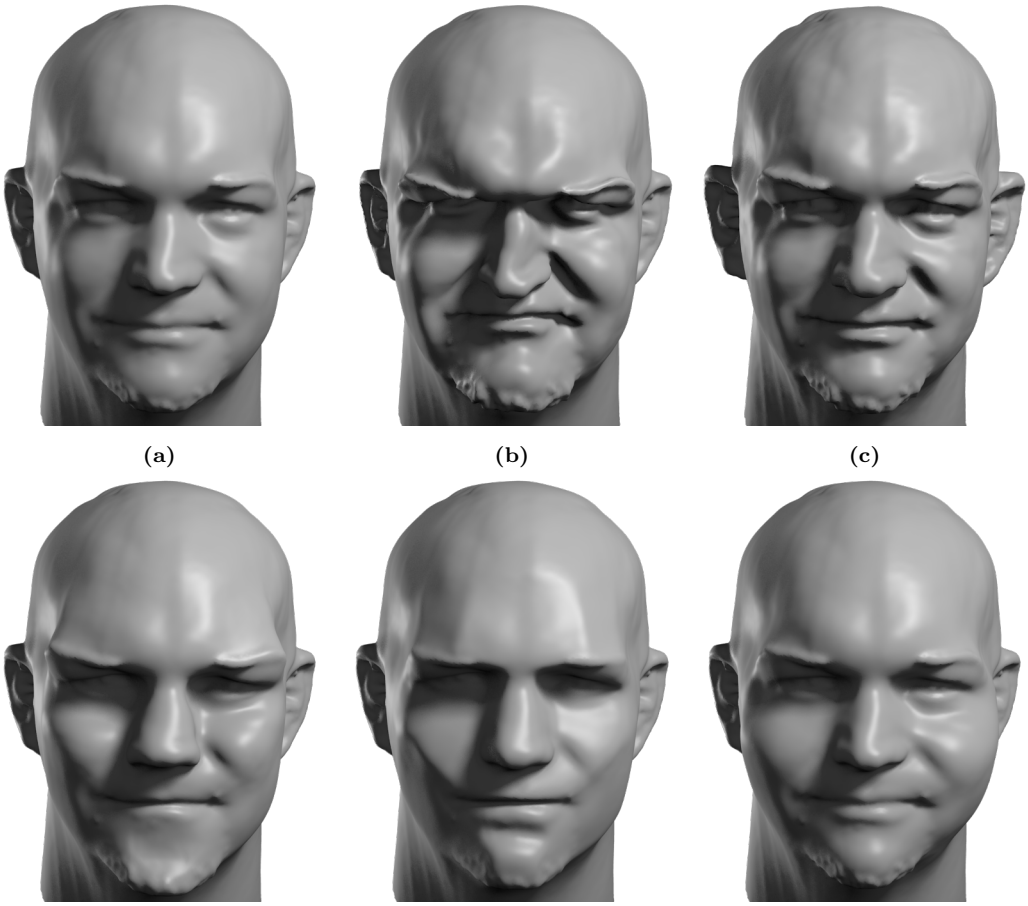


Figure 4.20: *(a) original mesh, (d)-(f) results from our system, (b) result from PriMo [13], (c) result using the screened Poisson equation [19]. Our system is capable of a wider range of custom stylizations: (d) bulbous nose, sharpened chin and accentuated brow, (e) planarization with low exaggeration, (f) rounder face, larger cheeks.*

Figure 4.21: *Original mesh (left), segmentation (middle), and stylized mesh (right). For these non-bust meshes we use Variational Shape Approximation [20] to segment the mesh into sculptors planes based purely on geometry. Note the emphasis of facial features on the bunny (Mesh source: [119]) and sharper definition of the eye sockets on the lion head (Mesh source: [29]).*

5

Real-time Depth from Stereo

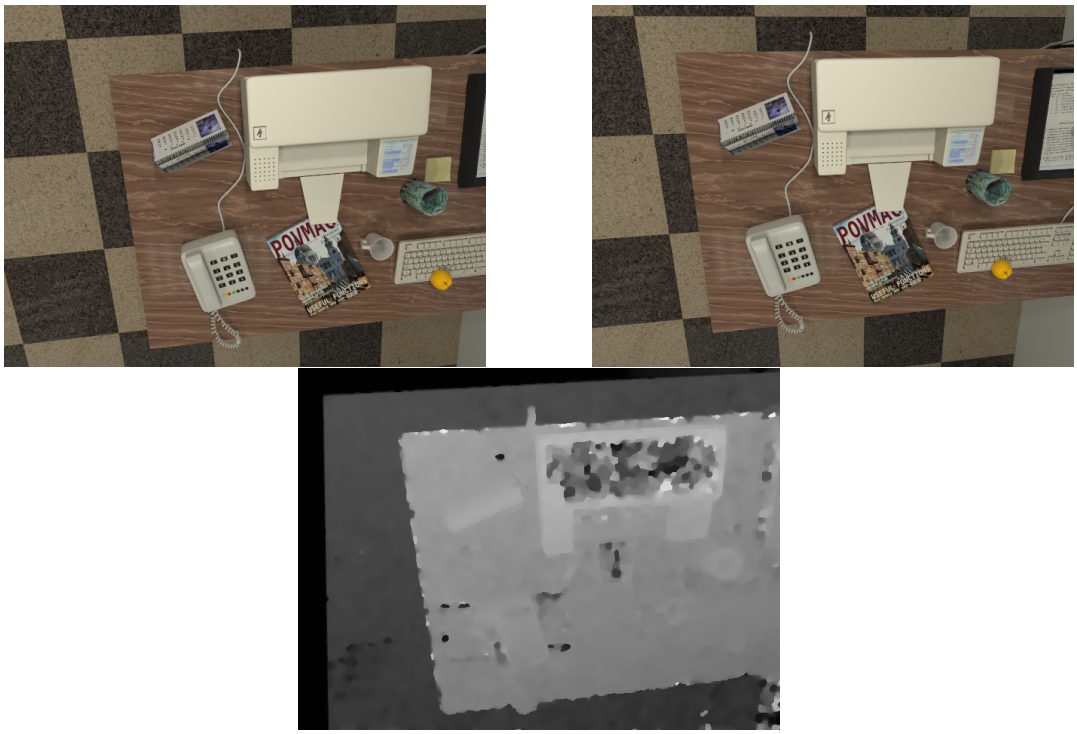


Figure 5.1: *In stereo, two (or more) images (top row) are used to recover the geometry of the scene in the form of a depth-map (bottom row). Lighter pixels indicate that the surface is closer to the camera.*

Following the work in Chapter 3, we start work on trying to expand planar surface light-field capture to general 3D surfaces. A first step towards this goal is to recover the 3D geometry from a monocular moving camera. To do this we use a depth-map fusion approach, outlined in Chapter 6. The goal of this chapter is to investigate fast and efficient ways to generate the required depth-maps from a set of tracked frames from a monocular camera (i.e. the relative poses between camera frames are known).

In the computer vision world, "stereo" is the process of recovering geometry

(usually represented as depth or disparity maps) from two (or more) images (Fig. 5.1). Quite often, stereo algorithms will assume a *rectified* stereo pair — this means that the cameras have been aligned so that the transformation between the two only consists of a translation in the x-direction, the length of which is known as the *baseline*. In this situation, we will often solve for *disparity*: given a point (u, v) in the left image then the disparity d is defined so that the 3D point observed at that pixel projects to position $(u-d, v)$ in the right image. We can then find the depth D of that pixel as $D = \frac{fB}{d}$, where f is the focal length and B is the baseline. This type of rectified configuration is clearly not suited to pairs of images taken from a hand-held monocular video feed, the relative transformation is unconstrained. Additionally, since we have a full 30fps video feed there is no reason to limit ourselves to only using two frames to estimate a depth-map. Therefore, in this chapter we will consider non-rectified, multi-view, short baseline stereo; as this is the type of data available in a monocular video stream. Our focus will be on algorithms which have the potential to run at real-time rates so that they can be used for online 3D reconstruction, such as the system outlined in Chapter 6.

In Section 5.1 we will discuss existing stereo methods and some of their strengths and limitations. In Sections 5.2 to 5.4 we will give an overview of some common high-performance algorithms applied to the stereo problem. In Section 5.5 we will take a close look at the PatchMatch Stereo algorithm and how it can be used for non-rectified, multi-view stereo. PatchMatch Stereo and its variants have managed to obtain state-of-the-art accuracy on the Middlebury Stereo Datasets [111, 112, 113, 114]. However, these datasets are mostly focused on accuracy rather than speed and efficiency. For robotic and augmented reality applications, real-time constraints force us to use the fastest algorithm possible but we still strive to retain the highest quality results. To this end, in Section 5.6 we will take the high accuracy PatchMatch Stereo algorithm and investigate ways to optimise and speed up convergence in the hope of obtaining a high quality stereo algorithm that can run at interactive rates.

One of the key points that makes PatchMatch Stereo unique and is key to its performance is the use of slanted patches. In Section 5.7 we do some analysis on when the use of slanted patches will actually produce a worthwhile increase in quality. In some cases the effect is minimal.

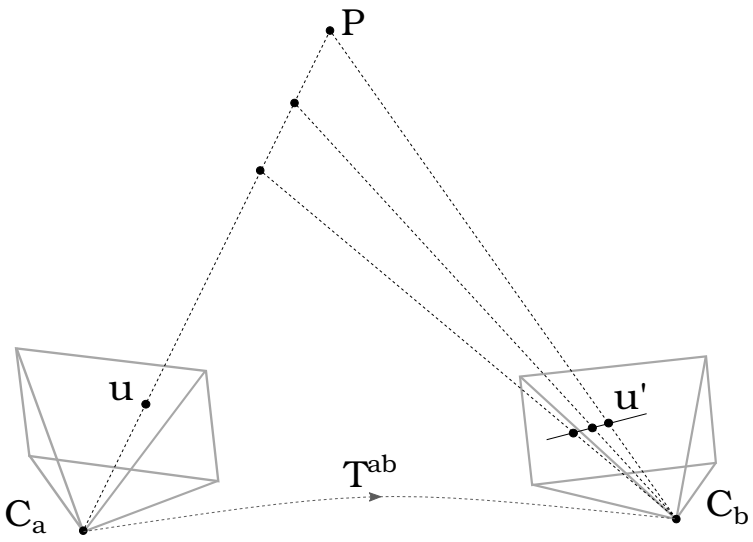


Figure 5.2: *The stereo correspondence problem. We wish to find the position of a 3D point* $\mathbf{P}$ *which lies along the ray emanating from pixel* $\mathbf{u}$ *in camera* C_a*. All points along the ray project onto the* epipolar line *in camera* C_b*. By estimating the position of* $\mathbf{u}'$ *along the epipolar line, corresponding to the projection of* $\mathbf{P}$*, we can solve for the position of* $\mathbf{P}$*.*

5.1 Background

Stereo is a correspondence problem; we wish to find pixels in two (or more) images which correspond to the same 3D point, as seen in Fig. 5.2. In general we do this by assuming that the point will look the same in both images and so minimise an error which is the difference of the two pixels, known as the *photometric error*:

$$\mathbf{u}' = \arg\min_{\mathbf{v}\in\mathcal{L}} \big(I_a(\mathbf{u}) - I_b(\mathbf{v})\big), \tag{5.1}$$

where I_a is the image in camera C_a, as shown in Fig. 5.2, and $\mathcal{L}$ is the set of points on the epipolar line. To reduce ambiguity causing incorrect correspondences we often compute the photometric error by summing over a patch, as we will demonstrate in Section 5.5.1.

In the most simple case we can just sample a discrete number of depths, or points on the epipolar line, and take the one with the lowest cost as the solution. This simple method is referred to as *plane-sweep* or *cost-volume* stereo and is explained further in Section 5.2.

Stereo is very closely related to the optical flow problem. Just like stereo, optical flow finds corresponding pixels between two images but without the constraint that the correspondence must lie on the epiploar line. In fact, some works use optical-flow as a precursor to estimating geometry: In [86], Newcombe and Davison make use of real-time optical flow combined with sparse structure from motion to estimate depth for live dense reconstruction. Valgaerts *et al.* [125] jointly solve for depth and the fundamental matrix by first estimating the optical flow and then iteratively incorporating an epipolar constraint.

Another way to reduce bad correspondences and improve the overall quality of the result is to incorporate regularisation or smoothness constraints. There are a number of stereo methods which make use of Total Variation style optimisations as outlined in Section 2.5.1. This is achieved by minimising an energy consisting of a sum of a regularisation term and a photometric data term. Due to linearisation of the function during minimisation a good initial estimate is required. Therefore, the initial guess is either estimated from some other method, as in [85, 88], or a coarse-to-fine method is used [102]. Ranftl *et al.* [102] make use of second order Total Generalised Variation (TGV) [14] in their stereo system. This higher-order regularisation allows for reconstruction of piecewise-planar surfaces, rather than the first-order total variation which favours piecewise-constant solutions. Newcombe [85] makes the claim that given a tight computational budget, the increased processing time of TGV is not worth it given the only slight increase in accuracy over using a Huber regularisation term. More recently, Graber *et al.* [49] present a novel regularisation term which attempts to smooth the 3D surface rather than just the depth-map. They present a way to efficiently solve in a primal-dual framework and show how their approach does not exhibit the stair-casing of classic TV. However, they do not make a comparison to second-order TGV which also does not exhibit stair-casing.

There are now a large number of stereo algorithms based on the PatchMatch [6] randomised correspondence algorithm. The original, PatchMatch Stereo [12], over-parameterises depth by fitting a 3D plane per pixel. This allows for effective reconstruction of slanted planes and PatchMatch provides an efficient optimisation framework (we give an overview of the PatchMatch Stereo algorithm in Section 5.5.1). PM-Huber [57] extends that work by adding Huber regularisation. To optimize the non-convex energy they alternate between it-

erations of PatchMatch Stereo and primal-dual based gradient ascent/descent. Another approach to regularization with PatchMatch is to use belief propagation, as in PMBP [9]. Using methods from particle belief propagation they are able to add a pairwise energy to the otherwise unary energy of PatchMatch Stereo. Their approach is effective but requires a number of particles per node which increases computation and is a hindrance in real-time scenarios. Uh *et al.* [124] use PatchMatch in a multi-view stereo setting — reconstructing a full geometric model rather than just a depth map. The reconstruction is represented by a set of oriented patches in 3D which are projected into the camera coordinates for efficient propagation. PatchMatch Filter [77] combines PatchMatch Stereo with efficient edge-aware filtering, which has already been shown to be effective for the stereo correspondence problem [106]. They make use of a superpixel segmentation to achieve this and also propose an extension of the PatchMatch algorithm to achieve propagation at the level of superpixels, potentially speeding up propagation and convergence over larger areas. MonoFusion [101] uses a cut-down version of PatchMatch Stereo which only estimates depth. They also only perform a single iteration of scanline propagation. This simplification allows real-time performance but the quality of the depth-maps is greatly reduced.

Heber and Pock [56] detail a novel approach to multi-view stereo where each image is compared to every other image in the set. This is in contrast to many of the algorithms detailed above which only compare to a single reference image. Two images which are not the reference image are never directly compared. They make use of densely acquired data from light-fields and use robust PCA methods to solve the stereo problem. This interesting approach could be a step towards joint geometry and lighting estimation using light-field cameras as hinted at in Chapters 3 and 6.

One of the major difficulties of passive stereo methods is recovering the geometry of textureless regions. The use of very large image patches can help up to a point but seriously increases the processing time. Some recent works have attempted to incorporate higher level priors to help deal with textureless regions. Pinies *et al.* [97] make use of non-local total generalized variation (an extension of total variation which enables pairwise terms between any two nodes) to incorporate a planar constraint on textureless regions. This approach essentially propagates depth from the edge of the textureless area and so struggles if the edges are not seen in the image. [22, 24] make use of a superpixel

segmentation and novel superpixel matching techniques to fit planar geometry to textureless regions. DPPTAM [23] combines this superpixel approach with semi-dense SLAM to create real-time dense reconstructions on the CPU.

Recently the rapid upsurge in deep learning and Convolutional Neural Networks (CNNs) has led to learning based approaches to stereo and related problems. DeepStereo [44] trains a CNN to predict new views from a set of posed views. The input to the CNN is similar to a cost-volume, each reference image being projected into the virtual camera at a discrete set of depths. This means that the network does not need to learn the complex camera projection and rigid transformation operations. Such a system could also be used to generate depth-maps, but the main obstacle is obtaining enough training data to make this possible. FlowNet [43] applies CNNs to the problem of optical flow. Due to insufficient amounts of real-life training data they generate synthetic training data by rendering "Flying Chairs" over random images from Flickr as the background. The synthetic training data is good enough to achieve state-of-the-art accuracy among real-time optical flow methods. Wang *et al.* [127] use CNNs to predict surface normals from just a single image. This prediction can be used as an additional scene prior for stereo as discussed in [25].

5.2 Cost-volume

In this section we will give a brief overview of cost-volume stereo, a relatively common approach we will refer to in later sections.

Given a reference image, a discrete set of depth hypotheses is created and the photometric cost is evaluated at each depth for each pixel by projection into one or more other images. Hence, a three-dimensional *cost-volume* is effectively created.

In the simplest case we just take the minimum cost depth-hypothesis per-pixel. If this is all that is required then we do not need to store then entire volume, we can just store the current best depth hypothesis and its cost as we loop over the possible depth values. This is often referred to as *plane-sweep* stereo.

Generally there is a fixed minimum and maximum depth, D_{min} and D_{max} respectively, and discrete depth samples come from a uniform sampling in in-

verse depth between these values. We make the sampling independent of the depth range by using the parameter $\xi \in [0, 1]$ defined by:

$$\xi = \frac{\frac{1}{D} - \frac{1}{D_{max}}}{\frac{1}{D_{min}} - \frac{1}{D_{max}}} \quad \Longrightarrow \quad D = \frac{1}{\left(\frac{1}{D_{min}} - \frac{1}{D_{max}}\right)\xi + \frac{1}{D_{max}}}. \tag{5.2}$$

For a cost-volume with B depth bins, the discrete depth hypotheses will be defined by $\xi_i = \frac{i}{B-1}$ for $i = 0, ..., B-1$.

Let $\mathcal{I}$ be the set of images that will be incorporated into the cost-volume. Then the cost at pixel $\mathbf{u}$ and inverse depth ξ is given by:

$$C_{\mathbf{u}}(\xi) = \frac{1}{|\mathcal{I}|} \sum_{I_k \in \mathcal{I}} \Psi(I_k, \mathbf{u}, \xi), \tag{5.3}$$

where Ψ is a function representing the photometric error. For example, we could use the absolute pixel error:

$$\Psi(I_k, \mathbf{u}, \xi) = \left| I(\mathbf{u}) - I_k\big(\mathbf{u}_k(\xi)\big) \right|, \tag{5.4}$$

where I is the reference image and the reprojected pixel position $\mathbf{u}_k(\xi)$ is given by Eq. 2.30. Generally for a cost-volume the photometric error is computed using just single pixels but we could also use patches in this error.

Once this cost-volume has been created we take the minimum cost depth per pixel to obtain the depth-map. Due to the discrete sampling of the cost-volume, the depth values obtained will always lie at the sampling points. To improve on this we can perform sub-pixel refinement by fitting a quadratic to the cost function about the minimum. Let ξ_i be the minimum cost inverse depth at pixel $\mathbf{u}$. The sub-pixel refined inverse depth is given by:

$$\xi^* = \xi_i - \frac{C'_{\mathbf{u}}(\xi_i)}{C''_{\mathbf{u}}(\xi_i)} \tag{5.5a}$$

$$= \xi_i - \frac{C_{\mathbf{u}}(\xi_{i+1}) - C_{\mathbf{u}}(\xi_i)}{C_{\mathbf{u}}(\xi_{i+1}) - 2C_{\mathbf{u}}(\xi_i) + C_{\mathbf{u}}(\xi_{i-1})} \xi_{step}, \tag{5.5b}$$

where we have used the forward difference to compute the first derivative of the cost and central difference for the second derivative.

One of the advantages of the cost-volume approach is that many frames can be combined into the cost-function iteratively, as made use of by Newcombe *et al.* [88] in their real-time dense tracking and mapping system.

Some methods apply filtering or regularization techniques to the volume to reduce the level of noise in the solution. DTAM [88] uses variational denoising methods (Section 2.5.1) to obtain a regularized solution. Rhemann *et al.* [106] use efficient, edge-aware filtering techniques to average samples over each slice before taking the per pixel minimum. This is almost equivalent to having a patch based error per pixel in Eq. 5.3 but their formulation means that they can make use of fast, efficient filtering algorithms.

5.3 Coarse-to-Fine Stereo

As an alternative method for comparisons in Chapter 6 we describe an extremely efficient coarse-to-fine approach to stereo which is capable if running well over 30fps on modern GPUs.

We generate an image pyramid with L levels for each frame. Level 0 of the pyramid is the original image and level $l+1$ is the downsampled version of level l. Hence, if level l has dimension $M \times N$ then level $l+1$ has dimension $\frac{M}{2} \times \frac{N}{2}$. We perform down-sampling by computing the mean of the four-pixel neighbourhood. This means that the camera centre of projection stays in the same place, we just scale it by a factor of 0.5 (the focal length is also scaled by 0.5). Some other down-sampling techniques, for example, blurring and sub-sampling, can result in a translation of the camera centre which would need to be accounted for in the camera calibration matrix.

