RGB-D Scene Flow via Grouping Rigid Motions

by

Francis Li

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2016

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The method presented in this thesis is based on the following conference paper:
F.Li, A.Wong, J.Zelek "Hierarchical Grouping Approach for Fast Approximate RGB-D Scene Flow", Computer and Robot Vision, 2016. Large parts of Chapter 3 and 4 are taken verbatim from this paper and have been expanded upon. F.Li, A.Wong, and J.Zelek conceived and designed the general approach to scene flow estimation. F.Li collected and analyzed data. F.Li, A.Wong, and J.Zelek contributed to the writing and editing of the paper.

Abstract

Robotics and artificial intelligence have seen drastic advancements in technology and algorithms over the last decade. Computer vision algorithms play a crucial role in enabling robots and machines to understand their environment. A fundamental cue in understanding environments is analyzing the motions within the scene, otherwise known as scene flow. Scene flow estimates the 3D velocity of each imaged point captured by a camera. The 3D information of the scene can be acquired by RGB-D cameras, which produce both colour and depth images and have been proven to be useful for solving many computer vision tasks. Scene flow has numerous applications such as motion segmentation, 3D mapping, robotic navigation and obstacle avoidance, gesture recognition, etc. Most state-of-the-art RGB-D scene flow methods are set in a variational framework and formulated as an energy minimization problem. While these methods are able to provide high accuracy, they are computationally expensive and not robust under larger motions in the scene.

The main contributions of this research is a method for efficiently estimating approximate RGB-D scene flow. A new approach to scene flow estimation has been introduced based on matching 3D points from one frame to the next in a hierarchical fashion. One main observation that is used is that most scene motions in everyday life consist of rigid motions. As such, large parts of the scene will follow the same motion. The new method takes advantage of this fact by attempting to group the 3D data in each frame according to like-motions using concepts from spectral clustering. A simple coarse-to-fine voxelization scheme is used to provide fast estimates of motion and accommodate for larger motions. This is a much more tractable approach than existing methods and does not depend on convergence of some defined objective function in an optimization framework. By assuming the scene is composed of rigidly moving parts, non-rigid motions are not accurately estimated and hence the method is an approximate scene flow estimation. Still, quickly determining approximate motions in a scene is tremendously useful for any computer vision tasks that benefit from motion cues.

Evaluation is performed on a custom RGB-D dataset because existing RGB-D scene flow datasets presented to date are mostly based on qualitative evaluation. The dataset consists of real scenes that demonstrates realistic scene flow. Experimental results show that the presented method can provide reliable scene flow estimates at significantly faster runtime speed and can handle larger motions better than current methods.

Acknowledgements

I would like to sincerely thank Alexander Wong and John Zelek for supervising my masters studies. I am grateful for the opportunities they have provided and for supporting my interests.

I would also like to thank the members from both the VIP lab and John Zelek's lab, for they are too kind and have made my time spent here both a joyful and fulfilling experience. I am grateful to the VIP lab directors for encouraging and fostering a collaborative graduate experience.

Last but not least, I want to thank my family for their continual support, patience, and love.

Dedication

This is dedicated to my mom.

Table of Contents

List of Tables

List of Figures

Nomenclature

$\overrightarrow{pp}$	Camera principal point 2D coordinates
f	Camera focal length
x, y	Coordinates to represent spatial location in image space
X, Y, Z	Spatial Cartesian coordinates in world space
$\overrightarrow{p}$	Image point coordinates
$\overrightarrow{P}$	3D world point coordinates
I	2D array of image intensity values
D	2D array of depth image values
Ω	Variable representation of the 2D image domain
t	Variable representation of time
ζ	Motion along the Z axis in an RGB-D frame
u, v	Motion along the x, y directions respectively in image plane
G	Variable representation of a graph
ν	Set of vertices in a graph
ε	Set of edges in a graph
$\mathbf{W}$	$n \times n$ affinity matrix or similarity matrix for a graph with n nodes
w_{ij}	$(i, j)^{th}$ element of W

d	Degree of a vertex in a graph
$\mathbf{\Psi}$	Degree matrix
δ	Disparity value of a pixel in an image for stereo vision
$\mathbf{L}, \mathbf{L}_{sym}, \mathbf{L}_{rw}$	(Respectively) Unnormalized Laplacian matrix, symmetric normalized Laplacian matrix, random walk-based normalized Laplacian matrix of a graph
$\overrightarrow{h}$	Indicator vector used to determine which node belongs to which cluster.
$\mathbf{H}$	Matrix of indicator vectors used for clustering more than than two groups.
k	Number of desired groups used in spectral grouping
λ	Eigenvalue of a matrix
$\mathbf{R}$	3×3 matrix representation of rotation in three space
$\overrightarrow{T}$	3×1 vector representing translation in three space
$\overrightarrow{F}$	2D vector representing optical flow vector
V	3D voxel grid
A_l, B_l, C_l	Voxel grid sizes (height, width, depth) at voxel level l
a, b, c	Voxel grid indices
r	Resolution of a voxel in a voxel grid
l	Level index of a voxel grid
$\overrightarrow{m}$	3×1 motion vector between matching voxels
α	Weighting parameter on colour similarity affinity between nodes in a graph
β	Weighting parameter on motion similarity affinity between nodes in a graph
s_c, s_m	Colour and motion similarity measures, respectively

Chapter 1

Introduction

This section introduces the basic concepts of RGB-D cameras and scene flow and why they are important. The main motivation behind this work is demonstrated by providing some important example applications of scene flow. The shortcomings of existing methods and the main contributions of this research are explained.

1.1 RGB-D Cameras and Scene Flow

Advancements in 3D imaging technology have introduced a wide variety of ways to capture 3D data, such as laser scanners, LiDAR, time-of-flight cameras, and infrared (IR) depth cameras. In particular, there is tremendous interest in RGB-D cameras, which are cameras that are equipped with both a regular camera sensor as well as a depth sensor. Hence, RGB-D cameras produce both an RGB image as well as a depth image, where each pixel in the depth image represents a physical measurement of how far away that point is from the camera in the real world. Examples of commercial RGB-D cameras include the Microsoft Kinect [86], Intel RealSense [37], Asus Xtion [2], etc. (see Figure 1.1 for an example of the Kinect RGB-D camera). These cameras are appealing due to their robust depth measurements, low cost, and real time performance. Nowadays, the technology has gained tremendous interest in industry aiming to deploy RGB-D cameras into smartphone devices e.g., Intel [37], Google [1]. That future smartphones and tablets will eventually be equipped with an RGB-D camera is highly likely.

Having both RGB image and depth measurements have shown to have major advantages in solving computer vision problems, such as segmentation, tracking, recognition,

Figure 1.1: Example of an RGB-D camera. Shown above is the Microsoft Kinect v2 [53]

etc. [11]. For instance, the problem of image segmentation is to distinguish regions of an image according to the task at hand. Suppose we want to segment the scene according to different individual objects. With a regular colour or intensity image, image segmentation algorithms rely on the colour/intensity differences in the image space. With depth information, differences in depth can be a better cue for differentiating between different objects. In general, the additional depth information provides a whole new dimension to the measured information of a scene, providing additional cues and features for machines to better recognize and understand the scene. This work is focused on the use of RGB-D cameras for motion analysis, where the area of research regarding scene motion analysis can benefit significantly from the use of RGB-D cameras. Within the field of 3D vision, estimating scene flow is a fundamental problem that is concerned with estimating the 3D motion field (the 3D velocity vector at each pixel) of a given scene relative to the camera viewpoint. It is the 3D equivalent of optical flow, where optical flow is concerned with estimating the 2D $[x, y]$ motion vector of each pixel from one frame to the next, scene flow estimates the 3D motion vector $[x, y, z]$. With RGB-D cameras, the depth information is directly measured and does not need to be inferred from using regular cameras.

1.2 Motivation: Applications of Scene Flow

Scene flow estimates the 3D motion vector of every imaged point, and knowing the exact movements of each pixel from frame to frame has numerous applications. Some examples include robotic navigation, obstacle avoidance, action recognition, human-computer inter-

facing, object tracking, augmented reality, etc [10, 52, 77]. These applications are described next.