Once the pyramid has been built we perform plane-sweep stereo on the coarsest level of the pyramid. Since the resolution is very low this is computed very quickly. Starting with the coarsest level depth-map we now traverse the pyramid. Given the depth-map for pyramid level $l > 0$ we compute the depth-map on level $l-1$ by up-sampling the lower resolution depth-map and sampling s points evenly distributed about that point. Let ξ_{step} be the inverse depth step which moves a distance of one pixel along the epipolar line (estimated by dividing the length of the epipolar line by the inverse depth range). If Ξ_l is the inverse depth map for level l of the pyramid then the sampled depth points for pixel $\mathbf{u}$ on level $l-1$ are:

$$\left\{ \Xi_l \left(\left[\frac{u}{2} \right], \left[\frac{v}{2} \right] \right) + \left(k - \frac{s-1}{2} \right) \xi_{step} \,\middle|\, k = 0, ..., s-1 \right\}. \tag{5.6}$$

With both the plane-sweep on the coarsest level and the depth-sampling on the subsequent levels of the pyramid we perform sub-pixel refinement in the same way as described in Eq. 5.5 (this requires $s \geq 3$ in the up-sampling steps).

There are three main parameters which define the behaviour of this stereo method: the number of pyramid levels L, the number of depth samples when up-scaling s, and the size of the patches used to compute the photometric error. If L is small the plane-sweep step takes longer because it is working at a higher resolution. However, if L is too large and s is small we are more likely to get stuck in a local minimum and miss some fine geometric details, a common limitation of coarse-to-fine methods.

Table 5.1 shows some timings and errors for a variety of parameters of this method, tested on the synthetic dataset outlined in Section 2.11.1. We see that, as expected, the computation time starts to go down as the number of pyramid levels goes up. This is because the number of photometric cost computations on the coarsest level goes down. However, once we pass four levels of the pyramid the runtime starts increasing again due to the extra computation. Obviously the runtime increases as we increase the number of samples but a small number of samples gives a large error. Once the number of samples is greater than two the error reduces dramatically due to the ability to perform sub-pixel refinement. From the data in the tables we see that four levels and four samples per level appears to be the best compromise between runtime and accuracy.

5.4 Temporal Consistency Check

Most high accuracy binocular stereo algorithms use left-right consistency checks to remove outlier depth estimates. This works by generating a depth map for both the left and right images and comparing the depths of corresponding pixels. If the difference in depth is greater than a threshold the pixel is removed.

This approach is very effective but immediately doubles the amount of computation required since two depth-maps must be generated. In an interactive setting this is very wasteful. Instead, we propose a temporal consistency check which is applicable to real-time stereo methods and monocular sequences.

Given a new depth-map and a reference depth-map (from a previous point in time) we project each pixel into the reference depth-map's coordinate frame

# Levels \ # Samples	1	2	3	4	5
1	38	38	38	38	38
2	7.0	7.1	8.0	8.7	9.4
3	3.3	3.4	4.7	5.5	6.3
4	2.8	3.0	4.3	5.1	6.0
5	2.9	3.1	4.5	5.3	6.3
6	3.0	3.2	4.6	5.5	6.5

(a) Timings in ms

# Levels \ # Samples	1	2	3	4	5
1	0.069	0.069	0.069	0.069	0.069
2	0.049	0.091	0.055	0.053	0.056
3	0.080	0.10	0.048	0.049	0.050
4	0.15	0.15	0.063	0.047	0.049
5	0.30	0.31	0.073	0.051	0.055
6	0.34	0.51	0.093	0.060	0.049

(b) Mean absolute depth error

Table 5.1: *Comparison of the time taken and the mean absolute depth error for the coarse-to-fine stereo method when varying the number of pyramid levels and the number of samples when up-scaling. Analysis done on the dataset in Section 2.11.1 with a patch-size of* 5×5*. Timings for an NVIDIA GTX 780 Ti GPU.*

---

and compare the depth to the depth-map values of the four nearest pixels (if the point projects to floating-point pixel location $\mathbf{u}$ then we look at the four integer pixel locations surrounding $\mathbf{u}$). If the depth-error of at least one of those pixels is less than a threshold we mark the pixel as consistent.

To make the system more robust we store a buffer of N previous depth-maps and a pixel is marked as consistent if it is consistent with at least $M \leq N$ of the frames in the buffer.

Generally, if a pixel is not marked as consistent (an outlier) then it is removed from the depth-map. In some cases we may use some kind of filling method so the resulting depth-map does not contain holes.

Figure 5.3 shows the result of applying this temporal consistency check to the output of the coarse-to-fine stereo method detailed in Section 5.3. We use a patch-size of 3×3, 3 pyramid levels and 3 sample points.

5.5 PatchMatch Stereo

In contrast to the fast-but-basic coarse-to-fine stereo method, PatchMatch Stereo [12] is a high quality stereo method which makes use of the PatchMatch [6] randomized correspondence algorithm. The depth/disparity map is

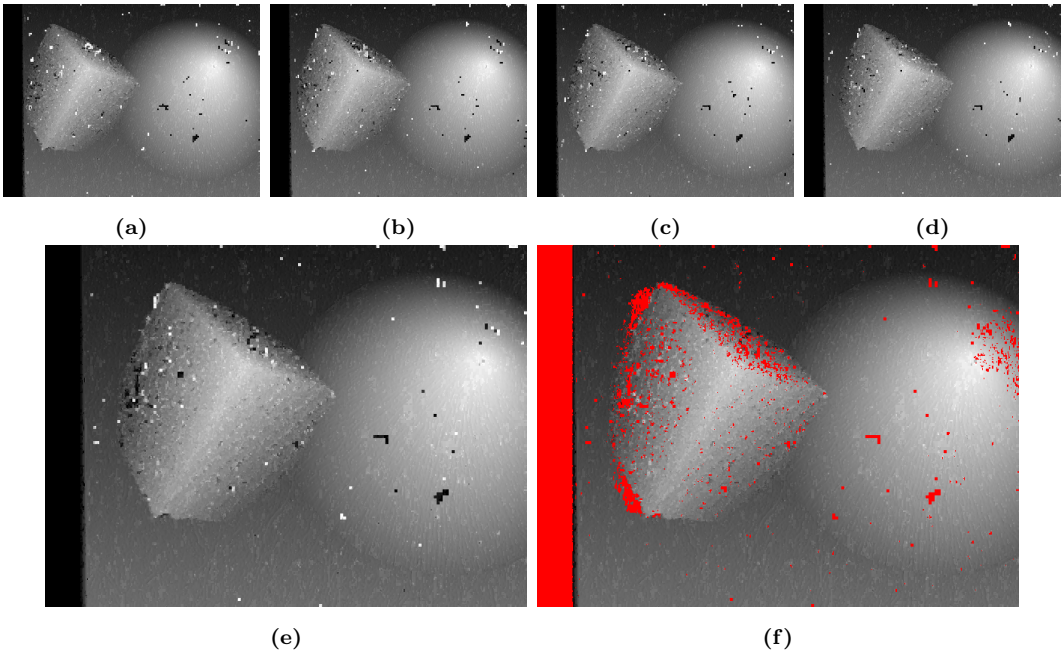


Figure 5.3: *An example of temporal consistency check. (a)-(d) are noisy depth-maps in the buffer and (e) is the current depth-map. (f) shows inconsistent pixels marked with red.*

---

over-parametrized by fitting a 3D plane per pixel. The use of slanted patches allows high-quality reconstruction of sloped surfaces. Therefore, instead of one variable per pixel, there are now three variables to solve for. The solution space is now high dimensional and would be difficult to solve via traditional optimisation methods. However, by propagating high quality random guesses to neighbouring pixels, the PatchMatch algorithm makes it efficient to solve.

5.5.1 A brief overview

Most high quality stereo methods compute the matching cost with a summation over a patch around the pixel in question. This works well because the increase in data helps to remove ambiguities and reduce noise. However, it makes the assumption that all pixels in the patch share the same depth/disparity value, which is only true for surfaces which are fronto-parallel to the camera. This assumption can cause unwanted artefacts when attempting to find the depth of slanted planes.

PatchMatch Stereo (PM-S) introduces slanted support regions — the depth at each pixel is over-parametrised by a 3D plane. While the final result is still a single depth-map, the plane parameters are applied to the patch when computing the stereo matching cost and during the propagation step.

The original PatchMatch Stereo paper formulates the problem for rectified, binocular stereo. We will present the same algorithm for general stereo correspondence.

A plane will be represented by $\mathbf{f} \in \mathbb{R}^3$. We define a per-pixel matching cost for each pixel $\mathbf{p}$:

$$m(\mathbf{p}, \mathbf{f}) = \sum_{\mathbf{q} \in N(\mathbf{p})} g(\mathbf{p}, \mathbf{q}) \rho\big(\mathbf{q}, w(\mathbf{q}, \mathbf{f})\big), \tag{5.7}$$

where $N(\mathbf{p})$ is the set of pixels in the patch centred at $\mathbf{p}$, $g(\mathbf{p}, \mathbf{q})$ is a weight function, $\rho(\cdot, \cdot)$ is the photometric error function and $w(\mathbf{q}, \mathbf{f})$ is the warp function transforming the pixel $\mathbf{q}$ to the other camera by projection onto the plane $\mathbf{f}$.

The goal is to find the optimum plane per pixel:

$$\mathbf{f}_{\mathbf{p}}^{*} = \arg\min_{\mathbf{f} \in \mathcal{F}} m(\mathbf{p}, \mathbf{f}). \tag{5.8}$$

We define the plane $\mathbf{f}$ by a normal vector $\mathbf{n}$ such that points lying on the plane satisfy the equation $\mathbf{x} \cdot \mathbf{n} + 1 = 0$. Now, the intersection of the ray from pixel $\mathbf{p}$ and the plane is:

$$\mathbf{x} = -\frac{\mathtt{K}^{-1}\dot{\mathbf{p}}}{\mathbf{n}^T \mathtt{K}^{-1}\dot{\mathbf{p}}}. \tag{5.9}$$

Using this, the warp equation in Eq. 5.7 can be written down:

$$w(\mathbf{p}, \mathbf{f}) = \pi\left(\mathtt{K}\mathtt{R}\mathbf{x} + \mathtt{K}\mathbf{t}\right) \tag{5.10}$$

$$= \pi\left(-\mathtt{K}\mathtt{R}\frac{\mathtt{K}^{-1}\dot{\mathbf{p}}}{\mathbf{n}^T \mathtt{K}^{-1}\dot{\mathbf{p}}} + \mathtt{K}\mathbf{t}\right). \tag{5.11}$$

The photometric error function is defined in [12] as:

$$\rho(\mathbf{p}, \mathbf{q}) = (1 - \alpha) \min\big(\|I(\mathbf{p}) - I'(\mathbf{q})\|, \tau_{col}\big) + \alpha \min\big(\|\nabla I(\mathbf{p}) - \nabla I'(\mathbf{q})\|, \tau_{grad}\big), \tag{5.12}$$

although any sensible cost-function could be used. The weight function:

$$g(\mathbf{p}, \mathbf{q}) = e^{-\frac{\|I(\mathbf{p}) - I(\mathbf{q})\|}{\gamma}} \tag{5.13}$$

is defined that so that pixels in the patch have a lower weight if their intensity does not match the centre pixel.

Solving Eq. 5.8 is not a trivial problem. If we only considered fronto-parallel planes then the search space is small and brute force methods such as plane sweep work well. However, with three degrees of freedom per pixel, the space to too large to solve via brute force. Therefore, we make use of the PatchMatch algorithm.

To start with, the plane parameters are randomly initialized. The space over which the planes are sampled is such that the pixel depths are within the depth-range under consideration and so that the plane normals are no more than θ_{max} from fronto-parallel.

Following initialisation we perform *spatial propagation*. We define a set of candidate planes $\mathcal{S}_{\mathbf{p}}$ per pixel consisting of the plane parameters from neighbouring pixels. We look at the matching cost for each of the candidate planes and assign the lowest cost plane to the pixel. The shape of the neighbourhood and the order in which the pixels are processed has a big effect on performance and is investigated further in Section 5.6.1. In the original PatchMatch method the pixels are processed in scanline order. On odd iterations they start in the top-left corner and finish in the bottom-right. On even iterations they start bottom-right and finish in the top-left. The neighbours which contribute to the set of candidate planes are the above and left or below and right for odd and even iterations respectively.

In the original PM-S paper the optimisation is performed on both the left and right frames of a stereo pair. Therefore they can also do *view propagation*. This is where a plane from one frame is projected into the other frame and used as another candidate plane. This step requires that PatchMatch Stereo is computed for both images, and, hence, doubles the computational cost. Therefore we do not make use of this step in real-time systems.

The next step is *plane refinement*. We randomly perturb the plane parameters per pixel and check to see if this lowers the cost. To do this we convert the plane parameters into a depth d and unit normal $\hat{\mathbf{n}}$. A new depth hypothesis is uniformly sampled from the range $[d - \Delta d, d + \Delta d]$ where Δd decreases on each iteration. A new normal hypothesis is generated by $\frac{\hat{\mathbf{n}}+\Delta\mathbf{n}}{|\hat{\mathbf{n}}+\Delta\mathbf{n}|}$ where $\Delta\mathbf{n}$ has each component randomly sampled from the range $[-\Delta n, \Delta n]$.

The algorithm then continuously loops, performing spatial propagation and plane refinement until convergence or some other stopping criteria has been met (such as time constraints).

5.5.2 Multi-view PM-S

PM-S can easily be extended to use multiple views, all projected into a single reference frame. To do this all we need to do is modify Eq. 5.7 to sum over all the images:

$$m(\mathbf{p}, \mathbf{f}) = \sum_{k} \sum_{\mathbf{q} \in N(\mathbf{p})} g(\mathbf{p}, \mathbf{q}) \rho_k\big(\mathbf{q}, w_k(\mathbf{q}, \mathbf{f})\big), \tag{5.14}$$

where ρ_k is the error between the reference image I and I_k, and w_k is the warp between the two frames.

Computing this cost function is linear in the number of images used. However, in Section 5.6.3 we explore ways to speed up this computation.

5.6 Accelerating PatchMatch Stereo

PatchMatch Stereo was originally designed as an offline stereo method. It made use of very large patch-sizes (35×35) and used a serial scanline propagation approach. However, by considering different propagation schemes we can make the system highly parallelisable and GPU friendly (Section 5.6.1). There are also some other tricks that can be done to speed up computation and convergence of the algorithm (Sections 5.6.2 and 5.6.3) so that we can use PM-S in an online setting.

All timings results in this section are run on an NVIDIA GTX 780 Ti GPU with a processing power of 5000 GFLOPS.

5.6.1 Comparison of Propagation Methods

A variety of propagation methods have been proposed for the PatchMatch algorithm. In this section we will compare the performance of each with a parallel GPU implementation. We look at both the rate of convergence per iteration and the rate of convergence in time.

The original PatchMatch formulation [6] describes a *scanline* propagation method (Fig. 5.4a), as discussed in Section 5.5.1. This seemingly serial method can be parallelised slightly by processing all pixels on a diagonal at once. The propagation method allows information to propagate arbitrarily far in a single iteration but exhibits the lowest degree of parallelism of the methods.

To fully utilize the highly parallel processing ability of modern GPUs we look for a propagation method which can process each pixel independently. The most basic way to do this is for each pixel to look at its nearest neighbours (Fig. 5.4d). This approach means that information can only propagate one pixel on each iteration but the high parallelism makes it fast to compute. We refer to this as the *parallel* method.

Barnes *et al.* [6] also proposed a parallel GPU implementation using the *jump flooding* method outlined in [107]. This approach is also fully parallel but accelerates the rate of propagation by using a varying stride when looking for neighbours (Fig. 5.4e). For some N the first iteration will have a stride of 2^N, the next iteration will have a stride of 2^{N-1} and so on until we reach a stride of 1 (which is equivalent to the parallel method). In our implementation we set $N = 3$ and repeat the loop again when the stride reaches 1.

Bailer *et al.* [5] introduce a *sweep* method which uses horizontal and vertical passes combining three pixels into one (Figs. 5.4b and 5.4c). This allows information to propagate across the whole width (or height) of the image in a single iteration. For horizontal passes each pixel in a column can be processed in parallel and for vertical passes each pixel in a row can be processed in parallel. We cycle through each of the directions as iterations increase so each direction is repeated every 4 iterations.

Each propagation method is implemented with a parallel GPU implementation. The parallel and jump flood methods are trivially parallelisable. Each pixel can be processed independently at the same time. To avoid data races we use a double buffer, one to store the old values which provide the plane hypotheses and one to store the new best result.

The other two methods have less parallelism. For the sweep method we can process each row (or column) in parallel for each vertical (or horizontal) pass. In this case the number of parallel threads is the width (or height) of the image. For the scanline method there are a maximum of $\min(width, height)$

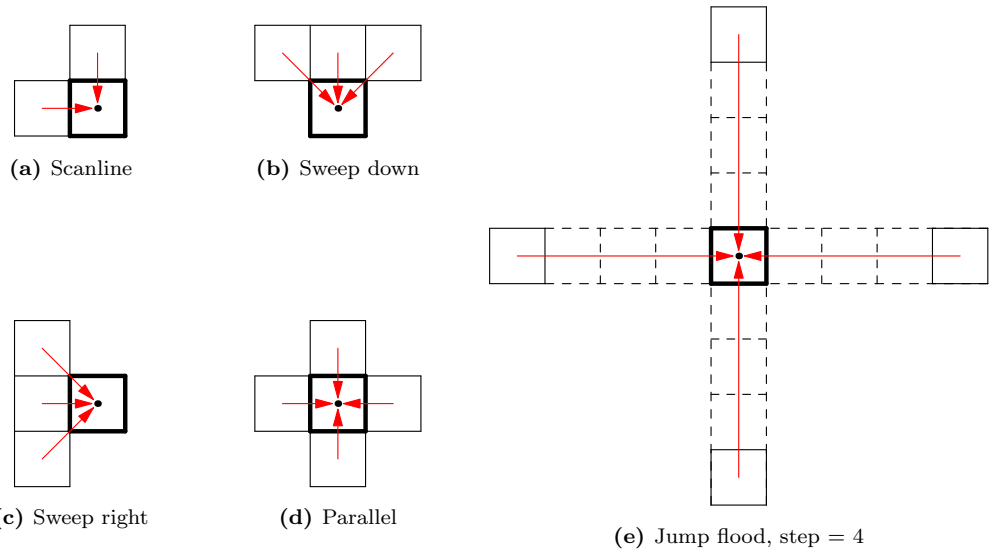


Figure 5.4: *PatchMatch propagation methods. (a), (b) and (c) exhibit a dominant propagation direction (diagonal, down and right, respectively) which allows for sequential processing and further propagation in a single iteration. (d) and (e) must be processed using an additional buffer because of cross-dependency between pixels.*

---

threads active at once, but often less since the diagonals have varying sizes. The sweep and scanline methods do not require an extra buffer but require synchronisation between each strip (horizontal, vertical or diagonal) which is processed in parallel.

In CUDA we can only synchronize threads in the same block within a kernel. On modern CUDA GPU architectures the maximum number of threads per block is 1024. Therefore, with the image sizes we are using we can use a single block of threads and synchronize within the kernel.

Appendix E provides the source-code for the propagation kernels so that the reader can see exactly how the parallelism was implemented.

Table 5.2a shows the time taken to perform a single iteration of each propagation method for a number of patch-sizes. The biggest bottleneck is computing the photometric error because of the number of texture accesses required. Even though the scanline and sweep methods do not require as many computations of the photometric error per pixel, (only 2 and 3, respectively against 4 for the parallel and jump-flood methods) their reduced level of parallelism means that

Patch size	Scanline	V-Sweep	H-Sweep	Parallel	Jump Flood
3×3	54ms	68ms	72ms	4.7ms	4.7ms
11×11	0.59s	2.2s	0.71s	45ms	50ms
19×19	2.1s	11s	2.2s	0.16s	0.18s

(a) Without cache, 2 images

Patch size	Scanline	V-Sweep	H-Sweep	Parallel	Jump Flood
3×3	28ms	28ms	37ms	2.4ms	2.4ms
11×11	0.28s	0.83s	0.33s	22ms	23ms
19×19	0.97s	4.1s	1.5s	75ms	84ms

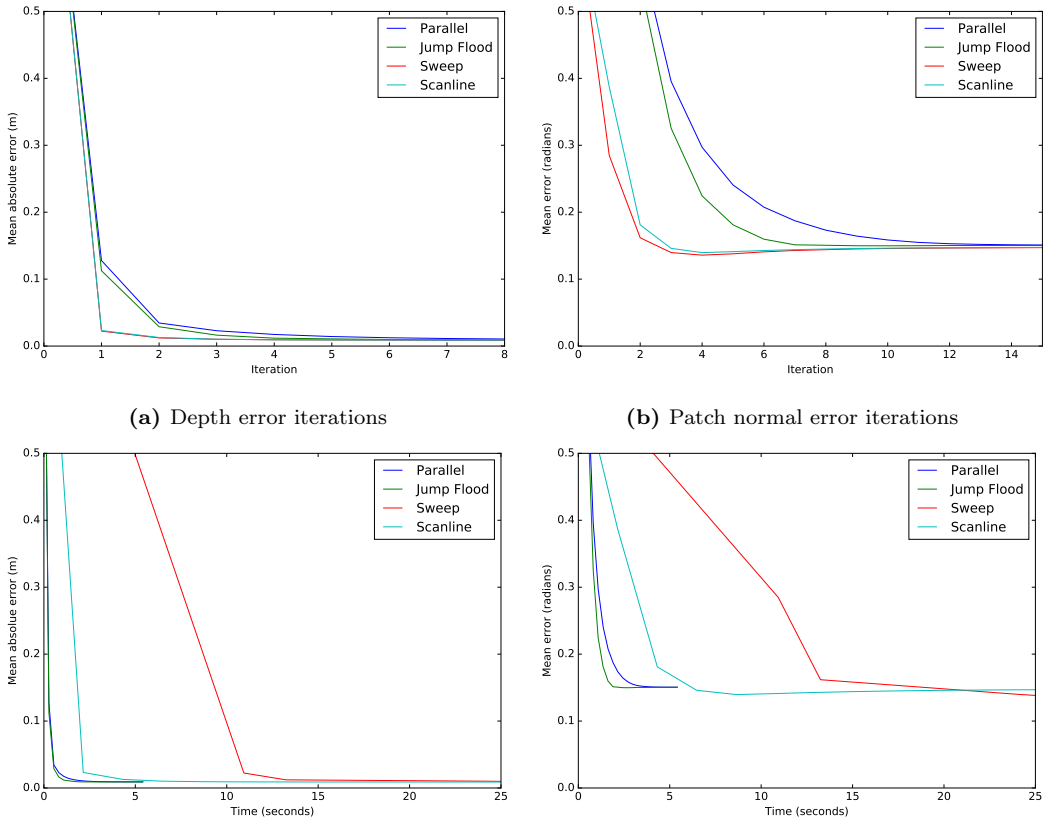
(b) With cache

Table 5.2: *Average timings for a single propagation iteration for the various methods. Note that the parallel and jump flood methods actually perform twice as much computation (in terms of photometric error calculations) as the scanline method and 33% more computation than the sweep method. Experiment performed on the dataset in Section 2.11.1.*

---

they are much slower to compute. As expected, the jump flood timings are almost the same as the parallel timings. For some reason the vertical sweep passes slow down considerably more than the horizontal sweep passes as patch-size increases. Our current hypothesis is that the ordering of the vertical passes does not suit the caching structure on the GPU resulting in a high number of cache misses. The vertical passes exhibit higher parallelism (for landscape images) and better concurrency of memory reads, which is reflected in the faster times for small patch-sizes. The hypothesis is that the order in which plane samples are processed in vertical passes leads to a much lower rate of texture cache hits when computing the photometric error.

Figure 5.5 shows the rate of convergence for each of the propagation methods against iteration count and time taken. The test data is the synthetic scene introduced in Section 2.11.1. We see that due to the ability to propagate large distances in a single iteration, the sweep and scanline methods reduce the error very rapidly per iteration. However, because of their less efficient GPU utilisation they actually are much slower than the parallel and jump flood methods. We also see one of the interesting properties of the PatchMatch algorithm in Fig. 5.5b: the error goes down and then increases slightly. Consider running the PatchMatch algorithm for infinite time — the result would be the per-pixel



(a) Depth error iterations

(b) Patch normal error iterations

(c) Depth error time

(d) Patch normal error time

Figure 5.5: *Plots showing the convergence of the PatchMatch Stereo algorithm for a variety of propagation methods. While the parallel and jumpflood methods are slower to converge per iteration (top row) we see that they can be computed much faster so that they are quicker to converge in time (bottom row). The jump-flood method performs the best overall.*

---

global minimum. Due to noise and ambiguities this is not necessarily the optimum result — this is why we often use regularisation. The propagation step of PatchMatch acts as a kind of regularisation by initially sampling planes which match neighbours. However, as time goes on the random sampling has the potential move away for the smoother solution in favour of the per-pixel global minimum, resulting in the increase in error.

5.6.2 Plane Fitting

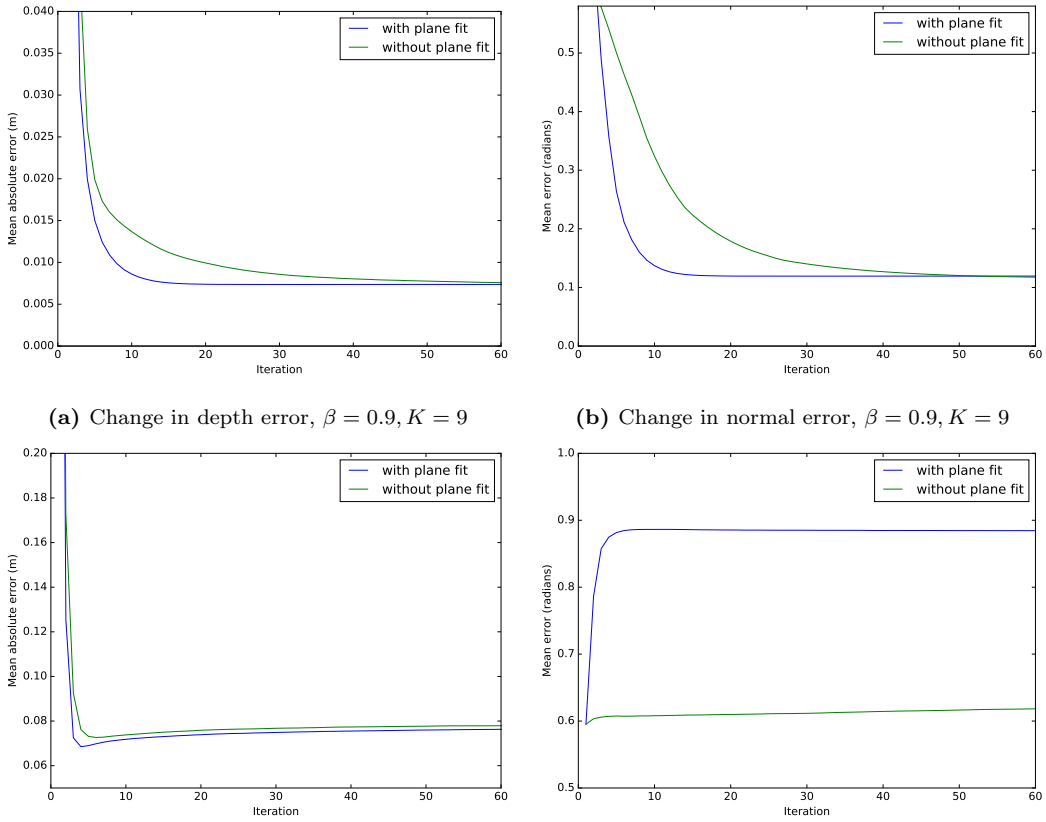
As can be seen in Fig. 5.5, the patch normals converge slower than the depth estimate. This is because normals are a second-order quantity and the fact that the photometric error is much less dependent on the normal of a patch than its depth. To accelerate the convergence of normals, a trivial extension is to use the current depth estimate as a hypothesis for the slanted plane.

To compute the new plane hypothesis the normal is obtained from the current depth estimate using the formula in Section 2.4.4, and the depth hypothesis is computed from the mean inverse depth of the four neighbour vertices. The normal is constrained to stay within the threshold specified in Section 5.5.1.

Figure 5.6 shows the convergence rate with and without plane fitting when using the parallel propagation scheme. The top row of results use a large baseline and large patches so patch normals have a significant effect. We see that plane fitting accelerates the convergence of both the depth (Fig. 5.6a) and the normals (Fig. 5.6b). In the bottom row we show results for a smaller baseline and a very small patch-size. In this situation the normals have almost no effect on the photometric cost. The results show that the plane fitting can still accelerate the convergence of the depth component (Fig. 5.6c) but estimating normals with such small patches actually has a negative effect and the plane fitting makes it worse (Fig. 5.6d). We believe this is because the photometric cost is very sensitive to a change in depth but a change in the normal has almost no effect. This means that a plane hypothesis with an accurate depth estimate but an arbitrary normal will be taken as the minimum. Due to the random nature of the optimisation and noise in the system it is now very unlikely that the depth estimate will be refined with a better normal hypothesis.

5.6.3 Cost-Volume Cache

The evaluation of the photometric cost is the most computationally expensive part of the PatchMatch stereo algorithm. When benchmarking the performance on modern NVIDIA GPU architectures it is clear that the bottleneck is the texture access. For N images with a patch-size of $K \times K$, evaluating the matching cost requires NK^2 texture lookups per pixel. When using more than 2 images, the computation time does not scale well with increasing patch-



(a) Change in depth error, $\beta = 0.9, K = 9$

(b) Change in normal error, $\beta = 0.9, K = 9$

(c) Change in depth error, $\beta = 0.4, K = 1$

(d) Change in normal error, $\beta = 0.4, K = 1$

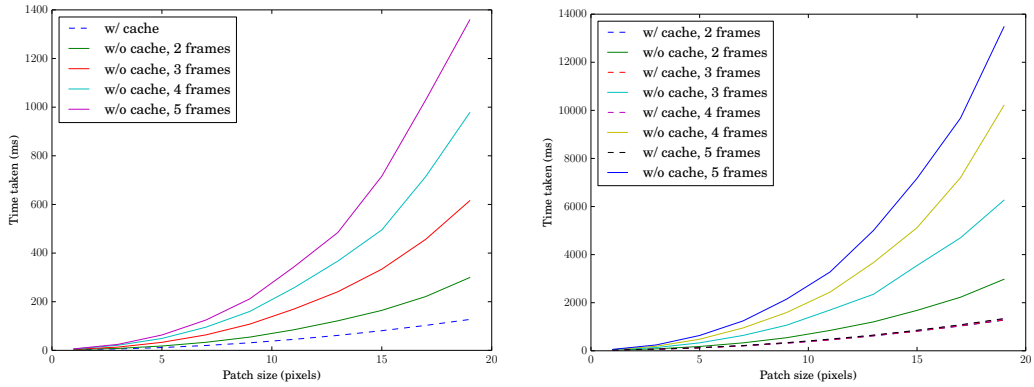
Figure 5.6: *Plots showing the change in error with increasing iterations with and without using plane fitting.*

---

size. One method we introduce to reduce this overhead is to construct a cost-volume (Section 5.2) and use it as a cache. For a cost-volume with B bins this requires NB texture lookups per pixel to create and then K^2 texture lookups to compute the photometric cost. For large patches and more images the gains can be considerable.

Since PatchMatch Stereo is an iterative process, the matching cost is evaluated multiple times. Let M be the number of evaluations. Then the total number of texture lookups when computing the matching cost per pixel comes to MNK^2 for the original method and $NB + MK^2$ when using the cache. Hence, for large M we have reduced the number of lookups by a factor of N (note that $N \geq 2$).

The depth in the cost-volume is sampled at discrete depths and then we



(a) **(b)**

Figure 5.7: *Using a cost-volume as a cache when computing photometric error can massively increase performance, especially when more than 2 images are used. On the left (a) we show the time taken for one iteration of propagation and refinement. On the right (b) we show the time taken to initialise the system (i.e. build the cost volume) followed by 10 iterations of propagation and refinement. We see that the initialisation time becomes negligible compared to the time spent on iterations and the cache massively increases performance. The cache here has 64 bins.*

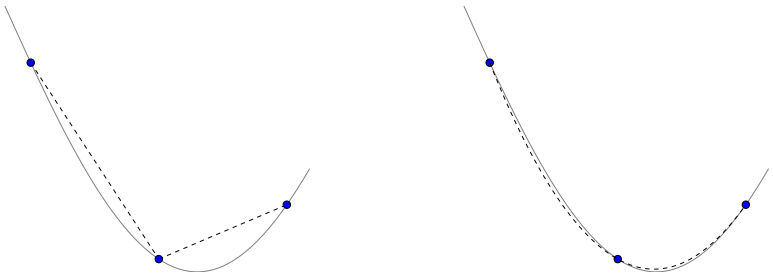
---

sample the cost at continuous depths by using linear interpolation in the depth component. The use of a cost-volume as a cache creates a restriction on the form of the patch-based photometric error. It must take the form of a sum over individual pixel errors and the sums must be separable:

$$m(\mathbf{p}, \mathbf{f}) = \sum_{\mathbf{q} \in N(\mathbf{p})} g(\mathbf{p}, \mathbf{q}) \sum_{k} \rho_k\big(\mathbf{q}, w_k(\mathbf{q}, \mathbf{f})\big). \tag{5.15}$$

The cost-volume stores the sum of the pixel errors over the images (indexed by k) and then computing the full matching cost consists of just summing over the neighbourhood $N(\mathbf{p})$.

Figure 5.7 shows how this caching method can increase the performance of PatchMatch Stereo. We use a cache with 64 depth bins. In Fig. 5.7a we see the times increase quadratically with the patch-size, as expected due to the number of texture lookups. However, that graph does not account for the added computation time to build the cost-volume. Therefore, we include the initialisation time in Fig. 5.7b. We note that with a high number of iterations the initialisation time becomes negligible compared to the time spent on the



(a) Linear approximation **(b)** Quadratic approximation

Figure 5.8: *If we use a linear approximation of the cost function (a) then the minimum cost will always lie at one of the discrete sample points. By using a quadratic approximation (b) we get a much better approximation of the real function allowing the minimum to lie at non-discrete locations.*

---

iterations. Table 5.2b shows timings for a single propagation iteration for all of the propagation methods outlined in Section 5.6.1.

If we use linear interpolation to approximate the cost function within the cache, the global minimum per pixel will always lie at one of our discrete sample points (see Fig. 5.8a). Even when we sum together these discrete minima in the patch we still end up with significant quantisation artefacts and a tendency to favour fronto-parallel plane solutions (see Fig. 5.9b).

We introduce the quadratic version of the cache (Fig. 5.8b). This requires three texture lookups in the cache instead of two but does increase the accuracy of the results. To find the cost of pixel $\mathbf{u}$ at inverse depth ξ we find the closest discrete inverse depth sample ξ_i, for $i \in 1, ..., B-2$, where the cache has B inverse depth bins. We then fit a quadratic to this point and its two neighbouring inverse depth samples:

$$C_{\mathbf{u}}(\xi) = C_{\mathbf{u}}(\xi_i) + C'_{\mathbf{u}}(\xi_i)\,(\xi - \xi_i) + \frac{C''_{\mathbf{u}}(\xi_i)}{2}\,(\xi - \xi_i)^2\,, \tag{5.16}$$

where $C_{\mathbf{u}}(\xi_i)$ is the cost for pixel $\mathbf{u}$ and inverse depth ξ_i sampled directly from the cost-volume cache. The derivatives are computed using the finite difference operators:

$$C'_{\mathbf{u}}(\xi_i) = \frac{C_{\mathbf{u}}(\xi_{i+1}) - C_{\mathbf{u}}(\xi_{i-1})}{2\Delta\xi} \tag{5.17}$$

$$C''_{\mathbf{u}}(\xi_i) = \frac{C_{\mathbf{u}}(\xi_{i+1}) - 2C_{\mathbf{u}}(\xi_i) + C_{\mathbf{u}}(\xi_{i-1})}{2(\Delta\xi)^2}, \tag{5.18}$$

Bins	Init.	Iterate	
		Lin.	Quad.
0	0.40ms	84ms	84ms
16	1.0ms	37ms	39ms
32	1.9ms	38ms	40ms
48	2.9ms	39ms	42ms
64	3.8ms	40ms	45ms
128	7.4ms	45ms	57ms
256	15ms	57ms	74ms

(a) Timings for initialisation of the cache and to perform an iteration of propagation and refinement.

Bins	Depth Error		Normal Error	
	Lin.	Quad.	Lin.	Quad.
0	0.013	0.013	0.24	0.24
16	0.046	0.027	0.52	0.57
32	0.023	0.015	0.41	0.31
48	0.018	0.013	0.32	0.26
64	0.015	0.013	0.28	0.25
128	0.013	0.013	0.24	0.24
256	0.012	0.013	0.24	0.24

(b) Depth and normal errors for both caching methods.

Images	2	3	4	5	6	7	8	9	10
Init. time (ms)	3.8	5.0	6.3	7.6	9.0	10.3	11.7	13.1	14.5

(c) The initialisation time to create the cost-volume cache is linear in the number of images used. Once created, the iteration times are constant for any number of images. Here we show initialisation times for a cache with 64 bins.

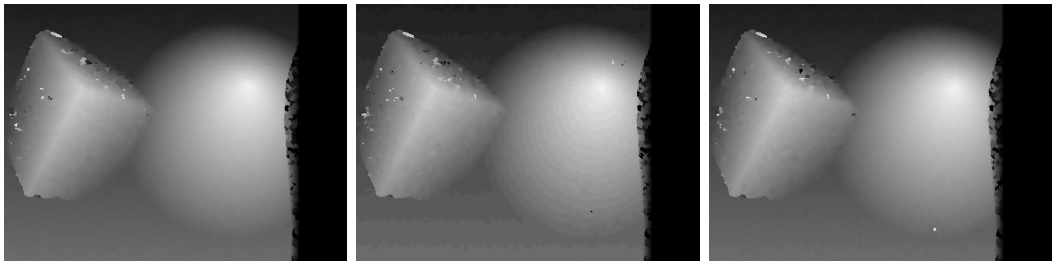
Table 5.3: *Tables demonstrating the timings and errors when using a cost-volume cache with varying number of depth bins (0 bins corresponds to no cache). The tests were run on the dataset in Section 2.11.1 with a patch-size of* 11×11 *and baseline of* 0.9*. We see that for a small increase in computation time we can get significant increases in accuracy when choosing quadratic over linear.*

---

where $\Delta\xi$ is the size of the discretisation in inverse depth $\Delta\xi = \frac{1}{B-1}$. Note that in this case the finite difference operators provide the *exact* solution for the quadratic passing through the three points, not just an approximation.

The result of using the quadratic cache is that we get higher accuracy with fewer bins in the cache. This can be seen in Table 5.3 where we show how the processing time and the error in the result change with varying number of bins for both linear and quadratic cache approaches. To get a good accuracy for the linear approach we need a large number of samples but the quadratic approach allows us to use a significantly smaller cache with only a very small increase in processing time.

Figure 5.9 shows how the use of a small number of bins causes significant discretisation artefacts when using a linear cache. The artefacts disappear



(a) No cache **(b)** Linear cache **(c)** Quadratic cache

Figure 5.9: *A comparison of the depth result when using the caching mechanisms introduced in this section. The linear approach causes quantisation artefacts which are not present when using the quadratic approach.*

---

when using the quadratic approach. Note that in this example the length of the epipolar line over the depth range is 43 pixels. Therefore, there is very minimal increase in accuracy once the number of bins exceeds this value.

One could compare this method to the approach of [106]; in this work a cost-volume is created and then filters are applied to each slice. The use of filtering is equivalent to aggregating the patch-based cost for fronto-parallel planes at each discrete depth sample. They use fast filtering techniques to be computationally efficient. Our approach is similar in terms of aggregation from a cost-volume but our method allows the use of slanted patches. The compromise is that we lose the ability to use the fast filtering techniques used in [106].

5.7 When do slanted windows make sense?

One of the reasons PatchMatch Stereo produced such high quality results on the Middlebury datasets is the use of slanted support regions. This enabled much better reconstruction of slanted surfaces. However, the question may be asked "do slanted patches always provide an improvement?" In this section we will try to answer this question.

We will start by doing some simple analysis in the case of rectified stereo to see the effect of slanted planes on the homography. We will quantify the change in the homography as the patches become slanted as a function of the baseline, depth and patch-size. From this we will draw some conclusions about

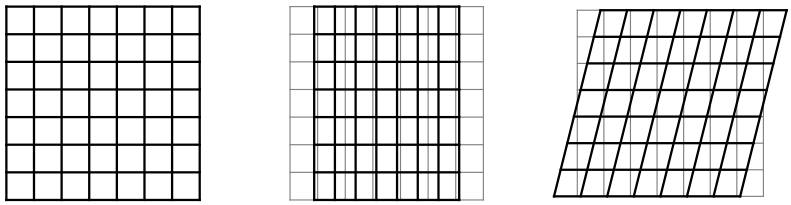


Figure 5.10: *The shape of the re-projected patch changes as the normal changes. On the left, a fronto-parallel patch re-projects to square. In the middle, the normal points in the x-direction resulting in a scaling in that dimension. On the right, the normal points in the y-direction resulting in a shearing transformation. In general, the shape of the reprojected patch will be related to the square by an affine transformation.*

---

when the slanted planes have a noticeable effect. We will then follow this up with some experiments on synthetic data to see the effect of the slanted planes on the depth-map error for a variety of baselines and patch-sizes.

Let us consider a rectified stereo pair of cameras $\mathcal{C}$ and $\mathcal{C}'$. In this case a square, fronto-parallel patch in camera $\mathcal{C}$ re-projects to a square patch in camera $\mathcal{C}'$. The projection between image coordinates (Section 2.4.1) is simplified:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} u + \frac{f_x \beta}{D(\mathbf{u})} \\ v \end{bmatrix}. \tag{5.19}$$

To quantify the effect of slanted patches we will look at the position of the re-projection of all pixels in a patch. We can see how the position changes when the normal of the patch is changed.