Robots are becoming more and more prevalent in modern times, from gadgets such as the autonomous vacuums and unmanned aerial vehicles (UAVs) to revolutionary technology such as self-driving cars. These robots need to be able to navigate their environment without crashing into obstacles or people. In dynamic environments, this can be a challenging task, but by knowing the motion of every object that the robot observes the robot can take predictive measures to avoid crashing into its surroundings. As well, knowing how the scene moves from the perspective of the camera allows the robot to know the direction that it's moving. In conjunction with the depth maps generated by the RGB-D camera, a 3D map of its surroundings can be created along with its trajectory for navigational purposes. This can be used, for example, to improve simultaneousness localization and mapping (SLAM) applications [10] and perform 3D reconstruction [25].

Scene flow can further be used as input to many other computer vision tasks. For example, motion segmentation becomes a much simpler problem once scene flow is known since one only has to segment the motion field produced by scene flow estimation into distinct motions [34]. As well, object tracking across time also becomes a much simpler problem once the motion of the object over time is known. As a final example, accurate scene flow provides additional information and features to use for tasks such as human action recognition [77], where for example knowing the velocity vector of how the arm is moving can be used to distinguish gestures.

In addition to helping solve traditional computer and robotic vision problems, scene flow has potential uses for future technologies. In human computer interfacing (HCI), a computer that is equipped with an RGB-D camera can perform scene flow analysis to measure the movements of a human user. In conjunction with action recognition methods, this provides another means for users to provide input to machines. In augmented reality applications, the challenge is to seamlessly overlay digital information on top of the imaged scene. In order to do so, the machine must have an extensive understanding of the scene, including 3D structure, object recognition, and scene dynamics. Overlaying digital information on top of a moving object will require accurate tracking and understanding of the object's 3D motion.

There are many more applications of scene flow. As such, scene flow is considered a fundamental computer vision problem that is an active area of research [23]. In terms of practicality, most scene flow applications require real-time or near real-time performance to be useful. Unfortunately, research in the area has thus far focused on accuracy and the run-time performance of most current state-of-the-art scene flow methods are highly

lacking, ranging from minutes to over an hour per frame on a CPU [39, 70, 78]. Clearly, this is nowhere near sufficient to be of practical use and there remains much room for further research. Even if accuracy is compromised for speed, many tasks can still benefit from approximate scene flow, such as obstacle avoidance or gesture recognition. Thus, there is a need for fast, efficient scene flow estimation methods that maintain the accuracy required for application use.

1.3 Thesis Contribution and Outline

This thesis aims to address the issue of computational efficiency in estimating scene flow. The main contribution is a new method of efficiently approximating scene flow based on grouping 3D data of similar motion between frames. The aim is to quickly estimate approximate scene flows from RGB-D sequences while retaining high accuracy on motion estimates. The driving motivation behind the approach is the observation that most everyday scenes consist of rigidly moving parts. If we consider camera movement, vehicular movement, pedestrians, etc., much of the movements of interest can be decomposed to a set of rigidly moving parts. This means we do not necessarily have to find the individual movements of every pixel, but rather find the overall movement of groups of pixels. By discretizing and dividing the scene into groups, we can estimate scene flow much more quickly while maintaining the general motions observed in the scene. Of course, this is not true scene flow if the scene has non ridid motions such as fluids; hence it is an approximate scene flow.

The remainder of the thesis is organized as follows. Chapter 1 introduces additional background knowledge on the problem of scene flow as well as existing methods. Chapter 3 presents the proposed method of estimating scene flow. Experimental data and results are reported in Chapter 4 and finally conclusions and future are discussed in Chapter 5.

Chapter 2

Background

This section introduces the background theory behind RGB-D scene flow estimation as well as existing methods. Section 2.1 begins by reviewing how an RGB-D camera is able to provide full 3D information of the imaged scene using perspective projection and the pinhole camera model. Following that, Section 2.2 discusses RGB-D scene flow methods and its roots from optical flow and multi-view scene flow . Finally, because the method in this work uses concepts from spectral grouping, Section 2.3 introduces the theory of spectral clustering and its usage in motion analysis.

2.1 RGB-D Camera Model

2.1.1 Pinhole Camera Model

Conventional cameras capture the light reflected from the physical world onto a sensor to generate a 2D image. This process can be modelled using the perspective camera model, or pinhole camera model [31]. The pinhole camera model defines the geometric relationship between a 3D point and its corresponding 2D projection onto the image plane. The model is shown in Figure 2.1.