For a fronto-parallel patch, a pixel $\mathbf{u} + \delta\mathbf{u}$ in the patch centred at $\mathbf{u}$ has the same depth as $\mathbf{u}$ and projects to:

$$\begin{bmatrix} u + \delta u + \frac{f_x \beta}{D(\mathbf{u})} \\ v + \delta v \end{bmatrix}. \tag{5.20}$$

By using the plane equation $\mathbf{x} \cdot \hat{\mathbf{n}} + d = 0$ and inverse projection (Eq. 2.6) we

can show that a planar surface leads to a constant gradient of inverse depth:

$$\mathbf{x} \cdot \hat{\mathbf{n}} + d = 0, \quad \mathbf{x} = D(\mathbf{u})\mathtt{K}^{-1}\dot{\mathbf{u}}, \tag{5.21}$$

$$\implies \frac{1}{D(\mathbf{u})} = -\frac{\hat{\mathbf{n}}^T\mathtt{K}^{-1}\dot{\mathbf{u}}}{d}, \tag{5.22}$$

$$\implies \frac{\partial}{\partial \mathbf{u}}\frac{1}{D(\mathbf{u})} = -\frac{1}{d}\begin{bmatrix}\frac{\hat{n}_x}{f_x} \\ \frac{\hat{n}_y}{f_y}\end{bmatrix}. \tag{5.23}$$

Let $\mathbf{g}(\mathbf{u}) = (g_x(\mathbf{u}), g_y(\mathbf{u}))$ be the gradient of the inverse depth for the slanted patch at $\mathbf{u}$, as defined by the plane parameters and focal length in Eq. 5.23. Now, a pixel $\mathbf{u}+\delta\mathbf{u}$ in a slanted patch will have depth dependent on its position relative to the patch centre:

$$\frac{1}{D(\mathbf{u}+\delta\mathbf{u})} = \frac{1}{D(\mathbf{u})} + \delta\mathbf{u} \cdot \mathbf{g}(\mathbf{u}), \tag{5.24}$$

and, hence, the pixel will re-project to:

$$\begin{bmatrix} u + \delta u + \frac{f\beta}{D(\mathbf{u}+\delta\mathbf{u})} \\ v + \delta v \end{bmatrix} = \begin{bmatrix} u + \delta u + f\beta\left(\frac{1}{D(\mathbf{u})} + g_x(\mathbf{u})\delta u + g_y(\mathbf{u})\delta v\right) \\ v + \delta v \end{bmatrix}. \tag{5.25}$$

Therefore, the change in re-projection (change in disparity) of a pixel in a slanted patch compared to a fronto-parallel patch is given by the difference between Eq. 5.25 and Eq. 5.20:

$$\delta x = f\beta\big(g_x(\mathbf{u})\delta u + g_y(\mathbf{u})\delta v\big) = -\frac{\beta}{d}\left(\hat{n}_x\delta u + \hat{n}_y\delta v\right), \tag{5.26}$$

where we have made use of Eq. 5.23. To infer useful information from this equation, we need to have it in terms of measurable parameters. $\beta, \hat{\mathbf{n}}$,and $\delta\mathbf{u}$ are measurable but the value of the plane parameter d is unclear. We therefore make use of Eq. 5.21 to rewrite d in terms of useful quantities:

$$d = -\mathbf{x} \cdot \hat{\mathbf{n}} \tag{5.27a}$$

$$= -D(\mathbf{u})\left(\frac{\hat{n}_x}{f_x}(u - u_0) + \frac{\hat{n}_y}{f_y}(v - v_0) + \hat{n}_z\right). \tag{5.27b}$$

To simplify this equation we will consider the centre pixel of the image, $\mathbf{u} = \mathbf{u}_0$. In this case Eq. 5.27a simplifies so that $d = -D(\mathbf{u})\hat{n}_z$. We write the plane normal in spherical polar coordinates:

$$\hat{n}_x = \sin\theta\cos\phi, \tag{5.28a}$$

$$\hat{n}_y = \sin\theta\sin\phi, \tag{5.28b}$$

$$\hat{n}_z = \cos\theta. \tag{5.28c}$$

We now have all the tools we need to write down the change in re-projection for a pixel in the patch centred at $\mathbf{u} = \mathbf{u}_0$:

$$|\delta x| = \left| \frac{\beta}{D(\mathbf{u}_0)\hat{n}_z} \big(\hat{n}_x \delta u + \hat{n}_y \delta v\big) \right| \tag{5.29a}$$

$$= \left| \frac{\beta}{D(\mathbf{u}_0)} \tan\theta \big(\cos\phi \delta u + \sin\phi \delta v\big) \right|. \tag{5.29b}$$

We can now deduce that the change in position from Eq. 5.26 is linear in the baseline, inverse depth and the distance of the pixel from the patch center. It would be useful to have an upper bound for the magnitude $|\delta x|$. To this end, consider the term in parentheses in Eq. 5.29b. This can be thought of as the dot product of a unit vector $\hat{\mathbf{v}}_\phi = (\cos\phi, \sin\phi)^T$ and the displacement vector $\delta\mathbf{u}$. The magnitude of this dot product will have a maximum when the two vectors are parallel such that $\hat{\mathbf{v}}_\phi = \pm\frac{\delta\mathbf{u}}{|\delta\mathbf{u}|}$ and the maximum is given by:

$$|\hat{\mathbf{v}}_\phi \cdot \delta\mathbf{u}| \leq |\delta\mathbf{u}| = \sqrt{\delta u^2 + \delta v^2}. \tag{5.30}$$

Over the the square domain of a patch $|\delta\mathbf{u}|$ has a maximum when $\mathbf{u} = (K, K)^T$, where the patch is of size $(2K+1) \times (2K+1)$. Hence, $|\delta\mathbf{u}| \leq \sqrt{2}K$ and we now have an inequality constraint on $|\delta x|$:

$$|\delta x| \leq \left| \frac{\beta\sqrt{2}K}{D(\mathbf{u})} \tan\theta \right|. \tag{5.31}$$

Using Eq. 5.31 we can now do a quantitative analysis of the effect of using slanted patches for a given camera configuration. Is there a configuration where the use of slanted patches will make no difference? Suppose we want the change in projection to be less than 1 pixel for all planes up to $\frac{\pi}{3}$ $(60°)$ from fronto-parallel, with a patch-size of 5×5 $(K = 2)$:

$$|\delta x| \leq \left| \frac{\beta\sqrt{2}K}{D(\mathbf{u})} \tan\theta \right| \leq 1 \tag{5.32}$$

$$\implies \frac{\beta}{D} \leq \frac{1}{2\sqrt{6}} = 0.20. \tag{5.33}$$

So for this to hold the baseline must be less than 0.20 times the minimum depth.

We now take a look at the popular Middlebury stereo dataset. The dataset from 2005/06 comes with calibration information. The focal length is 3740 pixels, the baseline is 160mm. The data is in the form of disparity maps stored

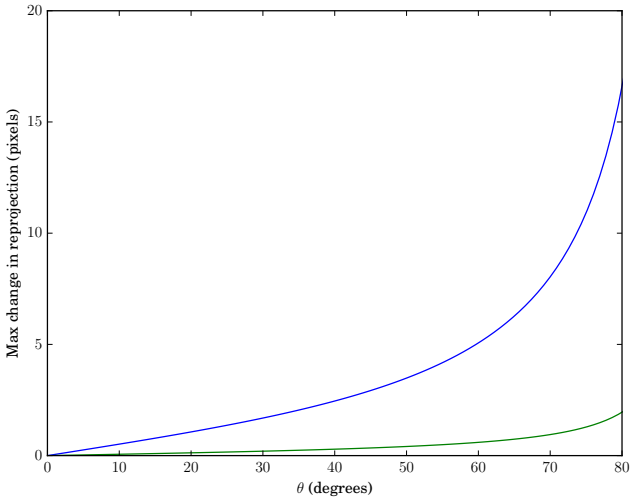


Figure 5.11: *The maximum change in re-projection when using slanted patches vs fronto-parallel patches for the Middlebury dataset. The blue line is for a patch-size of* 35×35*, as used in the PatchMatch Stereo paper. The red line is for a patch-size of* 5×5*, a typical size for real-time applications.*

---

as PNG images along with a file containing a value d_{min} such that the value in the disparity map $\in [0, 255]$ plus d_{min} provides the disparity to map to 3D coordinates (this is due to image cropping).

We can therefore say that

$$\frac{f\beta}{D} \leq 255 + d_{min}, \tag{5.34}$$

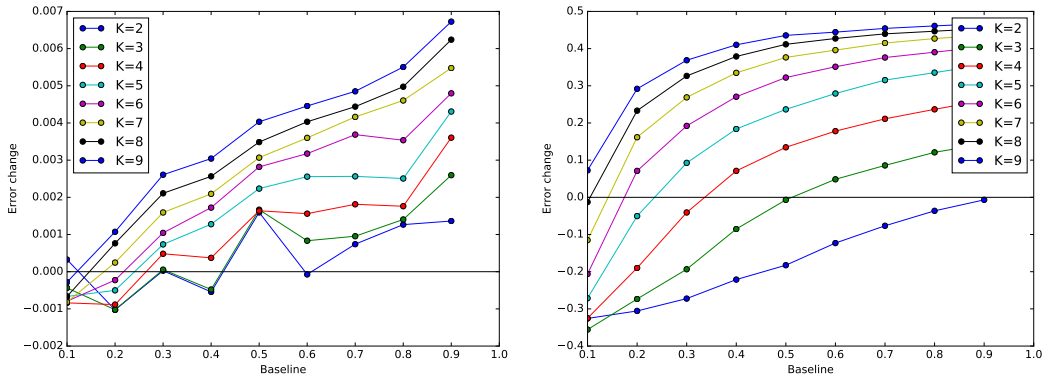
and, hence,

$$|\delta x| \leq \left| \frac{\beta\sqrt{2}K}{D} \tan\theta \right| \tag{5.35}$$

$$\leq \left| \sqrt{2}K \frac{255 + d_{min}}{3740} \tan\theta \right|. \tag{5.36}$$

Again, substituting $\theta = \frac{\pi}{3}$ and using $d_{min} = 200$ (from the Art dataset) we see that the pixel error $\delta x \leq 0.30K$. The original PatchMatch stereo paper used a patch-size of 35×35 so that $K = 17$. This means the maximum displacement is 5.1 pixels.

Figure 5.11 shows a plot of the function for a patch-size of 35×35 as used in the PatchMatch Stereo paper and a patch-size of 5×5, which is a typical size



(a) Change in depth error

(b) Change in normal error

Figure 5.12: *Plots showing the change in error between using fronto-parallel patches and slanted patches in PatchMatch Stereo. A positive value means that slanted patches outperform the fronto-parallel patches. We see that using slanted patches with a small baseline and patch size has a negative effect on the stereo result.*

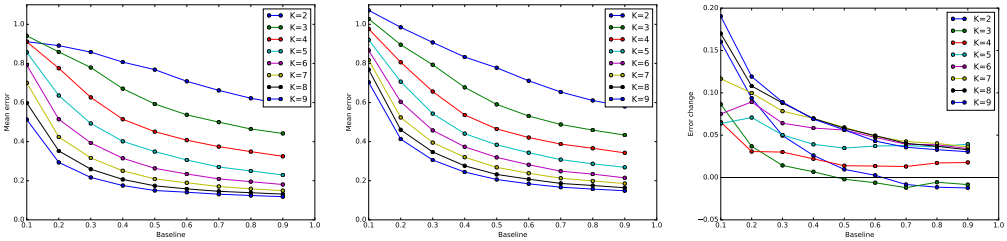
---

used in real-time applications. The change in re-projection for a 5×5 patch does not exceed one pixel until the angle exceeds 70 degrees for fronto-parallel.

The tangent function starts to increase rapidly, and at 80 degrees tangent it is 3.3 times larger than at 60 degrees. Therefore, it seems that slanted planes are most useful for planes which are very far from fronto-parallel but not so useful for the majority of cases.

Now we look at some experimental results. Using a synthetic rendered stereo pair we apply the PatchMatch Stereo algorithm with and without the use of slanted planes. In Fig. 5.12 we plot the change in error when using slanted patches versus fronto-parallel patches. The results show that for small patch sizes and baselines the use of slanted patches can actually make the depth-map worse. We believe this is due to the increase in the solution space which results in more local minima and less effective propagation – a correct depth hypothesis with a bad normal hypothesis can have a low cost but does not propagate its depth to its neighbours well, although, our plane fit sampling helps here.

In Fig. 5.13 we look at how accurately the normals of the patches reflect the true normals of the surface and compare this to the normals computed directly from the depth-map. As expected, the accuracy of the estimated normals (for



(a) Normals from patches **(b)** Normals from depth-map **(c)** Difference

Figure 5.13: *Plots showing the mean normal error for a test sequence with varying baseline and patch-size. On the left we have the normals recovered from the patches. In the middle we show the normals computed from the depth-map. In both cases the normal error reduces with larger patches and larger baselines. On the right we show the depth-map normal error minus the patch normal error. A positive value means that the patches give a better estimate of the normal than the depth-map.*

---

both the patches and from the depth-map) increases with increasing patch size and baseline. In general, the normals estimated from the patches are better than those estimated from the depth-map and the bigger patch size, the bigger the difference.

In our implementation of PatchMatch Stereo we see that texture access is the bottleneck of the algorithm. Therefore we did not see much change in processing time when using slanted patches vs. fronto-parallel patches.

5.8 PatchMatch Stereo with Regularization

One of the drawbacks of PatchMatch Stereo is the lack of regularisation or smoothing. In the original paper they use a patch-size of 35×35 which means there is less ambiguity between patches and hence outliers are not very common. This is in opposition to a real-time scenario where we are limited to small patch sizes resulting in a much noisier depth-map. In addition, PM-S compute depth-maps for both the left and right images in the stereo pair and so uses consistency checks between the two to remove any outliers.

Computing depth-maps for both images images immediately doubles the computation needed, something infeasible in an interactive setting. However,

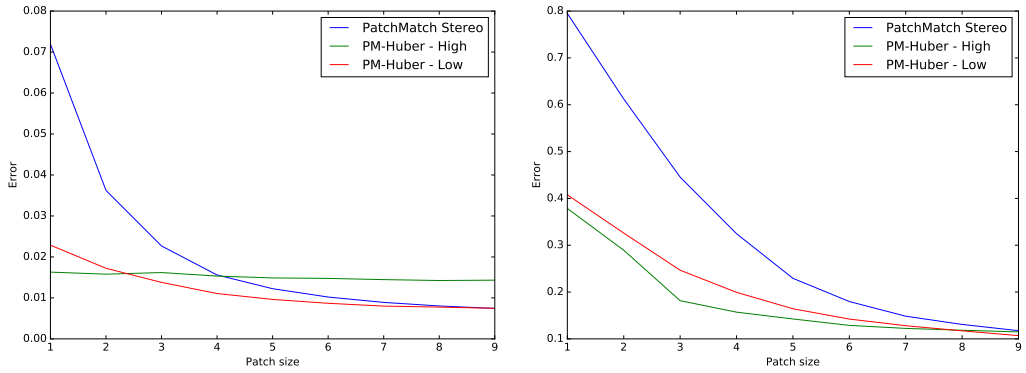
due to the interactive setting we can use temporal consistency to help remove outliers on the current frame, as outlined in Section 5.4.

Various approaches have been made to add pairwise regularisation terms to PatchMatch Stereo. The most notable of these are PM-Huber [57], which uses variational denoising methods to smooth the result, and PMBP [9], which uses belief propagation to solve an energy which includes a pairwise term. Out of the two methods, PM-Huber shows a lot more potential for interactive applications due to the use of primal-dual optimisation schemes which have already been demonstrated as real-time capable. Therefore, we will focus our analysis on this approach.

The results demonstrated by PM-Huber were again on the Middlebury Stereo Datasets. Due to the clear focus on accuracy the patch-size used was 41×41 pixels. This is not a usable patch-size in real-time stereo so we have no intuition for how this algorithm will perform when small patches are used which will lead to more outlier noise. The effect of PM-Huber versus PatchMatch Stereo is expected to be much more noticeable when there is significant noise present, as in short baseline stereo with small patches, which is common in real-time methods. The full PM-Huber approach consists of a joint optimisation which alternates between PatchMatch Stereo and Huber regularisation. We also compare against applying Huber regularisation to the final output of PatchMatch stereo, denoted "PM + Huber".

Figure 5.14 shows the effect on the error when using PM-Huber versus PatchMatch Stereo for varying patch-sizes. We see that the reduction in error is much greater for small patches which reinforces the view that PM-Huber could be very effective when working within the constraints of real-time systems.

Table 5.4 compares the inverse depth error for PatchMatch Stereo, PM + Huber and PM-Huber. PM-Huber gives the lowest error and the largest error is without any regularisation. We note that the reduction in error from using the full PM-Huber joint optimisation is smaller than the improvement from just applying Huber regularisation to the PatchMatch Stereo result. Therefore, PM + Huber is a good balance between accuracy and computational cost.



(a) Depth error

(b) Patch normal error

Figure 5.14: *A comparison of PatchMatch Stereo versus PM-Huber. 'High' and 'Low' refer to the amount of smoothing applied using PM-Huber. With a high level of smoothing PM-Huber is very effective at reducing the error from small patches but over-smooths and degrades the result when using larger patches. The effect of PM-Huber in reducing the error of PatchMatch Stereo appears to be reduced as patch-size increases.*

5.9 Conclusions

In this chapter we have presented a number of ways to solve the depth-from-stereo problem, targeted at a real-time system. Taking inspiration from Newcombe *et al.*'s DTAM [88] we aimed to make a stereo system capable of making high-quality depth-maps in real-time. However, while DTAM uses several hundred frames to create each individual depth-map, we aim for a method using only a few frames. In the case of a moving camera this leads to a temporally dense stream of depth-maps which we then fuse into a 3D model in Chapter 6.

While investigating different stereo methods we first gave an overview of a fast, basic coarse-to-fine method which can easily run at real-time rates. In the quest for higher quality depth maps we took a close look at the PatchMatch Stereo algorithm and proposed a number of optimisations and modifications which allowed a once offline method to run at close to real-time speeds. Both these approaches to stereo will be used in the next chapter as part of a full 3D reconstruction system.

	PatchMatch		PM + Huber		PM-Huber	
	all	no outliers	all	no outliers	all	no outliers
Median	0.016	0.011	0.010	0.0068	0.0076	0.0054
Mean	0.068	0.018	0.051	0.0094	0.044	0.0075
Std. Dev.	0.12	0.019	0.11	0.0087	0.10	0.0067

Table 5.4: *The inverse depth error for a variety of PatchMatch Stereo methods. The first column is just PatchMatch, the second is PatchMatch followed by Huber regularisation and the third is the joint optimisation approach of PM-Huber. The "no outliers" statistic is computed by removing any errors which are greater than 1.5 times the upper quartile error.*

6

Real-time Monocular Reconstruction using Depth-map Fusion

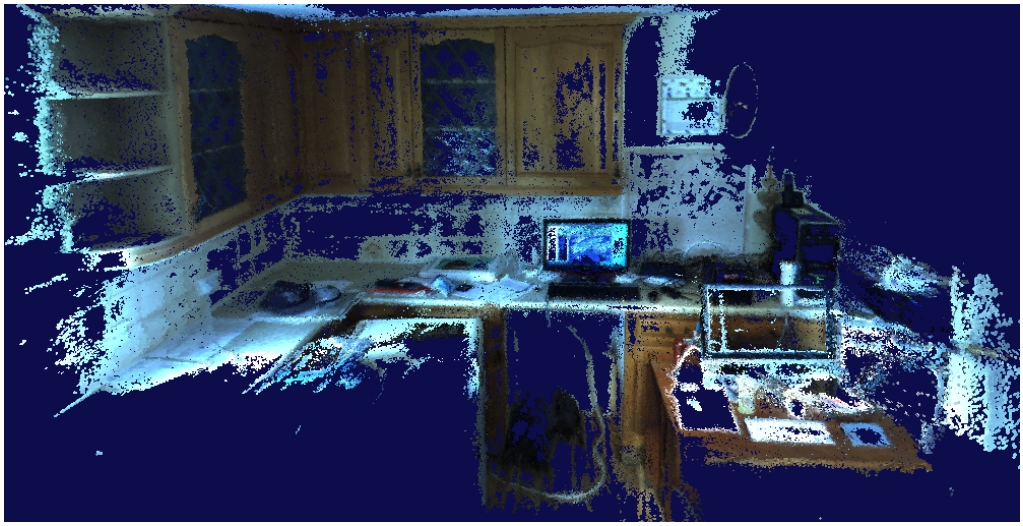


Figure 6.1: *An example reconstruction of a scene using the system described in this chapter.*

The work in Chapter 3 showed that the surface light-field is a convenient representation for inferring illumination and reflectance information. However, that work was initially limited to planar surfaces and we wish to extend it to work with general 3D surfaces. Therefore, in this chapter we will present a novel system to generate dense 3D models from a hand-held monocular camera in real-time. Such a system we hope can be a foundation on which to capture general 3D surface light-fields. A reconstruction is created by combining the multi-view stereo methods of Chapter 5 with a surfel-based fusion approach. The system can theoretically run with any tracking method: we initially use ORB-SLAM as a robust feature-based tracker and also demonstrate how to perform dense, direct tracking of the camera to the surfel model. We then compare the results obtained from both approaches..

6.1 Background

Real-time, dense reconstruction has only recently (within the last 5 years) become feasible due to the increased performance and programmability of consumer GPUs. In 2010, Newcombe and Davison [86] demonstrated live, dense reconstruction with a moving camera. They generate a coarse base mesh using feature points and then deform the base mesh into highly accurate depth-maps using variational optical flow on a GPU. In 2011, KinectFusion [87] made a huge impact demonstrating 30fps dense tracking and depth-map fusion of a RGB-D Microsoft Kinect sensor. The system uses the Truncated Signed-Distance Function (TSDF) to fuse depth-maps on every frame into a consistent, smooth 3D model. Also in 2011, Newcombe *et al.* [88] presented Dense Tracking and Mapping (DTAM) which, in real-time, fuses hundreds of frames into a cost-volume and uses variational methods to extract smooth depth-maps for sparse key-frames. At the same time, Graber *et al.* [50] used plane-sweep stereo to generate depth-maps for key-frames and use variational methods to fuse the depth-maps in a TSDF framework.

Further extensions of the KinectFusion method have focused on making it more scalable. Kintinuous [129] uses a sliding volume approach to make very large scale maps and adds the ability to perform large-scale loop closures. Niessner *et al.* [90] use efficient voxel hashing to avoid storing the complete TSDF leading to huge memory savings. Zhou and Koltun [138] add full pose graph optimisation as a post-processing step and focus on high quality reconstruction of *points of interest*. Poses on the paths in between the points of interest are considered more flexible in the optimisation.

As an alternative map representation, Keller *et al.* [65] make use of a surfel map to represent the world instead of the memory hungry TSDF volume. Their method allows better reconstruction of thin structures and also extends to handle basic dynamic scenes. Salas-Moreno *et al.* [110] detect planar surfaces within a surfel-based reconstruction. By combining surfels belonging to the same plane a more compact representation if achieved. The extracted planes are then used as surfaces for augmented reality. ElasticFusion [128] also uses a surfel map but adds support for small scale and large scale loop closure. This makes it especially suited to the loopy trajectories often experienced when making scans of rooms.

The interesting thing that most of these methods have in common is that, as input they take a depth-map from an RGB-D sensor. However, there is no constraint that this must be the case. If depth-maps can be provided from another source, such as multi-view stereo from a moving monocular camera, then all of these methods should be directly usable, hence the inspiration for this chapter.

Newcombe demonstrates this in his thesis [85]; that it is possible to generate depth-maps from a monocular camera at near frame-rate ($\sim$ 20fps) and then fuse these depth-maps using the TSDF. The 3D model can then be textured and used for direct tracking of the camera. The system is bootstrapped with a feature-based tracker and once there are enough tracked frames to generate a reliable 3D model, the feature tracker can be deactivated to create a fully-dense tracking and mapping system.

REMODE [99] is another dense, monocular 3D reconstruction approach. They generate dense depth maps within a probabilistic framework. Each pixels depth is represented using a Gaussian plus outlier model. The parameters of the model are then used to weight each pixels contribution in a variational regularisation framework. The result is a set of keyframe depth-maps but they don't appear to fuse them together. MonoFusion [101] uses a greatly simplified version of the PatchMatch Stereo algorithm to generate depth-maps and fuses them together using a TSDF volume. However, the minimal number of results and no quantitative evaluation casts doubt over the robustness of the system.

We goal of this chapter is to create a monocular 3D reconstruction framework on which we can expand the surface light-field work of Chapter 3. Therefore, the map needs to be represented in world space rather than a keyframe-based representation. The two potential representations that stand out from the prior works above are the TSDF volume and the surfel model. Both of these have previously been used for RGB-D reconstruction and the TSDF volume has been demonstrated to work within a monocular system [85]. However, the TSDF representation is not well suited to surface light-field capture. Within a surfel based system the surface is already discretised into elements which could easily be extended to contain lumispheres. The TSDF's implicit volumetric representation does not have a simple extension to include surface light-field data. Additionally, the resolution of the TSDF volume is fixed and independent of the resolution of the camera whereas a surfel based model adapts to

the resolution and scale of the input images and can even handle multiple different scales in one model. Due to these differences we see a surfel-based fusion approach to be the superior method when the end goal is to capture surface light-fields.

6.1.1 Tracking

None of these dense-reconstruction methods would be possible without the massive advancements in camera tracking made over the last decade. Davison's MonoSLAM [32] was one of the first real-time camera tracking algorithms which made use of sparse features in a probabilistic framework, notable by the uncertainty ellipses surrounding each feature in the map. On of the next big advancements is with Klein and Murrays Parallel Tracking And Mapping (PTAM) [68] which split tracking and mapping into two separate CPU threads allowing the use of costly but accurate bundle adjustment in a real-time system. This parallel approach has been reused in many other systems since. The source-code for PTAM was released open-source which gave many researchers access to efficient, accurate camera tracking to build upon. More recently, ORB-SLAM [83] is a full SLAM system which is very robust and highly scalable. It has features not available in PTAM, such as fully automatic initialisation and large-scale loop closure. As implied by the name, the system uses highly efficient ORB features [108] and does so for every part of the SLAM pipeline: tracking, mapping, relocalisation and loop closure detection. This state-of-the-art SLAM system has been made available open-source and we make use of it later on.

Another sparse approach is the work of Forster *et al.* [45] with their Semi-direct Visual Odometry (SVO) system. This differs from traditional feature-based methods because it does not use features to compute correspondences between images, instead it uses the features to define patches on which to perform direct Lucas-Kanade style image alignment. Removing the costly feature extraction and matching during tracking makes the system very efficient and so it runs at more than twice the frame-rate of PTAM.

More recently with increased processing power dense and semi-dense methods have appeared. Engel *et al.* [41] introduce semi-dense visual odometry. Their system is efficient because it only processes pixels of the image which are

useful for tracking purposes. They do this by introducing an uncertainty measure (as in Section 2.4.3) and discarding pixels below a threshold. One of the reasons that this method works so well is that the distribution of errors in stereo is generally non-Gaussian due to the presence of outliers, while the inliers do tend to be well approximated by a Gaussian error model in inverse depth. The approach of [41] essentially discards pixels which would not fit with their Gaussian error model and the system works well. The surfel fusion approach we use permits a somewhat similar idea but allows possible outliers to be initially modelled and then discarded when they are known to be outliers. LSD-SLAM [40] extends the visual odometry of [41] into a full semi-dense SLAM system which handles loop-closures and scale-drift. They show how the system can scale to reconstruct entire buildings and can even be adapted to omni-directional cameras [16]. Mur-Artal and Tardós [84] also present a semi-dense mapping system. For camera tracking they make use of the highly accurate, feature-based ORB-SLAM and only use the keyframes, which have been optimised using bundle adjustment, to generate the semi-dense map. The claim is that such a system is more accurate than a fully semi-dense approach due to the improved local tracking of keyframes from bundle adjustment.

In this chapter we experiment with various different approaches to camera tracking; we look at two feature-based trackers, PTAM and ORB-SLAM, which are available open-source and also implement direct, dense tracking.

6.2 Surfel-based Fusion

Keller *et al.* [65] proposed a real-time method to fuse RGB-D data based on surfels (surface elements). We adapt some of the details of their fusion approach for use in a monocular reconstruction framework: the main differences being the criteria for deciding whether two surfels should be fused (we use a threshold that increases with depth) and the confidence function applied to each depth measurement.

The representation of the world (i.e. map) will be a collection of surfels. Each surfel is minimally specified by its position in space, a normal vector defining its orientation, and a radius. Additionally, each surfel can store some associated metadata. In our case we store a timestamp of when it was created, an RGB colour value and an uncertainty measure. Rendering of the map can

be done efficiently with standard graphics pipeline, in this case OpenGL. A surfel is rendered as a disk with its given position, normal, radius and colour.

As each new depth-map is generated it is fused with the existing surfels or creates new surfels when no data association can be found. To do the data association we render the current map as a set of points (zero radius) from the same viewpoint as the depth-map to create an index map I. It is possible that multiple surfels render to the same pixel, in which case only one of them will appear in the index map. To overcome this, Keller *et al.* render the index-map at 4×4 the resolution of the depth-map and then perform data association within a 4×4 window. We have found this to be unnecessary and prefer to have the increased performance of rendering at the lower, native resolution. However, we still search for data associations within a local window.

For each pixel $\mathbf{u}$ we look within the local window and find the best associated surfel from the index-map using the following criteria:

- We discard any surfel where $\frac{\left|D(\mathbf{u}) - D_{S_i}\right|}{D_{S_i}} > \tau_{depth}$ where D_{S_i} is the depth of surfel i in the local window. Unlike in [65] the fuse threshold increases with depth which better reflects the uncertainty.

- We discard any surfel where $\cos^{-1}(\hat{\mathbf{n}}(\mathbf{u}) \cdot \hat{\mathbf{n}}_{S_i}) > \tau_{angle}$ so that the angles between the normals are too large. We use a larger threshold of $\tau_{angle} = 60°$ due to increased noise from monocular depth-maps.

- From the remaining surfels choose the one whos centre is closest to the viewing ray emitted from pixel $\mathbf{u}$. Working in the camera frame of reference, if $\mathbf{d_u}$ is the direction of the ray emitted from $\mathbf{u}$ then the distance from the ray to surfel S_i centred at $\mathbf{v}_i$ is $\frac{|\mathbf{d_u} \times \mathbf{v}_i|}{|\mathbf{d_u}|}$.

If no surfels were remaining after the first two filtering steps then a new surfel is created with a timestamp of the current frame.

Once data associations have been made we combine the new depth measurement with the existing surfel information in a Kalman filter style approach. In reality we store the inverse of the variance, which we will refer to as confidence. Let $\lambda_s = \frac{1}{\sigma_s^2}$ be the confidence of an existing surfel in the model and let $\lambda_m = \frac{1}{\sigma_m^2}$ be the confidence of a new measurement associated with that surfel. Then, for any quantity $\mathbf{v}_s$ (position, normal, or colour) of the surfel, the

confidence-based update is as given by

$$\mathbf{v}_s \leftarrow \frac{\lambda_s \mathbf{v}_s + \lambda_m \mathbf{v}_m}{\lambda_s + \lambda_m}, \quad \lambda_s \leftarrow \lambda_s + \lambda_m. \tag{6.1}$$

If we re-write this in terms of the variances then we see that this is equivalent to a Kalman filter update:

$$\mathbf{v}_s \leftarrow \frac{\sigma_m^2 \mathbf{v}_s + \sigma_s^2 \mathbf{v}_m}{\sigma_s^2 + \sigma_m^2}, \quad {\sigma_s}^2 \leftarrow \frac{\sigma_s^2 \sigma_m^2}{\sigma_s^2 + \sigma_m^2}. \tag{6.2}$$

Keller *et al*. used an estimate of uncertainty based on the radius from the centre of the image. This seems to correspond to some approximate measure of uncertainty when working with Kinect images. In our monocular reconstruction system we use the depth uncertainty as outlined in Section 2.4.3 although modified with a maximum uncertainty (minimum confidence) so that texture-less regions will still fuse into the model.

We also remove surfels to help clean-up the model. For this we use the same heuristics as Keller *et al*.:

- Surfels are removed if their uncertainty is still less than some threshold after a time t_{max} since the surfel was created.
- We remove any surfels that occlude surfels which have just been fused with a measurement. This to to satisfy the free-space constraint.
- If after fusion there are surfels with very similar positions and normals we join them together into one.

As discussed in Section 6.1.1, Engel *et al*.'s LSD-SLAM works well because it only considers the pixels which are well modelled by a Gaussian inverse depth uncertainty. In our surfel fusion approach we are assuming a similar Gaussian model in the way we fuse and update surfels. But since we are aiming for a fully dense system, what happens to the pixels which do not obey the Gaussian model? There are essentially two stages where these outlier pixels can be filtered out. The first is in the stereo algorithm where we can apply outlier rejection using the consistency check outlined in Section 5.4. The other place where outliers are filtered is in the surfel model clean-up. Pixels which have outlier depth measurements will not be consistent from frame to frame, also they tend to have a low confidence weighting. The spurious nature of the outliers means that the surfel created by one is unlikely to be fused with any other

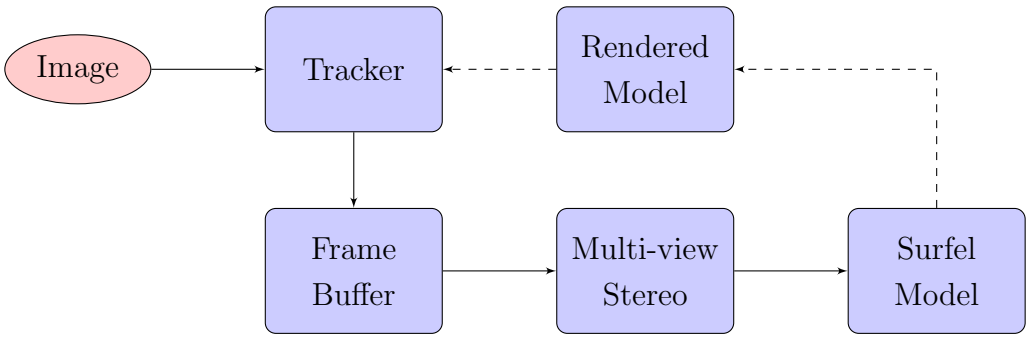


Figure 6.2: *Flowchart of our monocular reconstruction pipeline. When using a feature-based tracker the dashed line does not connect and, hence, the camera tracking is not linked to the 3D model. Using dense tracking closes the loop resulting in a more consistent model and framework.*

---

new measurements which means that it will be discarded after t_{max} when the confidence is still below the threshold.

6.3 System Overview

Figure 6.2 gives an overview of the pipeline of the system. Each new image from the camera is sent to the tracker (feature-based or dense) and, if successfully tracked, added to a buffer. Frames are then selected from the buffer (Section 6.3.2) to generate a depth-map using the methods outlined in Chapter 5. Finally, the depth-map is fused into the surfel model.

If using dense tracking then the surfel model is used directly for tracking purposes which, in theory, should result in a more consistent reconstruction. As the current implementation stands, when using a feature-based tracker the feature map is disjoint from the surfel map. An area of future work is to unify these maps.

We consider two implementations of the same system, which depends on the computation time of the multi-view stereo step. The single-threaded approach does all processing in the main thread. This includes, image pre-processing, camera tracking, depth-map generation, fusion and rendering. This system works well provided the stereo algorithm is fast enough so that the tracking can still run at a high frame-rate so we use this with the efficient

coarse-to-fine stereo method detailed in Section 5.3. The double-threaded approach shifts just the multi-view stereo part into a separate thread. Once a depth-map has been generated it is passed back to the main thread for fusion and more frames are grabbed to generate the next depth-map. We use this approach with our PatchMatch Stereo implementation which does not quite run at 30fps to allow fast camera tracking.

6.3.1 Tracking

We explore two approaches to camera tracking: the first is to use an existing feature-based tracking method and the second is to use dense alignment to the model as discussed in Section 6.4. Using existing feature-based trackers is the easiest approach to building a reconstruction system but the common approach of running bundle adjustment in a separate thread causes inconsistencies with the framework we have established. One can argue that the initial pose (before bundle adjustment) could be inaccurate and not globally consistent but the alternative of using the bundle adjusted pose introduces a delay in the system.

We are approaching the reconstruction problem with the application to robotics and augmented reality. In these situations the thing that really matters is the estimate of camera position right now and the estimate of the world geometry right now. Feature-based trackers may provide excellent globally consistent models with the use of bundle adjustment and loop closure but, as will be discussed further in Section 6.5.1, these things don't necessarily matter for robotics and AR where we are only interacting with things within our current field-of-view.

The difficulties of using an off-the-shelf feature-based tracker with a separate dense model is the fact that there are 2 separate maps which aren't consistent with each other. If bundle adjustment and loop closure cause deformations to the feature map then these must also be propagated to the dense map otherwise inconsistencies appear. In practice we have obtained quality reconstructions using this approach but once the size of the map increases and large-scale loop-closures need to be performed this issue will need to be addressed. One approach could be to associate each surfel with a key-frame and optimise the key-frame poses in the event of a loop-closure. However, such a system leads to tearing in the dense model at the boundaries between key-

frames. We hope that future work will be done into combined feature and dense map representations so that the two can be processed simultaneously.

The recent work of Whelan *et al.* [128] demonstrates that it is possible to perform loop-closures on dense surfel-maps obtained from an RGB-D sensor. As well as large scale loop closures their system is able to cope with smaller scale loopy trajectories that are often found when scanning with a 3D sensor, revisiting places which need more detail. Their system works by adding a temporal dimension to the surfel-map. Surfel fusion will not happen unless the timestamps of the surfels to be fused are within a threshold. This leads to a map which can have overlapping geometry from separate passes but can then be fused when a loop-closure is detected. Loop closure is performed by warping the space, rather than using key-frames, in order to prevent tearing of the dense geometry. Such a system shows promise for loop-closure in a dense RGB tracking and reconstruction system.

6.3.2 Frame selection

In order to ensure good quality results from multi-view stereo we need to have sufficient baseline between the frames used. We store a fixed size ring-buffer of past frames from which we select the frames used for stereo. For a new camera frame to be added to the buffer it must satisfy two conditions: First, the frame must be successfully tracked relative to the model. Second, the baseline between the new frame and the last frame added to the buffer must be greater than a threshold. This is to prevent the buffer filling up with very low baseline images in the event that the camera stays still.

Given a choice of stereo method from Chapter 5 we choose an optimum baseline β^* based on the the last views minimum and maximum observed depths: We approximate the median inverse depth γ as the mean of the minimum and maximum inverse depths. Then, the optimum baseline is computed as:

$$\beta^* = \frac{\tau f}{\gamma}, \tag{6.3}$$

where τ is a fixed, user defined parameter. The inspiration for this equation comes from the formula relating focal length, baseline and depth to disparity in rectified stereo. The variable τ is essentially the disparity of the median inverse depth value. Given β^* we then choose the frames from the buffer which are

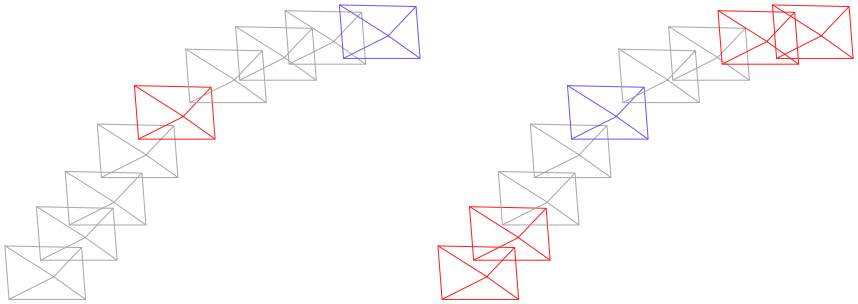


Figure 6.3: *Left: When using the coarse-to-fine stereo method we use the latest frame as the reference frame (blue) and another frame chosen from the buffer (red) for stereo. Right: When using PatchMatch Stereo we choose frames on either side of the reference frame. Therefore, the reference frame (blue) is in the middle of the buffer and one of more frames are chosen from either side (red) closest to the optimum baseline.*

---

closest to this optimal baseline from the current view.

Depending on the stereo method chosen, the reference frame we use may not be the most recent frame (Fig. 6.3). In our implementation, when using the coarse-to-fine stereo method of Section 5.3 we use the most recent frame as the reference and one other frame from the buffer to do stereo. In our PatchMatch Stereo method we prefer to use images from either side of the reference frame to help reduce occlusions. Therefore, the reference frame is not the latest but one from the middle of the buffer. We then choose frames for stereo from either side of the reference frame, close to the optimum baseline.

6.4 Dense, surfel-based tracking

Once we have a system capable of reconstructing coloured, dense 3D models it is possible to use dense tracking methods. The camera pose is found by direct image alignment with the rendered model.

We use the Lucas-Kanade style approach, as outlined in Section 2.9. In our pipeline we effectively invert the RGB-D odometry method of Steinbrücker *et al.* [120]. In their system they perform frame-to-frame tracking by warping the current image via its depth-map into the previous frame and minimising the photometric error. In the case of a monocular camera, the frames have no

depth. However, we can render the fused model to generate an RGB-D frame from the model. We then track the current monocular RGB image by warping the rendered RGB-D frame. We use a zero-motion prior, that is, the pose at which we render the model for tracking is the pose of the previous frame. In the case when tracking may have failed on the previous frame we use the last successfully tracked frame.