While pinhole cameras are considered antique technology, the pinhole model still holds well for today's standard cameras. For mathematical and visualization convenience, it is convention to move the image plane in Figure 2.1 forward to be in front of the camera center. This simply makes the image appear upright and the projective rays can be seen to

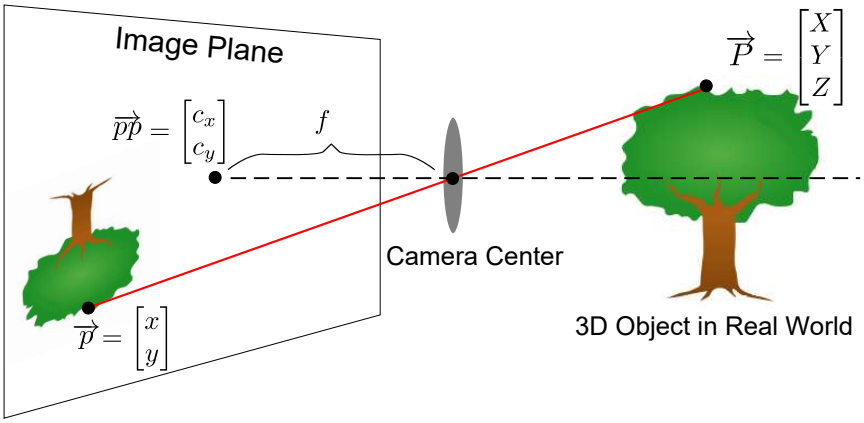

Figure 2.1: How a 3D scene is imaged using the pinhole camera model. $\overrightarrow{pp}$ is the principle point (the projection of the camera center on the image plane) and f is the focal length of the camera. $\overrightarrow{P}$ is the 3D real world coordinate and $\overrightarrow{p}$ is the image pixel coordinate.

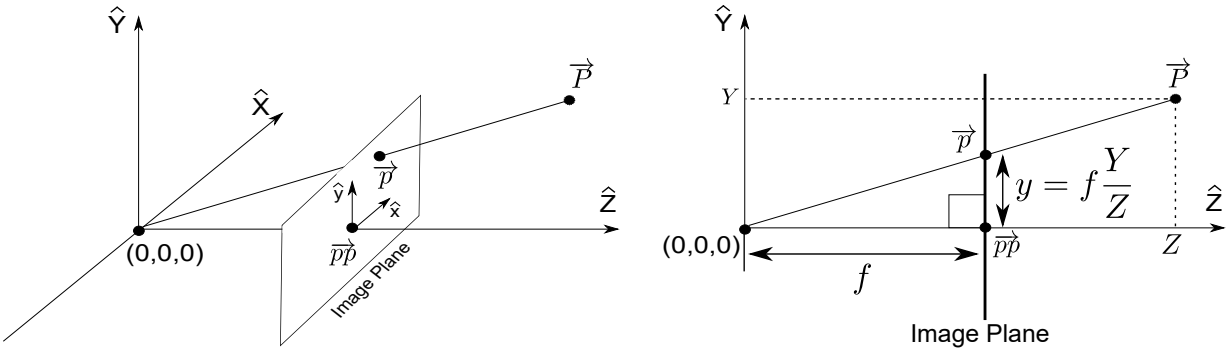

Figure 2.2: Pinhole model showing the geometric relationship between a world 3D point and its 2D image projection via similar triangle principle.

pass through the image plane, making visualization more intuitive. The new representation is shown in Figure 2.2.

The model shown assumes that the origin $(0, 0, 0)$ of the 3D Euclidean world coordinate axis is at the camera center and that the image coordinate origin is at the principle point pp. Then, the x, y image coordinates of $\overrightarrow{P}$ after projecting onto the image plane can be computed using the property of similar triangles:

$$\begin{aligned} \frac{x}{f} = \frac{X}{Z} \rightarrow x = f\frac{X}{Z} \\ \frac{y}{f} = \frac{Y}{Z} \rightarrow y = f\frac{Y}{Z} \end{aligned} \tag{2.1}$$

The focal length f and principle point $pp = [c_x, c_y]$ are often referred to as the camera's intrinsic parameters. Digital images normally have their origin in the top left corner of the image, so the principle points have to be taken into account by simply adding them to Eq. 2.1:

$$\begin{aligned} x = f\frac{X}{Z} + c_x \\ y = f\frac{Y}{Z} + c_y \end{aligned} \tag{2.2}$$