Let I_r be the RGB-D frame obtained by rendering the surfel model from the previous camera position and let I_c be the current camera image. The goal is to obtain the incremental transformation $\mathtt{T} \in \mathrm{SE}(3)$ between the two images. We define a warp function $\mathbf{w}()$ transforming points between two images as in Eq. 2.30:

$$\begin{aligned} \mathbf{w}(\mathbf{u}; \mathtt{T}) &= \mathbf{w}(\mathbf{u}; [\mathtt{R}|\, \mathbf{t}]) && (6.4) \\ &= \pi \left(\mathtt{K}\mathtt{R}\mathtt{K}^{-1} D(\mathbf{u}) \dot{\mathbf{u}} + \mathtt{K}\mathbf{t} \right). && (6.5) \end{aligned}$$

We wish to minimise the error between the current camera image and the rendered RGB-D frame warped into the frame-of-reference of the current camera:

$$E(\mathbf{x}) = \sum_{\mathbf{u} \in \Omega} \left(I_r(\mathbf{u}) - I_c(\mathbf{w}(\mathbf{u}; \mathtt{T}(\mathbf{x}))) \right)^2, \tag{6.6}$$

where Ω defines the set of valid pixels $\mathbf{u}$ such that both $\mathbf{u}$ and $\mathbf{w}(\mathbf{u}; \mathtt{T}(\mathbf{x}))$ are within the bounds of the images. As detailed in Section 2.9 we take an iterative approach to the optimisation and linearise around the current estimate after each iteration. This leads to minimising the following energy on iteration $(k+1)$:

$$E^{(k+1)}(\mathbf{x}) = \sum_{\mathbf{u} \in \Omega} \left(I_r(\mathbf{u}) - I_c(\mathbf{w}(\mathbf{u}; \hat{\mathtt{T}}^{(k)} \mathtt{T}(\mathbf{x}))) \right)^2, \tag{6.7}$$

where $\mathtt{T}^{(k)}$ is the estimate after the k-th iteration. After each iteration we compute the update to the estimate as:

$$\mathtt{T}^{(k+1)} = \mathtt{T}^{(k)} \mathtt{T}(\mathbf{x}^*). \tag{6.8}$$

With this forward-compositional formulation the computation of the jacobian is dependent on the current estimate $\mathtt{T}^{(k)}$. It is also possible to formulate and inverse-compositional approach:

$$E^{(k+1)}(\mathbf{x}) = \sum_{\mathbf{u} \in \Omega} \left(I_r(\mathbf{w}(\mathbf{u}; \mathtt{T}(\mathbf{x})) - I_c(\mathbf{w}(\mathbf{u}; \hat{\mathtt{T}}^{(k)})) \right)^2, \tag{6.9}$$

where the update vector $\mathbf{x}$ is now estimating the inverse of the transform. Hence, the update changes to:

$$\mathtt{T}^{(k+1)} = \hat{\mathtt{T}}^{(k)}\mathtt{T}^{-1}(\mathbf{x}^*) \tag{6.10}$$

$$= \hat{\mathtt{T}}^{(k)} \exp(-\mathbf{x}^*). \tag{6.11}$$

Note that with this inverse formulation we do not need to compute two warped images because we only evaluate the function and its derivatives at $\mathbf{x} = \mathbf{0}$. In the this case the warp function on I_r is just the identity: $\mathbf{w}(\mathbf{u}; \mathtt{T}(\mathbf{0})) = \mathbf{u}$. The derivative of the energy is now independent of the current estimate. This means that $\mathtt{J}^T\mathtt{J}$ is constant over all iterations (unless using an M-estimator) and does not need to be recomputed, saving processing time.

6.4.1 Removing outliers

We use the rendering of the model from the previous camera pose for tracking. Due to lighting changes and incomplete or inaccurate geometry there may be surfels rendered which are not consistent with that viewpoint. Therefore, to make tracking more robust we compare the RGB rendering of the model to the RGB image associated with that camera pose. Given the rendered reference frame I_r we compare to all corresponding pixels in a local neighbourhood of the reference camera image I_{r_c}. If there are no values within the threshold $\tau_{outlier}$ of each other then we remove those pixels from the tracking step. We look within a local neighbourhood, rather and just direct pixel correspondences, to make it more robust to slight tracking and aliasing errors. We introduce the weight $w_{outlier}$ which is zero when the pixel is removed from the tracking:

$$w_{outlier}(\mathbf{u}) = \begin{cases} 0, & \text{if } \|I_{r_c}(\mathbf{u}') - I_r(\mathbf{u})\| > \tau_{outlier} \text{ for all } \mathbf{u}' \in \mathcal{N}(\mathbf{u}), \\ 1, & \text{otherwise.} \end{cases} \tag{6.12}$$

We generally set $\tau_{outlier} = 20$ and the neighbourhood $\mathcal{N}(\mathbf{u})$ is a 5×5 window.

In practice, this process removes specular regions and inaccurate geometry caused by occlusions or motion of objects. Figure 6.4 demonstrates this.

6.4.2 Surfel weighting

Some surfels will provide better information for camera tracking than others. Therefore we apply a weighting scheme to each surfel based on a number of

Figure 6.4: *Outlier removal. The rendered model image (middle) is compared to the camera image (left). Pixels that are inconsistent are removed (right). Removing the outliers can make tracking more robust. Specularities, such as on the table in the bottom left of the images, are a typical occurrence which are removed by this processing.*

---

factors. The weights are defined in the coordinates of the rendered reference image I_r as we have direct pixel to surfel correspondences after the initial render pass.

To track the camera at pixel or sub-pixel accuracy, the resolution of the rendered model needs to match (or exceed) the resolution of the camera. In other words, the size of the surfels projected into the camera should be the same (or smaller) than the size of the pixels. Therefore we weight the pixels to be tracked by their projected surfel sizes.

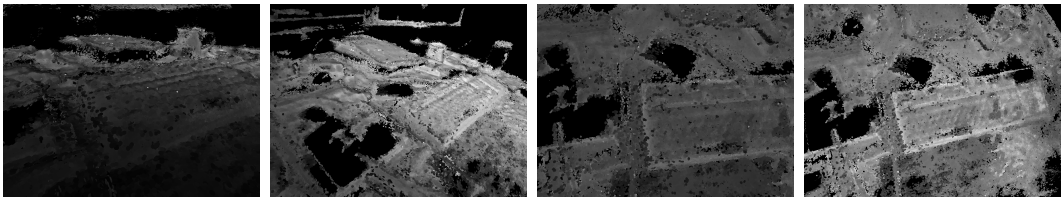
$$w_{radius} = \min\left(1, \frac{R_{pixel}}{R_{surfel}}\right), \tag{6.13}$$

where R_{pixel} is the radius of the pixel with the current rendered depth estimate and R_{surfel} is the radius of the surfel rendered to that pixel. If the radius of the surfel is larger than that of the pixel then it is down-weighted.

Another important surfel property to use in tracking is the normal. Every surfel has a normal and surfels which are fronto-parallel to the camera view will provide more reliable information than those at oblique angles. This leads to the following weight function:

$$w_{normal} = \hat{n}_z. \tag{6.14}$$

Surfels with normals pointing towards the camera ($\hat{n}_z = 1$) are given the most weight and surfels with normals pointing away from the camera are ignored in the rendering step (hence $w_{normal} > 0$).



(a) (b) (c) (d)

Figure 6.5: *Some examples of the weights used for surfel-based tracking. Most noticeable is the effect of surfel radius; points closer to the camera in (a) and (b) have been down-weighted because they lack the resolution for accurate tracking. In (c) the weights are lower than in (d) because the camera is closer. Fusion was turned off while capturing these images, if it had been active then the surfels would have reduced in size to match the resolution of the camera and the weights would be more even.*

The final weighting term is based on the surfel confidence. When rendering the reference frame from the model we have already discarded unstable surfels which have a confidence below a given threshold. However, there are still a range of confidences in the remaining surfels and it makes sense to give more weight to those which are more certain. To this end we introduce the following energy.

$$w_{confidence} = \min\left(1, \frac{\lambda}{\alpha\lambda_{thresh}}\right), \tag{6.15}$$

where λ is the confidence measure of the pixel, λ_{thresh} is the confidence threshold used when rendering the frame model for tracking, and $\alpha \geq 1$ is a tunable parameter. The pixels will then have weights on the range $[\alpha^{-1}, 1]$. We typically set $\alpha = 2$.

We combine all these weights as a product and an example of the final weights is given in Fig. 6.5.

6.4.3 M-Estimators

As well as the fixed, surfel-based weighting we make use of an M-estimator to further enhance robustness. M-estimators have been used extensively in the literature on dense tracking methods [40, 66, 76, 85]. The principal is to have a model predict whether a pixel is an outlier or not and then automatically down-weight the outliers to minimise their effect on the result. With a traditional

L_2 metric on the pixel errors, large outliers cause large gradients which can pull the solution away from the true optimum. By being able to automatically remove these pixels their effect is minimized. We outline the approach to using an M-estimator in Section 2.9.2.

For tracking we make use of Tukey M-estimator (as used in [76]) which is defined as follows:

$$\rho(x) = \begin{cases} \frac{c^2}{6}\left(1 - \left(1 - \left(\frac{x}{c}\right)^2\right)^3\right) & |x| \leq c, \\ \frac{c^2}{6} & \text{otherwise}, \end{cases} \tag{6.16}$$

$$w(x) = \begin{cases} \left(1 - \left(\frac{x}{c}\right)^2\right)^2 & |x| \leq c, \\ 0 & \text{otherwise}. \end{cases} \tag{6.17}$$

Because the Tukey function down-weights pixels with large errors, starting with a small value of the Tukey parameter, c, is bad for tracking; initially, almost all pixels have high residuals and end up with zero weight. Therefore, we start with a large value of the Tukey parameter to enable initial convergence towards the solution and then decrease it on subsequent iterations to remove any outliers.

Note that the M-estimator could be used to automatically remove the outliers processed in Section 6.4.1. However, given that it is easy enough to remove these pixels manually, it makes sense to do so and this enhances the convergence of the M-estimator problem.

Combining the M-estimator with the weights defined in Section 6.4.2, the final energy which we minimize for tracking purposes is:

$$E(\mathbf{x}) = \sum_{\mathbf{u}\in\Omega} w(\mathbf{u})\rho\Big(I_r(\mathbf{u}) - I_c(\mathbf{w}(\mathbf{u}; \mathtt{T}(\mathbf{x})))\Big), \tag{6.18}$$

where $\rho(x)$ is the Tukey function and $w(\mathbf{u})$ are the fixed weights computed as the product of the outlier, radius, normal and confidence weights introduced above.

6.4.4 Initialisation

When using a feature-based tracker the initialisation of the dense reconstruction system is trivial, it just follows from the initialisation of the tracker. When using a dense tracking system this is not so trivial.

Typically a feature-based system will initialise by taking two frames which involve camera translation and use feature matching to estimate the fundamental matrix. PTAM provides a manual interface for the user to select these two frames while ORB-SLAM does this automatically once sufficient translation baseline has been achieved.

When we want to use dense tracking with our system there are two ways that we have used to initialise. The first method is to just bootstrap the system with a feature-based tracker. Once a partial model of the world is constructed, we turn off the feature-based tracking and rely solely on the dense tracking from the dense model. The second method is to give a random or constant depth-map to the first camera frame, fuse it into the model and then track from it. This artificial initialisation is often good enough to start tracking a few frames to get better depth-maps and start refining the model. The constant depth assumption works especially well if the system looks at a roughly planar surface face-on. A similar random depth initialisation scheme is used by Engel *et al*. [40] to initialise their semi-dense SLAM system.

In practice we tend to use a feature-based tracker to bootstrap the system because it is more robust than the random initialisation method.

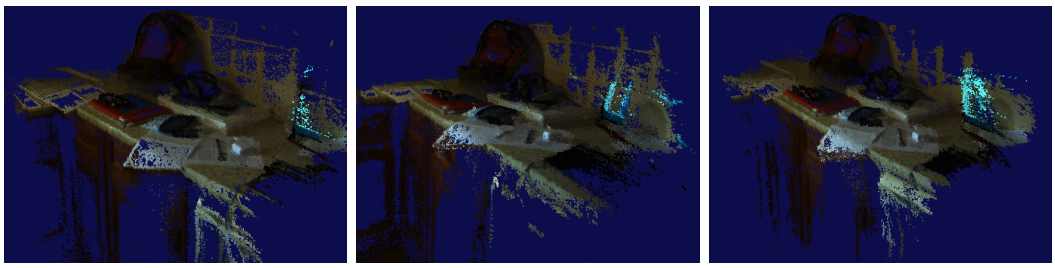
A future avenue to explore in fully-dense SLAM initialisation is joint optical flow and fundamental matrix estimation. Valgaerts *et al*. [125] use variational methods to do exactly that; Optical flow is estimated between two camera frames and then the weighting of an epipolar constraint is incrementally increased. This constrains the flow to be along the epipolar lines defined by the estimated fundamental matrix. The recovered fundamental matrix and optical flow can then be used to generate a depth-map and we now have a fully dense equivalent of the two frame initialisation used in feature-based systems.

6.5 Results

Figure 6.6 shows a number of scenes reconstructed with the described system. We also provide some screen-captures of the reconstruction process in Appendix A.

The reader will note that the models are not fully dense, they still show holes in some areas. The holes are regions where there has not been sufficient

Figure 6.6: *Reconstructions made using the reconstruction system outlined in the previous section. We use a coarse-to-fine approach to estimate depth maps and use ORB-SLAM for tracking.*



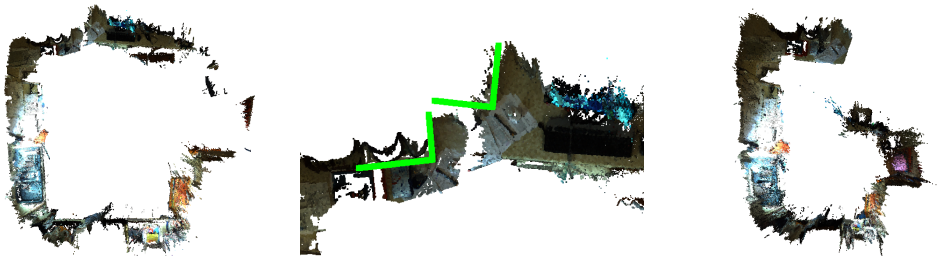
(a) ORB-SLAM, PM-S **(b)** ORB-SLAM, C2F **(c)** Dense tracking, C2F

Figure 6.7: *Reconstructions of the same scene using different stereo methods and different tracking methods.*

---

confidence to know the placement of the model. Regions where this occurs are usually due to texture-less regions. The depth measurements in these areas tend to be highly noisy, so are either discarded in the temporal consistency check (Section 5.4) or low-confidence surfels have been created but not enough of them have fused to pass the confidence threshold required. Unlike semi-dense systems we don't discard low-confidence measurements from the beginning but allow multiple low-confidence measurements the chance to fuse into a more confident estimate. Holes can also appear in regions which don't adhere to the Lambertian assumption, like specular regions, which causes erroneous depth measurements which don't fuse (further discussed in Section 6.5.2).

Figure 6.7 demonstrates the reconstruction of the same scene using different stereo methods and tracking methods. For tracking we use the feature-based ORB-SLAM and the direct, dense method outlined in Section 6.4. For depth-map generation we use the accelerated PatchMatch Stereo with Huber regularisation and compare to the fast coarse-to-fine method in Section 5.3. In this example, the is qualitatively not much difference between the various methods. However, our PM-S method only runs at 2Hz, compared to the C2F method running at the full 30Hz of the camera. This means that there is a noticeable delay between a camera viewing an area of the scene and it being fused into stable vertices in the reconstruction. This lag makes the system less user friendly, less intuitive and is also the reason why there is no example show of it combined with dense tracking – the dense tracking relies on the surfel model being updated quickly so it has something to track from, it was unable to cope with the lag in the system. With the C2F method the fused model appears almost instantaneously leading to intuitive scanning of unseen areas and effective dense tracking. Overall, even though the PatchMatch based stereo



(a) ORB-SLAM **(b)** ORB-SLAM, zoomed **(c)** Dense tracking

Figure 6.8: *Reconstructions of loop trajectory around a 4m×4m room to evaluate the drift of different tracking methods. The drift with ORB-SLAM is minimal, as indicated by the green angles in (b) (which should align). The dense tracking appears to suffer from significant scale drift.*

---

method gave higher quality individual depth maps there was not a significant difference between the two approaches, possibly due to many more frame being fused in the coarse-to-fine approach. We found that using the simple coarse-to-fine method and fusing at 30fps gave a much more usable system for online reconstruction.

To evaluate the tracking accuracy over larger scale trajectories we performed a reconstruction of a 4m×4m. The camera was kept an average of 50cm from the surface around the edge of the room so that the relative length of the trajectory was large. The results for both ORB-SLAM and dense tracking are shown in Fig. 6.8. Note that loop closure in ORB-SLAM was turned off. ORB-SLAM performs remarkably well with a very low level of drift. In contrast, the dense tracking exhibits a significant scale drift. The reason for which is currently unknown. Both trackers used the same camera calibration parameters. One possible explanation is bias in the depth estimates. If, for some reason, there was a systematic under-estimate of the depth values then tracking from these could lead to the scale drift.

When generating the reconstructions in Fig. 6.8 it was interesting to note that the dense tracking approach appeared to fuse data much more quickly into the model. Given that the fusion parameters were exactly the same between the two this indicates that the frame-to-frame depth estimates were more consistent leading to more surfels fusing and become stable quicker. This in a way makes sense because the tracking should be self-consistent with the current estimate of the model. The effect of this is further noticable in the right hand side of the

trajectory. The ORB-SLAM based reconstruction in Fig. 6.8a has siginifcant holes in the top right hand corner. The depth estimates here were not consistent enough to fuse into stable surfels. In contrast, the dense model was able to reconstruct this area, although there were a significant number of un-tracked frames when navigating that area.

6.5.1 Evaluation of Dense Reconstruction

The evaluation of dense reconstruction methods is a tricky problem in itself. As far as we are aware there are no evaluation datasets aimed specifically at real-time monocular reconstruction. We therefore take a look at some RGB-D evaluation datasets which provide some useful tools we can use to evaluate on our own datasets.

The TUM RGB-D Dataset [122] is aimed at benchmarking visual odometry and visual SLAM systems using RGB-D input. They provide a number of ground-truth trajectories to evaluate tracking accuracy but no 3D model data.

The ICL-NUIM Dataset [54] is a synthetically rendered RGB-D dataset. Each trajectory consists of highly realistic, ray-traced RGB-D images, again it is mainly aimed at evaluating depth-sensor based systems. Since it is synthetic, ground-truth trajectories as well as the 3D model are available for accurate benchmarking. The metrics which they provide are focused at computing the mean error of the full reconstruction after processing an entire trajectory. Because the dataset is mostly aimed at RGB-D reconstruction systems, the trajectories don't always contain sufficient translation for monocular tracking to work robustly. There are also large areas with very little texture which are challenging for passive vision systems. Indeed, when processing these datasets we found that only a partial model could be reconstructed before the tracking would fail. ORB-SLAM performed the best job of tracking but still suffered from significant drift due to lack of features in some parts and lack of baseline for matching on others. Our reconstruction system currently does not handle loop closures in ORB-SLAM so this feature is turned off. This means that drift in the tracking causes inconsistent models, as seen in Fig. 6.9.

Since these existing datasets do not map well to monocular systems, we do some basic evaluation of our reconstruction system using the desk dataset

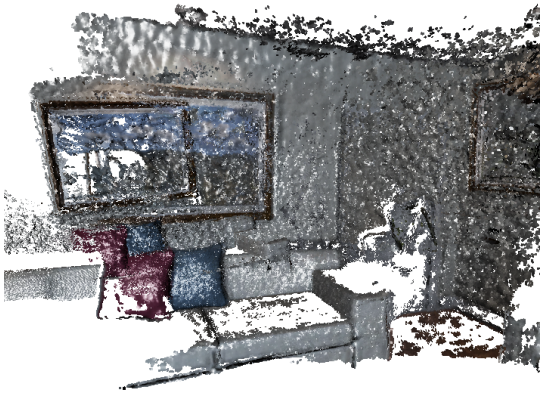


Figure 6.9: *Our reconstruction system does not yet handle loop closures. If drift accumulates the model severely deteriorates.*

---

in Section 2.11.2. The sequence is synthetically rendered using POVRay so we have ground-truth trajectory, depth-maps and a 3D model. We make use of the error metrics from the datasets detailed above and introduce our own metric as well.

To measure the trajectory error we use the absolute trajectory error metric as used in the TUM RGB-D dataset. They first align the trajectories using the method of Horn [61] and then compute the mean absolute error between corresponding poses. Since their dataset is aimed at RGB-D sequences the scale is known. We augment the trajectory alignment script to also solve for the scale difference between our captured monocular sequence and the ground-truth.

To measure the reconstruction error we use the SurfReg tool provided with the ICL-NUIM datasets. This system aligns the reconstructed point-cloud via ICP to the ground-truth and outputs the RMSE. Again, we must account for the scale difference when using this tool.

The results of using these metrics are shown in Table 6.1 where we compare the reconstruction when using sparse tracking (ORB-SLAM) versus direct, dense tracking. We see that the state-of-the-art feature-based SLAM system is able to give a more accurate trajectory and a more accurate reconstruction. The quality of the reconstruction is likely still the limiting factor for dense tracking. When comparing the final 3D reconstructions in Fig. 6.10 the dense-tracked version has more outliers and erroneous points. The build up of these bad surfels will affect the robustness and accuracy of the tracking. It is clear

	Trajectory	Reconstruction
Sparse	0.135474	0.0119082
Dense	0.213217	0.0204913

Table 6.1: *A comparison of the absolute trajectory error and the RMSE of the reconstruction for the desk dataset (Section 2.11.2). We compare the result using sparse tracking (ORB-SLAM) and direct, dense tracking.*

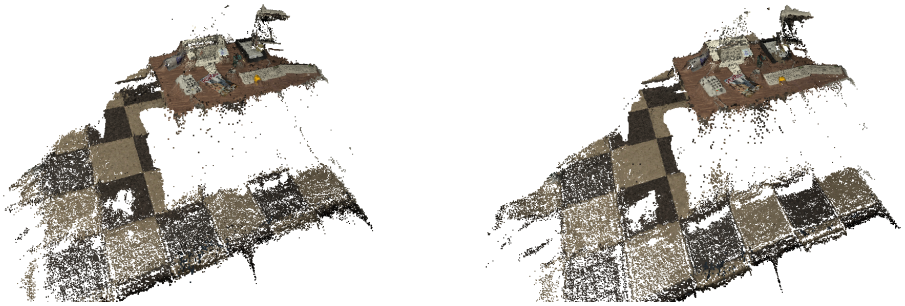
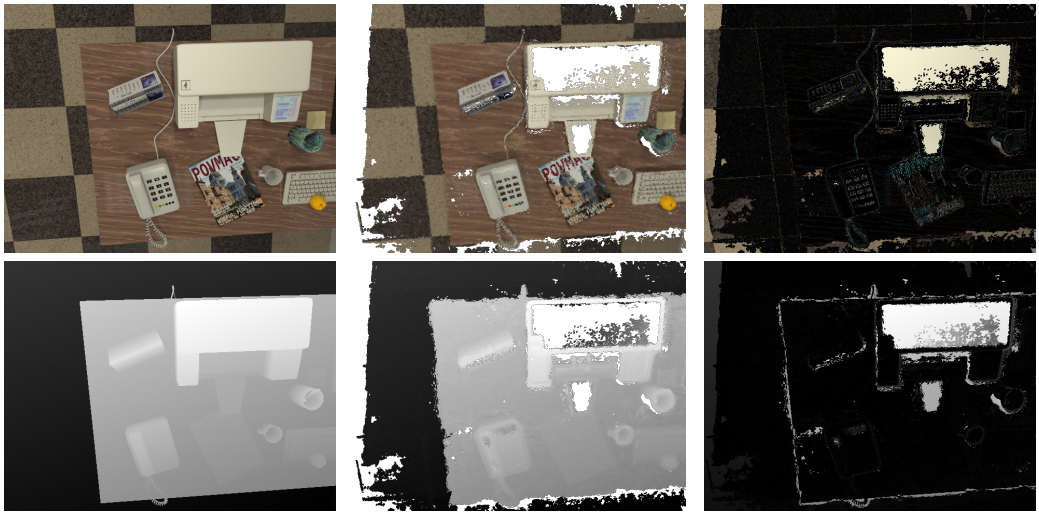


Figure 6.10: *Reconstruction of the desk dataset in Section 2.11.2 using ORB-SLAM for tracking (left) versus direct, dense tracking (right).*

to see that an overall in the quality of the depth-maps and reconstruction will have a positive effect on tracking performance as well.