This relationship is normally represented using homogeneous coordinates, which allows Eq. 2.2 to be represented as a matrix multiplication. Homogeneous coordinates are a system of coordinates used in projective geometry, just as Cartesian coordinates are used in Euclidean geometry. In homogeneous coordinates, an N-dimensional coordinate is represented using N+1 numbers. A point in Cartesian coordinates (x, y) is represented in homogeneous coordinates as (x', y', ω) where an additional variable ω is added. The relationship between the Cartesian and homogeneous coordinates is:

$$\begin{aligned} x = x'/\omega \\ y = y'/\omega \end{aligned} \tag{2.3}$$

Using homogeneous coordinates, the 3D to 2D relationship can be represented as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = [\mathbf{K}|0] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.4}$$

where $\mathbf{K}$ is the 3×3 camera matrix (or perspective transformation matrix) containing the camera intrinsic parameters.

Naturally, once a 3D point has been projected onto 2D space the 3D information is lost - the above transformation cannot be inverted. Specifically, since only the x, y image coordinates are known after the transformation, all information related to Z is lost and X, Y cannot be recovered due to their dependence on Z as per Eq. 2.2. Therefore, it is impossible to know the 3D information of the scene given a single image captured by a conventional camera. For example, given a picture of a car, we do not know the scale or size of the car. It could be a real car or it could be a toy model; they may look identical in the image plane. RGB-D cameras are useful because they provide the Z information required to recover the 3D information of the scene.

2.1.2 RGB-D Camera

An RGB-D camera is composed of a camera sensor and a depth sensor to produce a colour (RGB) image as well as a depth (D) image. The colour image is the same as what is captured by a standard camera following the pinhole camera model described in the previous section. The depth image, instead of holding RGB values per pixel, holds the Z value of the 3D scene that was captured. With the depth image, denoted D, and the intrinsic camera parameters f, c_x, c_y, a pixel $D(x, y)$ is projected onto 3D space with coordinates $[X, Y, Z]$ by simply the inverse of Eq. 2.2:

$$X = \frac{(x - c_x)D(x, y)}{f} \qquad Y = \frac{(y - c_y)D(x, y)}{f} \qquad Z = D(x, y) \tag{2.5}$$

If we project every pixel in D onto 3D space, the result is a 3D point cloud that is representative of the real world 3D geometry of the scene. Figure 2.3 shows an example of an RGB image, depth image, and point cloud captured by one frame from an RGB-D camera. The depth image is visualized by normalizing the depth values to be within the range of 8-bit gray-scale image values (0-255).

Depth Sensing Technologies

The depth sensing technology that actually measures the Z values differs between the various available RGB-D cameras. In the past, time-of-flight (ToF) sensors were the most popular means of measuring depth, coupled with a regular camera. These sensors rely on measuring the time it takes for a beam of light to travel to a destination and reflected back. Companies that provide ToF cameras include Heptagon [33], PMD [59], and Basler [6]. The main drawback to ToF cameras is that they require very specialized hardware, such as a

RGB Image Depth Image Point Cloud

Figure 2.3: Example of what an RGB-D camera produces. The left shows the RGB image captured by a conventional camera. The middle shows the depth image, captured by a depth camera, where the depth values have been normalized for visualization. Lighter parts of the depth image represent greater depth (objects farther away) and darker parts represent objects that are closer to the camera. Each point in the point cloud is an (X, Y, Z) coordinate obtained by projecting its corresponding point from the depth image according to Eq. 2.2 & 2.4.

pulsing laser source and calibrated sensor, which are very expensive. This has made them inaccessible to the general public and thus ToF cameras were mainly used in industry and research. More recently, Microsoft introduced the Kinect v2 which uses ToF technology for depth sensing at a much more affordable price of a couple hundred dollars [53].

The last few years have seen another type of RGB-D camera that has become popular. These RGB-D cameras use active stereo vision with infrared (IR) light, which is the technology behind the first Microsoft Kinect [86]. The principle behind active stereo visions is to project some light pattern onto the scene which is then captured by a camera and triangulation is used to determine depth. Typical projection sources include a projector (structured light systems), laser beams, and IR light. Consider a scene viewed by a camera and a laser pointer pointed at the scene from a different angle to the camera, shown in Figure 2.4. We can assume that both the laser pointer and camera have been calibrated beforehand, meaning we know all the intrinsic parameters of the camera as well as the pose (rotation and translation) of the camera and laser pointer. In this case we know the direction vector $\overrightarrow{q_1}$ of the ray emitting from the laser pointer and also the direction of the ray of the point being projected onto the image plane $\overrightarrow{q_2}$. The parameter of interest here is the length of the ray g_2 from the camera center to the point $\overrightarrow{P}$ in the physical world. This can easily be determined by simply finding the intersection of the two rays.

This simple example illustrates the concept of active vision for when we have a single point. When a projection system is used that consists of many points, such as a projector,

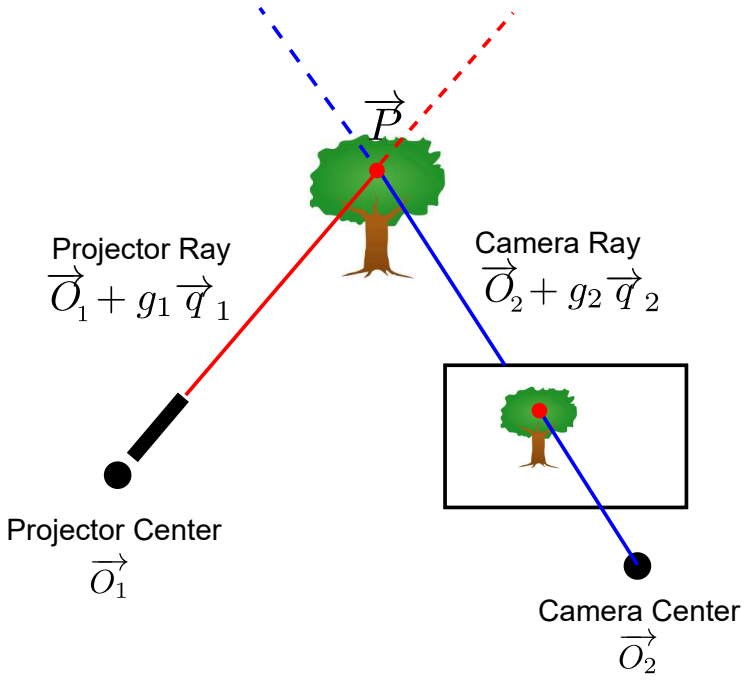

Figure 2.4: Example of active stereo triangulation with a laser point source.

the projection system can also be modelled using the pinhole camera model but instead of rays coming in to the camera, rays are going out. With many points, the camera has to be able to discern which projected point corresponds to which image point i.e., which rays are intersecting. Active vision systems use coded patterns in the projection such that the pattern is spatially discriminative, in contrast to passive vision which relies solely on finding correspondences between two camera images (more information on passive stereo vision in Section 2.2.2). The Microsoft Kinect uses an IR dot pattern where each patch of dots is unique from the rest of the dot patches so the camera can determine the spatial location of each dot based on its neighbouring dot pattern. Using IR is also helpful because it does not contaminate the scene in the visual light spectrum, so the projected pattern appears as if it were invisible to the naked eye. Companies that produce RGB-D cameras using active vision IR patterns include Microsoft [53], Intel [37], and Asus [2].

Thus far, the intrinsic values (focal length and principle points) of the camera were assumed to be known beforehand. While nominal values for these camera parameters are provided by the RGB-D camera manufacturer, more accurate estimates for these parameters can be obtained through camera calibration techniques [85]. Also, the RGB image is assumed to be aligned with the depth image such that the pixels in the RGB image refer