We propose an alternative metric for reconstruction accuracy based on the current viewpoint error, which we feel is highly relevant to online reconstruction algorithms, especially when used in AR or robotic interaction scenarios. In both of these cases it is the accuracy of the current, incremental estimate of the world which is relevant. For AR the accuracy and realism of how virtual objects interact with the scene is dependent on this. For robots it is the current estimate of the geometry that will be used for interaction. Using the final reconstructed result does not seem to make sense in these situations. For example, a trajectory could have accumulated significant drift which will severely degrade the absolute trajectory and reconstruction errors. However, if the local, incremental estimate is good then the system could still be a good choice for AR or robotics.

Our metric compares the current model estimate to the ground-truth on every frame during incremental reconstruction. After a frame has been fused into the model we render the model from the current estimated camera pose, yielding an RGB image and a depth-map (Fig. 6.11b). We then compare this to



(a) Ground-truth **(b)** Reconstruction **(c)** Difference

Figure 6.11: *An example of comparing the ground-truth to the current, incremental estimate of the surfel model. We believe that evaluating the error on a per-frame, incremental basis is more relevant in scenarios like AR.*

---

the ground-truth RGB frame and depth-map for the current frame (Fig. 6.11a) and compute the mean absolute error between the two.

Figure 6.12 shows an example of this metric applied to both dense and sparse tracking on the desk dataset Section 2.11.2. We see that, in general, dense tracking gives a lower RGB error. This is to be expected because this is the exact error minimised to perform dense tracking. However, the sparse tracking appears to give slightly better depth estimates. Since the dense tracking is performed frame-to-model and the depth-map generation is dependent on the frame-to-frame transform, it may be possible to improve the depth-maps estimated from the dense method by doing a frame-to-frame pose refinement. This is a topic for future research.

6.5.2 Reconstructing Specular Surfaces

One of our initial motivations for dense 3D reconstruction was the ability to extend the work in Chapter 3 to capture surface light-fields on non-planar surfaces. However, one will note that the assumptions made of photo-consistency break down when surfaces exhibit specular lighting effects. In practice we have

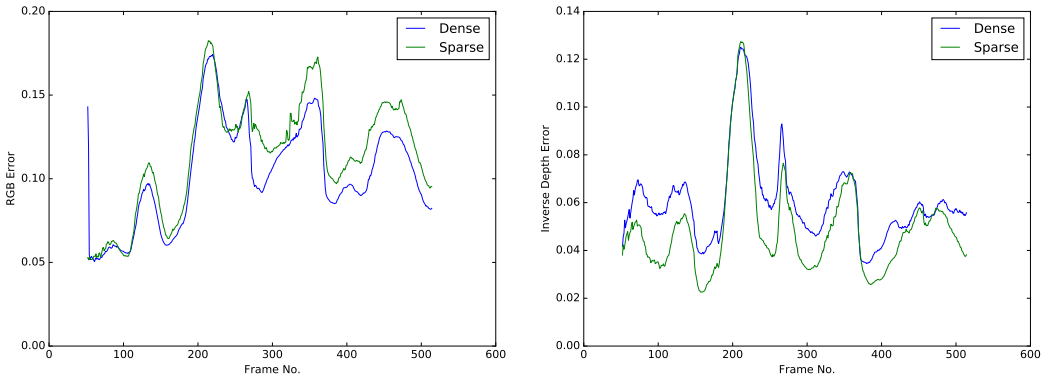


Figure 6.12: *The RGB error (left) and inverse depth error (right) of the incrementally reconstructed surfel model versus the ground truth RGB and inverse depth data. Here we compare the same algorithm run with both dense and sparse (ORB-SLAM) tracking.*

Figure 6.13: *The specularity on the lower right corner of the table (left) prevents the surface from being reconstructed (middle), resulting in a hole. However, with a change of viewpoint we are able to reconstruct the surface from views where there is no specular component (right).*

found that this isn't always a big problem.

Very often the specular component of the illumination is negligible except for a small set of viewing directions. What we observe is that when there is a strong specular component, the stereo algorithm does not give consistent depth from frame to frame. As a result, the points never fuse together to form stable surfels in the model and these do not pose a problem. If the viewing direction now changes to one without specularities the surface can be reconstructed accurately from the diffuse component.

Figure 6.13 shows an example of this happening. In the left image the surface of the table has not been reconstructed due to the large specular component

leading to inconsistent depth measurements. With the change of viewpoint in the right image the surface has been reconstructed from the diffuse component.

6.6 3D Surface Light-fields

Extending the planar surface light-fields of Chapter 3 to 3D surfaces is an extremely hard problem. Accurately recovering the geometric information and surface light-field is both computationally and memory intensive. This is made even more difficult by the decision to do acquisition using a single moving camera. The work in this chapter has taken a big step towards recovering the geometry required for the surface light-field but there is still work to do before we can apply the algorithms of Chapter 3 to general 3D surfaces.

The computational and memory demands for such a system are great. In a typical use case of our point-based fusion system we have approximately 4×10^5 surfels for a small desk sized scene and easily $> 1 \times 10^6$ for a larger scene. Compare that to typically $< 1 \times 10^5$ in our planar surface light-field work. The basic extension of the surface light-fields to 3D would be to allow a lumisphere per surfel, requiring 6.4GB of GPU memory to store the entire light-field, excluding surfel data (with 4000 points per lumisphere). A simple memory saving technique would be to subsample the surfels which will be selected to hold lumispheres, either focusing them densely on a particular region or spreading them sparsely over the entire scene.

Currently, the accuracy requirements are the biggest factor preventing our system generalising to 3D. Firstly, geometry must be estimated accurately enough to ensure that each light ray in a lumisphere corresponds to the exact same point on the surface. A slight error in the position can result in a significant misalignment, as shown in Fig. 6.14. In this case, the data in the lumisphere samples different points on the surface, instead of the same point from different direction, and so cannot give us true reflectance information. Secondly, if we would like to estimate illumination from the captured model then noise in the surface normals also becomes a significant problem. The angle between the incident ray and the reflected ray is twice the angle between the incident ray and the normal. Therefore a slight error in the orientation of the normal leads to double the error in estimating the light-source direction.

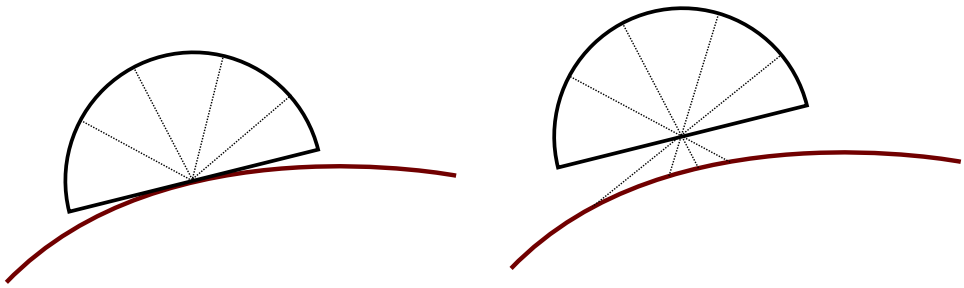


Figure 6.14: *When capturing a surface light-field the position of the lumisphere must be accurate to the true geometry (left). Otherwise, instead of sampling varying view directions of the same surface point we sample different points on the surface (right).*

Wood *et al.*'s [131] work on general 3D surface light-fields makes use of a laser scanner to acquire the 3D geometry of the object, so it is known very accurately. It would be possible to adopt such an approach while still using a hand-held camera to do the acquisition of the light-field, however, the focus of this thesis is to have the entire pipeline fully monocular. Currently, the monocular reconstruction system demonstrated above is not accurate enough for full surface light-field capture but we believe that the basic building blocks are in place. A monocular 3D reconstruction system is a union of a number of already complex systems (camera tracking, depth estimation, fusion) and to work to its full potential requires each component to be working at its best and complementing the other parts.

Aside from improving the 3D reconstruction there could be other approaches to overcome the accuracy problem. Even if the geometry is inaccurate, the data stored in the captured lumispheres is still valid light-field data; they store the intensity of the light ray passing through that 3D point in space. If the geometry could be refined in some way, via post-processing or otherwise then the data in the captured light-field could be remapped to lumispheres on the real surface. An interesting area of future work is to use light-field data captured from the inaccurate geometry to refine the 3D model.

7

Conclusion and Future Work

In this thesis we have taken some steps towards enhanced scene understanding from a monocular camera. There is still a lot to learn and to be discovered about how to efficiently interpret the huge wealth of data from a monocular video feed and we hope that the work presented here can help direct some of that future work.

In Chapter 3 we showed that real-time acquisition of planar surface light-fields is possible when harnessing the increased processing power of modern GPUs. We went on to use the surface light-field to estimate the environment map, in essence demonstrating the use us of the specular surface as a planar light probe. The recovered environment map can then be used to improve the realism of virtual objects placed in augmented reality, this was demonstrated by placing virtual objects on a specular surface while casting realistic shadows and handling specular occlusions. Finally, in the end of the chapter we used dense image alignment techniques to recover the bump-map of the surface, further enhancing the realism of augmented reality.

The work in Chapter 4 introduced a new class of mesh deformations known as *sculptural stylization*. Our motivation was based on the fact that sculptors do not always aim for geometric accuracy but will manipulate shapes to change the interaction of the illumination, reflectance and shadows which brings depth to a sculpt in a material of constant albedo. We built up a mathematical model of some techniques used in classical sculpture and put it into optimisation framework allowing the user to enhance, smooth and deform a mesh in an entirely new way.

The research in Chapters 5 and 6 was undertaken to try to expand the work of Chapter 3 to work with general 3D surfaces. In Chapter 5 we first gave an overview a few classic and high performance methods for real-time stereo and introduced a temporal consistency check to aid in the removal of outliers.

We followed this with an in-depth performance analysis of PatchMatch Stereo and presented a number of tweaks and modifications to help the once offline algorithm to run at real-time rates. Chapter 6 then built upon the real-time stereo work, integrating it into a full 3D reconstruction system. We outlined the key components needed to build a such a system and demonstrated it with both feature-based tracking and dense, camera-to-model tracking. During evaluation we showed that the quality of the reconstruction and dense tracking was not yet able to keep up with the accuracy of a state-of-the-art feature-based tracking method.

7.1 Discussion and Future Work

In the introduction we discussed the potential for lighting and reflectance information to aid in recognising materials. A real-time surface light-field capture tool, such as the one we presented, could be used to acquire training data for material classification problems. Designing a system to capture large quantities of data and finding the best representation on which to train a classifier are big challenges to overcome to achieve this goal. However, making use of surface reflectance properties has the potential to dramatically increase classification accuracy compared to current single image systems.

While we were not able to successfully expand our light-field capture work to general 3D surfaces, it is something we still believe to be achievable. The main limitation of our approach was the accuracy of the 3D reconstruction. We accepted a huge challenge by attempting to do everything from a monocular input, many components needed to work together in unison to ensure a consistent model and we don't feel the maximum potential of the system was achieved. To accelerate future work in the area it may be advantageous to capture the geometry initially using an active RGB-D sensor, like those from Primesense, and then switching to a monocular camera for surface light-field acquisition. High quality RGB-D reconstruction systems already exist and hence the complexity of the system and dependency on individual components to perform their best is greatly reduced.

In our experiments the feature-based tracker outperformed the dense tracking approach. With increases in the quality of the depth-maps from stereo we expect the dense tracking to improve but future work should also focus on

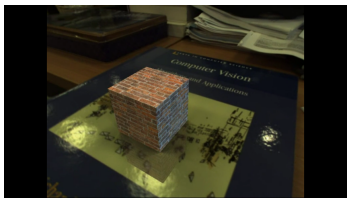
finding a way to tightly integrate live dense reconstruction with feature-based tracking. The maps from both dense stereo and feature-tracking should be consistent with each other and correctly handle bundle adjustment and loop closures.

We believe our work on sculptural stylization to be the first in this area of research and we hope that it will inspire more innovation in this field. So much work has been done stylizing 2D images to look like the work of famous artists that surely it is natural to extend much of this work to the 3D domain. We predict that decreasing costs of scanning hardware and easy access 3D printing services will help drive research in this area.

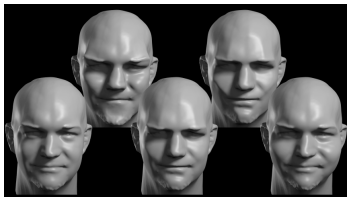
Appendices

A

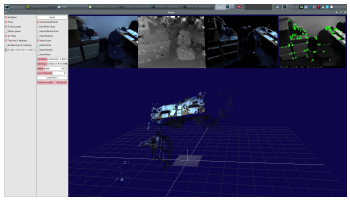
Video Appendix



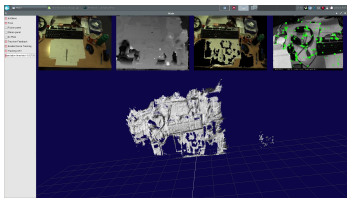
Real-Time Surface Light-field Capture for Augmentation of Planar Specular Surfaces
https://youtu.be/pky822zG4hM



Interactive 3D Face Stylization Using Sculptural Abstraction
https://youtu.be/CwclaQ1NpeM



Monocular reconstruction of a desk scene using ORB-SLAM for camera tracking
https://youtu.be/KaGhgzVEvDc



Monocular reconstruction of a desk using dense, surfel-based tracking
https://youtu.be/IKLp1Af_vdE

B

Lie Group Generators

B.1 Special Orthogonal Group: SO(3)

$$G_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}, G_1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, G_2 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

B.2 Special Euclidean Group: SE(3)

$$G_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G_1 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$G_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, G_5 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

C

Huber Conjugate

I've seen many unsatisfying derivations of the Legendre-Fenchel conjugate of the Huber norm, I hope to clarify here. Let us first recall the definition of the Huber function and the Legendre-Fenchel transform:

$$F(\mathbf{x}) = |\mathbf{x}|_\alpha = \begin{cases} \frac{|\mathbf{x}|^2}{2\alpha}, & |\mathbf{x}| \leq \alpha \\ |\mathbf{x}| - \frac{\alpha}{2}, & \text{otherwise.} \end{cases} \tag{C.1}$$

$$F^*(\mathbf{y}) = \max_{\mathbf{x}} \left(\mathbf{y}^T\mathbf{x} - |\mathbf{x}|_\alpha\right) \tag{C.2}$$

$$= \max\left(\max_{|\mathbf{x}|\leq\alpha}\left(\mathbf{y}^T\mathbf{x} - \tfrac{|\mathbf{x}|^2}{2\alpha}\right), \max_{|\mathbf{x}|>\alpha}\left(\mathbf{y}^T\mathbf{x} - |\mathbf{x}| + \frac{\alpha}{2}\right)\right) \tag{C.3}$$

Let's look at the first case $|\mathbf{x}| \leq \alpha$. This is a simple quadratic so we can find the maximum by setting the derivative w.r.t $\mathbf{x}$ to 0. This gives that $\mathbf{x} = \alpha\mathbf{y}$. Substituting this into the function and the constraint we get:

$$\max_{\mathbf{x}\leq\alpha}\left(\mathbf{y}^T\mathbf{x} - \frac{|\mathbf{x}|^2}{2\alpha}\right) = \frac{\alpha}{2}|\mathbf{y}|^2, \text{ for } |\mathbf{y}| \leq 1 \tag{C.4}$$

Now we look at the other case. The maximum of $\mathbf{y}^T\mathbf{x} - |\mathbf{x}|$ must be when $\mathbf{x}$ is in the same direction as $\mathbf{y}$ such that $\mathbf{x} = \lambda\frac{\mathbf{y}}{|\mathbf{y}|}$ for some $\lambda > 0$. Now, considering the restricted domain $|\mathbf{x}| > \alpha \implies \lambda > \alpha$ we can compute the following:

$$\max_{|\mathbf{x}|>\alpha}\left(\mathbf{y}^T\mathbf{x} - |\mathbf{x}| + \frac{\alpha}{2}\right) = \max_{\lambda>\alpha}\left(\lambda(|\mathbf{y}| - 1) + \frac{\alpha}{2}\right) = \begin{cases} \alpha|\mathbf{y}| - \frac{\alpha}{2}, & |\mathbf{y}| \leq 1 \\ \infty, & \text{otherwise.} \end{cases} \tag{C.5}$$

If we now look at the case when $|\mathbf{y}| \leq 1$ we see that the quadratic function in Eq. C.4 is always greater than the linear term in Eq. C.5. Therefore, on this domain the quadratic term wins in Eq. C.3 and outside this domain the infinity term in Eq. C.5 wins. We can now write down the conjugate:

$$F^*(\mathbf{y}) = \begin{cases} \frac{\alpha}{2}|\mathbf{y}|^2, & |\mathbf{y}| \leq 1 \\ \infty, & \text{otherwise.} \end{cases} \tag{C.6}$$

D

Derivatives on Graphs

Zhou and Schölkopf [135] give a good definition of a graph:

> A graph $G = (V, E)$ consists of a finite set V, together with a subset $E \subseteq V \times V$. The elements of V are the *verticies* of the graph, and the elements of E are the *edges* of the graph. We say that an edge e is *incident* on vertex v if e starts from v. [...] A graph is *undirected* when the set of edges is *symmetric*, i.e. for each edge $[u, v] \in E$ we also have $[v, u] \in E$. [...] A graph is *weighted* when it is associated with a function $\omega : E \to \mathbb{R}_+$ which is symmetric, i.e. $\omega([u, v]) = \omega([v, u])$, for all $[u, v] \in E$. The *degree* function $d : V \to \mathbb{R}_+$ is defined to be $d(v) = \sum_{u \sim v} \omega([u, v])$, where $u \sim v$ denote the set of vertices *adjacent with* v, i.e. $[u, v] \in E$. Let $\mathcal{H}(V)$ denote the Hilbert space of real-valued functions endowed with the usual inner product $\langle f, g \rangle_{\mathcal{H}(V)} = \sum_{v \in V} f(v)g(v)$, for all $f, g \in \mathcal{H}(V)$. Similarly define $\mathcal{H}(E)$.

From this definition it is easy to see that a triangular mesh is a type of undirected graph.

To define the gradient of a function on a graph we take inspiration from the finite difference gradient (in this case forward difference):

$$f'_h(x) = \frac{f(x + h) - f(x)}{h}. \tag{D.1}$$

In this case the change of f is divided by the finite difference length. We use a similar method on the graph. Let $g(u, v)$ be some distance function on the graph (for example, edge length) then we define the gradient along edge $[u, v]$ to be:

$$(\nabla f)([u, v]) = \frac{f(v) - f(u)}{g(u, v)} = \omega([u, v])\big(f(v) - f(u)\big), \tag{D.2}$$

where we define the edge weight $\omega([u, v])$ to be the inverse of the distance

function. We define the divergence operator using the following inner product relationship:

$$\langle \nabla f, F \rangle_{\mathcal{H}(E)} = \langle f, -\operatorname{div} F \rangle_{\mathcal{H}(V)}. \tag{D.3}$$

We can then derive the divergence explicitly:

$$\begin{aligned}
\langle \nabla f, F \rangle &= \sum_{[u,v] \in E} \nabla f([u,v]) F([u,v]) \\
&= \sum_{v \in V} \sum_{u \sim v} \omega([u,v]) \big(f(v) - f(u)\big) F([u,v]) \\
&= \sum_{v \in V} \sum_{u \sim v} \omega([u,v]) f(v) F([u,v]) - \sum_{v \in V} \sum_{u \sim v} \omega([u,v]) f(u) F([u,v]) \\
&= \sum_{v \in V} \sum_{u \sim v} \omega([u,v]) f(v) F([u,v]) - \sum_{v \in V} \sum_{u \sim v} \omega([u,v]) f(v) F([v,u]) \\
&= -\sum_{v \in V} f(v) \sum_{u \sim v} \omega([u,v]) \big(F([v,u]) - F([u,v])\big) \\
&= -\sum_{v \in V} f(v) \operatorname{div} F(v) = \langle f, -\operatorname{div} F \rangle.
\end{aligned}$$

Hence, the divergence operator is:

$$\operatorname{div} F(v) = \sum_{u \sim v} \omega([u,v]) \big(F([v,u]) - F([u,v])\big). \tag{D.4}$$

We now combine define the Laplacian in terms of the divergence of the gradient operator:

$$\begin{aligned}
(\Delta f)(v) &= -\frac{1}{2} \operatorname{div}(\nabla f)(v) && \text{(D.5)} \\
&= \sum_{u \sim v} \omega^2([u,v]) \big(f(u) - f(v)\big) && \text{(D.6)}
\end{aligned}$$

It is easy to see that this is the same as the mesh Laplacian in Section 2.10.1.