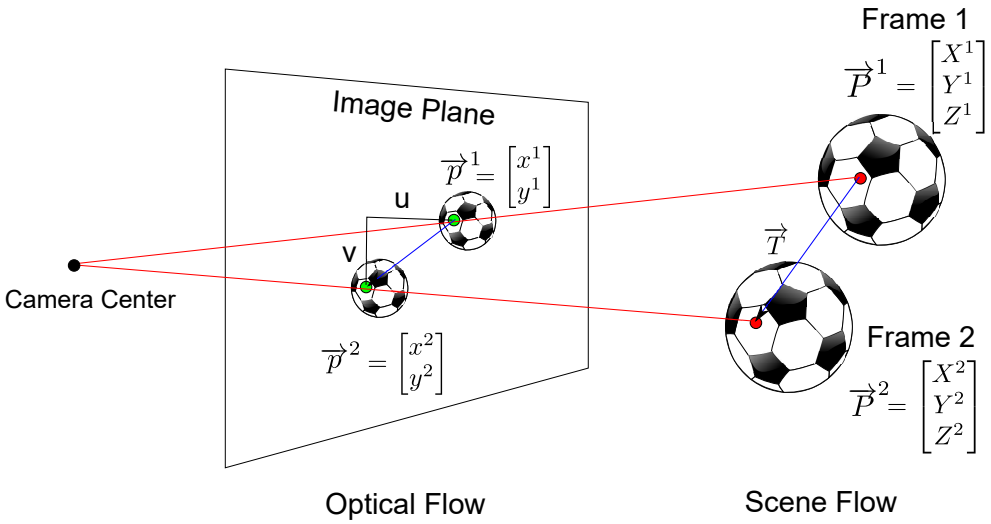

Figure 2.5: Example of a ball's motion captured at two different time steps / frames. Scene flow estimates the 3D velocity vector $\overrightarrow{T}$ of each point from the first frame to the second frame, while optical flow estimates the 2D motion u, v as observed in the image plane.

to the same point in space as the depth image. In practise, the RGB and depth image may be misaligned in which case camera calibration needs to be performed. The next section will discuss the use of RGB-D cameras for motion analysis.

2.2 Scene Flow Estimation

Scene flow estimates the 3D motion of the scene relative to the observer, which in this case is the camera center. A scene sampled at two time steps is captured by two consecutive frames in a video sequence. Given 3D points $\overrightarrow{P}^1 = [X^1, Y^1, Z^1]'$ from frame 1 and $\overrightarrow{P}^2 = [X^2, Y^2, Z^2]'$ from frame 2, scene flow will find the translation from $\overrightarrow{P}^1$ to $\overrightarrow{P}^2$ i.e. $\overrightarrow{P}^2 - \overrightarrow{P}^1$. This concept is shown in Figure 2.5, where a ball's movement is captured across two frames. Historically, due to the lack of depth information, much of motion analysis was performed in the 2D image plane and is termed optical flow. Referring again to Figure 2.5, optical flow finds the $[u, v]$ motion vector from the 2D image points $\overrightarrow{p}^1$ and $\overrightarrow{p}^2$. Scene flow analysis originates from optical flow concepts and hence optical flow will be briefly introduced here.

Frame 1 Frame 2 Optical Flow

Figure 2.6: Optical flow example. Optical flow estimates the motion from frame 1 to frame 2, the downsampeld 2D motion field is shown on the right. Images obtained from Human-assisted motion annotation dataset [48]

2.2.1 Optical Flow

Optical flow aims to find the apparent 2D motion of an imaged scene caused by relative motion between the scene and the camera. It does not take into consideration the 3D motion of objects in the physical world. Despite this, optical flow provides much of the basis behind current scene flow methods because colour images in general are much more distinctive and discriminative than depth information. For instance, a painting is very feature-rich in an RGB image whereas the depth is simply a flat plane.

Given an image sequence, let $I(x, y, t)$ represent the intensity image values over the x, y coordinates and time t and let u and v represent the motion in the x and y directions respectively for each pixel from one frame to the next. Figure 2.6 shows a visual example of a downsampled optical flow field between two frames (the optical flow field contains a motion vector for every pixel). To estimate this 2D motion field, the foundation of optical flow methods rely on the optical flow constraint, which simply states that a given pixel's colour remains constant after applying the motion. For example, the moving silver car in Figure 2.6 appears silver in the first frame and also in the second frame. Hence:

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{2.6}$$

This constraint assumes that temporal intensity changes are only as a result from motion. Eq. 2.6 holds for most image sequences but there are many cases where the constraint breaks down. For example, lighting changes, shadows, and occlusions will make objects appear differently from frame to frame. Nevertheless, this constraint is valid for high sampling rates for small u, v and if $I(x, y, t) \approx I(x + u, y + v, t + 1)$. In this case, we can