E

PatchMatch Propagation Kernels

```
__global__ void DownPassKernel(Image planes, ...) {
    const int x = threadIdx.x;

    if(x>width-1) return;

    for( int y = 1; y<height; y++)
    {
        planes(x,y) = GetMinCostPlane(
                            planes(x,y), //current
                            planes(x-1,y-1), //above, left
                            planes(x,y-1), //above
                            planes(x+1,y-1) //above, right
                            );

        //Synchronize threads in the block
        __syncthreads();
    }
}
```

```
__global__ void ScanlinePassULKernel(Image planes, ...) {
    const int y = threadIdx.x;
    const int smax = width+height-1;

    if(y>height-1) return;

    for(int s=0; s<smax; s++)
    {
        const int x = s - y;

        if( x<0 ) continue;
        if( x>width-1 ) return;

        planes(x,y) = GetMinCostPlane(
                            planes(x,y), //current
                            planes(x-1,y), //left
                            planes(x,y-1) //above
                            );

        //Synchronize threads in the block
        __syncthreads();
    }
}
```

```
__global__ void ParallelKernel(
        Image old_planes, Image new_planes, ...) {
    const int x = blockIdx.x*blockDim.x + threadIdx.x;
    const int y = blockIdx.y*blockDim.y + threadIdx.y;

    if(x>width-1 || y>height-1) return;

    new_planes(x,y) = GetMinCostPlane(
                        old_planes(x,y), //current
                        old_planes(x-1,y), //left
                        old_planes(x+1,y), //right
                        old_planes(x,y-1), //above
                        old_planes(x,y+1) //below
                        );
}
```

```
__global__ void JumpFloodKernel(
        Image old_planes, Image new_planes, int step, ...) {
    const int x = blockIdx.x*blockDim.x + threadIdx.x;
    const int y = blockIdx.y*blockDim.y + threadIdx.y;

    if(x>width-1 || y>height-1) return;

    new_planes(x,y) = GetMinCostPlane(
                        old_planes(x,y), //current
                        old_planes(x-step,y), //left
                        old_planes(x+step,y), //right
                        old_planes(x,y-step), //above
                        old_planes(x,y+step) //below
                        );
}
```

To call these CUDA kernels we must also specify the configuration arguments:

```
DownPassKernel<<<1,width>>>(planes,...);
UpPassKernel<<<1,width>>>(planes,...);
LeftPassKernel<<<1,height>>>(planes,...);
RightPassKernel<<<1,height>>>(planes,...);
ScanlinePassKernel<<<1,max(width,height)>>>(planes,...);

dim3 threadsPerBlock(16,16);
dim3 blocksPerGrid(
        width/16 + ( width%16 ? 1 : 0 ),
        height/16 + ( height%16 ? 1 : 0) );

ParallelKernel<<<blocksPerGrid,threadsPerBlock>>>(old_planes,
   new_planes,...);
for( int power=3; power>=0; power--) {
    int step = 1 << power; //power of 2
    JumpFloodKernel<<<blocksPerGrid,threadsPerBlock>>>(
       old_planes, new_planes, step,...);
}
```

F

Lanteri Constraints

In selecting geometric constraints to preserve individual facial characteristics, we identified seven passages of Lanteri [1911, 1985] that correspond to nine relative measurements between points on the surface of the model. All of these passages refer to maintaining distances between points, but our optimization only modifies the face region, so measurements to points such as the ears or the top of the sternum are treated as absolute positional constraints rather than relative ones. The table below provides these seven passages, along with our interpretation of each as a distance constraint.

Page	Passage	Constraint
15	"...with your calipers measure the distance from corner to corner of the mouth, and set this distance off on your horizontal line."	distance between mouth corners
19	"measure with calipers the length of the nose"	distance from bridge to tip of nose
43–44	"calculate or measure (from the place where you fixed the articulation of sternum and collar-bone) the height of the chin..."	absolute position of chin tip
48	"Measure on your model the distance from the notch of the ear to the most projecting part of the nose-tip (Fig. 38) (take this measure on both sides, for you will frequently find that the distance on right and left side varies)"	absolute position of nose tip
50	"...the distance from the chin to the eyebrows... by describing an arc on the model, in order to find out if the eyebrows are of the same height on either side..."	distance from chin to each eyebrow (2)
53	"...model the upper jaw, marking at once the two corners of the mouth, well observing their relation to the size of the nostrils."	distance from each corner of mouth to same side of nose (2)
58–59	"A careful measurement from inner corner to inner corner..."	distance between inner corners of eyes

References

[1] E. H. Adelson and J. R. Bergen. The Plenoptic Function and the Elements of Early Vision. In *Computational Models of Visual Processing*, pages 3–20, 1991.

[2] S. Agarwal, M. K., and Others. Ceres solver. http://ceres-solver.org.

[3] H. Alismail, B. Browning, and S. Lucey. Direct visual odometry using bit-planes. *arXiv preprint arXiv:1604.00990*, 2016.

[4] J. Asaro. Planes of the head. http://www.planesofthehead.com/.

[5] C. Bailer, M. Finckh, and H. P. Lensch. Scale robust multi view stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 398–411, 2012.

[6] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. PatchMatch: a randomized correspondence algorithm for structural image editing. In *Proceedings of SIGGRAPH*, 2009.

[7] T. Beeler, B. Bickel, P. Beardsley, B. Sumner, and M. Gross. High-quality single-shot capture of facial geometry. *Proceedings of SIGGRAPH*, 29(3), 2010.

[8] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] F. Besse, C. Rother, A. Fitzgibbon, and J. Kautz. PMBP: PatchMatch Belief Propagation for Correspondence Field Estimation. *International Journal of Computer Vision (IJCV)*, 110(1):2–13, Oct. 2014.

[10] P. Bhat, S. Ingram, and G. Turk. Geometric texture synthesis by example. In *Proceedings of the Eurographics Symposium on Geometry*

Processing, pages 41–44, 2004.

[11] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *Proceedings of SIGGRAPH*, pages 187–194, 1999.

[12] M. Bleyer, C. Rhemann, and C. Rother. PatchMatch Stereo — Stereo Matching with Slanted Support Windows. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2011.

[13] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. PriMo: coupled prisms for intuitive surface modeling. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, pages 11–20, 2006.

[14] K. Bredies, K. Kunisch, and T. Pock. Total Generalized Variation. *SIAM Journal of Imaging Sciences*, 3(3):492–526, 2010.

[15] R. Bridson. Fast Poisson disk sampling in arbitrary dimensions. In *Proceedings of SIGGRAPH*, volume 2007, page 5, 2007.

[16] D. Caruso, J. Engel, and D. Cremers. Large-scale direct SLAM for omnidirectional cameras. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 141–148. IEEE, 2015.

[17] A. Chambolle and T. Pock. A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[18] D. Chekhlov, A. Gee, A. Calway, and W. Mayol-Cuevas. Ninja on a Plane: Automatic Discovery of Physical Planes for Augmented Reality Using Visual SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[19] M. Chuang and M. Kazhdan. Interactive and anisotropic geometry processing using the screened Poisson equation. *Proceedings of SIGGRAPH*, 30(4), 2011.

[20] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *Proceedings of SIGGRAPH*, volume 23, pages 905–914, 2004.

[21] F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein,

T. Funkhouser, and S. Rusinkiewicz. Where do people draw lines? *Proceedings of SIGGRAPH*, 27(3), 2008.

[22] A. Concha and J. Civera. Using superpixels in monocular SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 365–372. IEEE, 2014.

[23] A. Concha and J. Civera. DPPTAM: Dense Piecewise Planar Tracking and Mapping from a monocular sequence. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015.

[24] A. Concha, W. Hussain, L. Montano, and J. Civera. Manhattan and piecewise-planar constraints for dense monocular mapping. In *Proceedings of Robotics: Science and Systems (RSS)*, 2014.

[25] A. Concha, W. Hussain, L. Montano, and J. Civera. Incorporating scene priors to dense monocular mapping. *Autonomous Robots*, 39(3):279–292, 2015.

[26] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, pages 51–72, 1986.

[27] G. Coombe, C. Hantak, A. Lastra, and R. Grzeszczuk. Online Construction of Surface Light Fields. In *Eurographics Symposium on Rendering*, 2005.

[28] Copyright *Physics_Dude*, licensed under CC BY-NC 3.0. 3D Scan of Sitting Fox Decor. http://www.thingiverse.com/thing:166984, 2013.

[29] Copyright *purita*, licensed under CC BY-NC 3.0. Lion Head. http://www.thingiverse.com/thing:215420, 2013.

[30] A. Davis, M. Levoy, and F. Durand. Unstructured Light Fields. In *Proceedings of Eurographics*, 2012.

[31] A. J. Davison, W. W. Mayol, and D. W. Murray. Real-Time Localisation and Mapping with Wearable Active Vision. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2003.

[32] A. J. Davison, N. D. Molton, I. Reid, and O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(6):1052–1067, 2007.

[33] P. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH*, 1997.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[35] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques*, pages 317–324, 1999.

[36] F. Devernay and O. Faugeras. Straight lines have to be straight. *Machine Vision and Applications*, 13:14–24, 2001.

[37] E. Eade and T. Drummond. Unified Loop Closing and Recovery for Real Time Monocular SLAM. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2008.

[38] Ego-3D. Ego-3d. http://www.ego-3d.de/.

[39] M. Eigensatz, R. W. Sumner, and M. Pauly. Curvature-domain shape processing. *Computer Graphics Forum*, 27(2):241–250, 2008.

[40] J. Engel, T. Schoeps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[41] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2013.

[42] FabliTec. FabliTec 3D Scanner. http://www.fablitec.com/.

[43] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning Optical Flow with Convolutional Networks. In *ICCV*, 2015.

[44] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. DeepStereo: Learning to Predict New Views from the World's Imagery. *arXiv preprint arXiv:1506.06825 [cs.CV]*, 2015.

[45] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[46] R. Gal, O. Sorkine, T. Popa, A. Sheffer, and D. Cohen-Or. 3D collage: expressive non-realistic modeling. In *Proceedings of the International Symposium on Non-Photorealistic Animation and Rendering*, pages 7–14, 2007.

[47] S. Gao, C. Werner, and A. A. Gooch. Morphable guidelines for the human head. In *Proceedings of the Symposium on Computational Aesthetics*, pages 21–28, 2013.

[48] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proceedings of SIGGRAPH*, 1996.

[49] G. Graber, J. Balzer, S. Soatto, and T. Pock. Efficient minimal-surface regularization of perspective depth maps in variational stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 4, 2015.

[50] G. Graber, T. Pock, and H. Bischof. Online 3D reconstruction using convex optimization. In *Workshop on Live Dense Reconstruction from Moving Cameras at ICCV*, 2011.

[51] M. D. Grossberg and S. K. Nayar. What is the Space of Camera Response Functions? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[52] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison. Applications of the Legendre-Fenchel transformation to computer vision problems. Technical Report DTR11-7, Imperial College London, 2011.

[53] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison. Real-Time Camera Tracking: When is High Frame-Rate Best? In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012.

[54] A. Handa, T. Whelan, J. B. McDonald, and A. J. Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[55] K. Hara, K. Nishino, and K. Ikeuchi. Light Source Position and Reflectance Estimation from a Single View without the Distant Illumination Assumption. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27:493–505, 2005.

[56] S. Heber and T. Pock. Shape from light field meets robust PCA. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[57] P. Heise, S. Klose, B. Jensen, and A. Knoll. PM-Huber: PatchMatch with Huber Regularization for Stereo Matching. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2013.

[58] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of SIGGRAPH*, pages 327–340, 2001.

[59] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of SIGGRAPH*, pages 517–526, 2000.

[60] D. D. Holm. *Geometric Mechanics Part II: Rotating, Translating and Rolling*. Imperial College Press, 2008.

[61] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[62] i.materialize. i.materialize Website. http://i.materialise.com/.

[63] J. Jachnik, D. B. Goldman, L. Luo, and A. J. Davison. Interactive 3D Face Stylization Using Sculptural Abstraction. *arXiv preprint arXiv:1502.01954 [cs.GR]*, 2015.

[64] J. Jachnik, R. A. Newcombe, and A. J. Davison. Real-Time Surface Light-field Capture for Augmentation of Planar Specular Surfaces. In *Proceedings of the International Symposium on Mixed and Augmented*

Reality (ISMAR), 2012.

[65] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. In *Proc. of Joint 3DIM/3DPVT Conference (3DV)*, 2013.

[66] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[67] A. Kirillov. *An introduction to Lie groups and Lie algebras*, volume 113. Cambridge University Press Cambridge, 2008.

[68] G. Klein and D. W. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[69] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, 2012.

[70] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1817–1824. IEEE, 2011.

[71] E. Lanteri. *Modelling; A Guide for Teachers and Students*. Chapman and Hall, 1911.

[72] E. Lanteri. *Modelling and Sculpting the Human Figure*. Dover Art Instruction, 1985.

[73] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *Proceedings of SIGGRAPH*, pages 85–94, 2000.

[74] M. Levoy and P. Hanrahan. Light Field Rendering. In *Proceedings of SIGGRAPH*, 1996.

[75] H. Li, B. Adams, L. J. Guibas, and M. Pauly. Robust single-view geometry and motion reconstruction. *SIGGRAPH Asia*, 28(5), 2009.

[76] S. J. Lovegrove. *Parametric Dense Visual SLAM*. PhD thesis, Imperial College London, 2011.

[77] J. Lu, H. Yang, D. Min, and M. N. Do. PatchMatch Filter: Efficient Edge-Aware Filtering Meets Randomized Search for Fast Correspondence Field Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[78] B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1981.

[79] Lytro, Inc. Lytro Website. http://www.lytro.com/, 2012.

[80] C. Madsen and B. Lal. *Probeless Illumination Estimation for Outdoor Augmented Reality*, chapter 2, pages 15–30. INTECH, 2010.

[81] R. Mehra, Q. Zhou, J. Long, A. Sheffer, A. Gooch, and N. J. Mitra. Abstraction of man-made shapes. In *SIGGRAPH Asia*, volume 28, 2009.

[82] M. Meilland, C. Barat, and A. Comport. 3D High Dynamic Range dense visual SLAM and its application to real-time object re-lighting. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 143–152, 2013.

[83] R. Mur-Artal and J. D. Tardós. ORB-SLAM: Tracking and Mapping Recognizable Features. In *Workshop on Multi View Geometry in Robotics (MVIGRO) - RSS 2014*, 2014.

[84] R. Mur-Artal and J. D. Tardós. Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015.

[85] R. A. Newcombe. *Dense Visual SLAM*. PhD thesis, Imperial College London, 2012.

[86] R. A. Newcombe and A. J. Davison. Live Dense Reconstruction with a Single Moving Camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[87] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J.

Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[88] R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[89] R. Ng, M. Levoy, M. Bredif, G. Duval, M. Horowitz, and P. Hanrahan. Light Field Photography with a Hand-held Plenoptic Camera. Technical report, Stanford Tech Report CTSR, 2005.

[90] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. In *Proceedings of SIGGRAPH*, 2013.

[91] K. Nishino, Z. Zhang, and K. Ikeuchi. Determining Reflectance Parameters and Illumination Distribution from a Sparse Set of Images for View-dependent Image Synthesis. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2001.

[92] D. Nowrouzezahrai, S. Geiger, K. Mitchell, R. Sumner, W. Jarosz, and M. Gross. Light factorization for mixed-frequency shadows in augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 173–179, 2011.

[93] P. of Vision Raytracer. Pov-ray website. http://www.povray.org/, 2015.

[94] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: A vector representation for smooth-shaded images. In *Proceedings of SIGGRAPH*, volume 27, 2008.

[95] C. Perwass and L. Wietzke. Single Lens 3D-Camera with Extended Depth-of-Field. In *SPIE*, volume 8291, 2012.

[96] B. T. Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[97] P. Piniés, L. M. Paz, and P. Newman. Dense mono reconstruction: Living with the pain of the plain plane. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[98] C. Pirchheim and G. Reitmayr. Homography-based planar mapping and tracking for mobile phones. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[99] M. Pizzoli, C. Forster, and D. Scaramuzza. REMODE: Probabilistic, monocular dense reconstruction in real time. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[100] T. Pock. *Fast Total Variation for Computer Vision*. PhD thesis, Graz University of Technology, 2008.

[101] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche. MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–88, 2013.

[102] R. Ranftl, S. Gehrig, T. Pock, and H. Bischof. Pushing the limits of stereo using variational stereo estimation. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2012.

[103] Raytrix GmbH. Raytrix Website. http://raytrix.de/, 2012.

[104] ReconstructMe. ReconstructMe Website. http://reconstructme.net/.

[105] RepRap. RepRap Website. http://reprap.org/.

[106] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[107] G. Rong and T.-S. Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 109–116. ACM, 2006.

[108] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient

alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE, 2011.

[109] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[110] R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison. Dense Planar SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014.

[111] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In *Pattern Recognition*, pages 31–42. Springer International Publishing, 2014.

[112] D. Scharstein and C. Pal. Learning Conditional Random Fields for Stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[113] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision (IJCV)*, 47:7–42, 2001.

[114] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 195–202, 2003.

[115] Sculpteo. Sculpteo Website. http://www.sculpteo.com/.

[116] Shapeways. Shapeways Website. http://www.shapeways.com/.

[117] Shapify.me. Shapify.me Tutorial. http://www.shapify.me/tutorial.

[118] Stanford Graphics Laboratory. The Stanford Light Field Archive. http://lightfield.stanford.edu, 2014.

[119] Stanford University Computer Graphics Laboratory. Stanford bunny. http://graphics.stanford.edu/data/3Dscanrep/, 2013.

[120] F. Steinbrücker, J. Sturm, and D. Cremers. Real-Time Visual Odometry

from Dense RGB-D Images. In *Workshop on Live Dense Reconstruction from Moving Cameras at ICCV*, 2011.

[121] J. Stuehmer, S. Gumhold, and D. Cremers. Real-Time Dense Geometry from a Handheld Camera. In *Proceedings of the DAGM Symposium on Pattern Recognition*, 2010.

[122] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for RGB-D SLAM evaluation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2012.

[123] V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: a local parameterization approach. 2003.

[124] Y. Uh, Y. Matsushita, and H. Byun. Efficient multiview stereo by random-search and propagation. In *Proceedings of the International Conference on 3D Vision (3DV)*, volume 1, pages 393–400. IEEE, 2014.

[125] L. Valgaerts, A. Bruhn, and J. Weickert. A variational model for the joint recovery of the fundamental matrix and the optical flow. In *Proceedings of the DAGM Symposium on Pattern Recognition*, pages 314–324, 2008.

[126] C.-P. Wang, N. Snavely, and S. Marschner. Estimating dual-scale properties of glossy surfaces from step-edge lighting. In *ACM Transactions on Graphics*, volume 30, page 172. ACM, 2011.

[127] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. *arXiv preprint arXiv:1411.4958 [cs.CV]*, 2014.

[128] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. ElasticFusion: Dense SLAM without a pose graph. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015.

[129] T. Whelan, J. B. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. J. Leonard. Kintinuous: Spatially Extended KinectFusion. In *Workshop on RGB-D: Advanced Reasoning with Depth Cameras, in conjunction with Robotics: Science and Systems*, 2012.

[130] H. Winnemöller, S. C. Olsen, and B. Gooch. Real-time video abstraction. *Proceedings of SIGGRAPH*, 25(3):1221–1226, 2006.

[131] D. Wood, D. Azuma, W. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Stuetzle. Surface Light Fields for 3D Photography. In *Proceedings of SIGGRAPH*, 2000.

[132] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[133] H. Yagou, A. Belyaevy, and D. Weiz. High-boost mesh filtering for 3-d shape enhancement. *Journal of Three Dimensional Images*, 17(1):170–175, 2003.

[134] M. E. Yumer and L. B. Kara. Co-abstraction of shape collections. *ACM Trans. Graph.*, 31(6):166:1–166:11, 2012.

[135] D. Zhou and B. Schölkopf. Regularization on discrete spaces. In *Proceedings of the DAGM Symposium on Pattern Recognition*, 2005.

[136] K. Zhou, M. Gong, X. Huang, and B. Guo. Data-Parallel Octrees for Surface Reconstruction. *IEEE Transactions on Visualization and Computer Graphics (VGC)*, 17(5), 2011.

[137] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-Time KD-Tree Construction on Graphics Hardware. In *SIGGRAPH Asia*, 2008.

[138] Q. Zhou and V. Koltun. Dense scene reconstruction with points of interest. In *Proceedings of SIGGRAPH*, 2013.