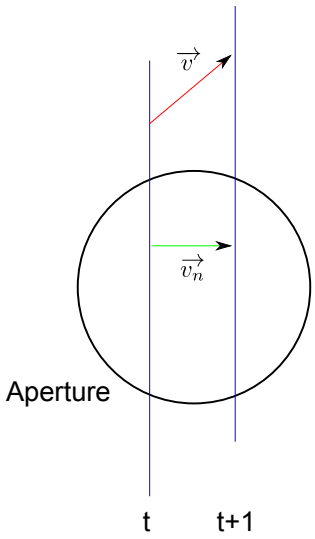

Figure 2.7: Visualization of the aperture problem. The blue line is moving to the right and up, indicated by the red vector $\overrightarrow{v}$. However, when viewed through the aperture of a camera, there it appears to only move to the right, indicated by the blue vector $\overrightarrow{v_n}$. Generally, only the motion that is normal to the structure is recovered.

apply a first order Taylor series expansion about $I(x, y, t)$:

$$I(x + u, y + v, t + 1) = I(x, y, t) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} \tag{2.7}$$

Subbing in Eq. 2.6:

$$\begin{aligned} 0 &= \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} \\ 0 &= I_x u + I_y v + I_t \end{aligned} \tag{2.8}$$

where I_x, I_y, I_t are the partial derivatives of the image $I(x, y, t)$ with respect to x, y, t respectively. The above constraint is one relation in the two variables u, v and is under-constrained, thus additional prior information is required to obtain a unique solution. This is referred to as the aperture problem in optical flow estimation, and is a result of lacking local structural information in the image to uniquely find the 2D motion. For example, an edge moving to the right and up when viewed through the aperture of a camera may only look like the edge is moving in the right direction (see Figure 2.7). Many optical flow methods address this issue by using some form of regularization on the motion field, adding additional constraints to the motions such as a smoothness assumption [22, 78]. Two

seminal works in optical field are often cited with regards to this problem. One is by Lucas and Kanade [50] which is considered a local regularized approach. They assume a constant optical flow field around a given pixel's neighbourhood. The optical flow constraint is computed using all pixels in the neighbourhood to get an overconstrained system where the solution is to minimize the sum of least squared deviations from the optical flow constraint i.e., for a neighbourhood patch N, where N consists of all pixels within a square window of a specified width around a given pixel:

$$[u, v] = \arg\min_{u,v} \sum_{x,y \in N} (I_x(x, y, t)u + I_y(x, y, t)v + I_t(x, y, t))^2 \tag{2.9}$$

The other approach is by Horn and Schunck [35] which is considered a global approach and is set in a variational framework. They use an energy objective function which contains a regularization term that penalizes large disparities in the flow field to encourage spatially smooth motions. The energy over the image domain Ω is defined as

$$E = \int_{\Omega} (|I_x u + I_y v + I_t|^2 + \mu(|\nabla u|^2 + |\nabla v|^2))d\Omega \tag{2.10}$$

where μ is a smoothness weighting parameter and $\nabla = \partial^2/\partial x^2 + \partial^2/\partial y^2$ is the Laplacian operator, used to apply a cost to large variations in the u and v motion fields. While the Lucas and Kanade method and Horn and Schunck method are relatively old, originating decades ago, they still hold up today in terms of optical flow accuracy [22]. The variational method in particular has spawned many different optical flow methods that use a similar framework. Optical flow methods that are set in a variational framework utilize a smoothness assumption on the motion field and optimize a global energy cost over the image domain Ω.

$$E_{global} = \int_{\Omega} E_{data}(I_1, I_2, u, v) + E_{reg}(u, v) \tag{2.11}$$

The data energy E_{data} will generally be some variant of the optical flow constraint, and the regularization term E_{reg} will penalize large variations in the flow field. Many other robust data and regularizer terms have been proposed [13, 18, 68, 82]. Optimizing this type of energy functional is the basic framework for many optical flow methods, where they differ in choice of energy terms and method of optimization [14, 17, 32, 72, 79, 82, 87]. This approach is particularly applicable in the context of scene flow because scene flow methods are set in a similar framework (see Sections 2.2.2,2.2.3 below).

In solving the energy functional, many approaches have been proposed in literature by the different methods. Examples include solving the Euler-Lagrange equation [7, 13, 35],

primal-dual frameworks [16], graph cuts [44], etc. Computational efficiency was and still is an issue with methods using the variational framework. The best performing optical flow methods remain computationally expensive [22].

Another challenge with optical flow methods is handling large displacements. This is due to the linear approximation in Eq. 2.8 that holds only in the near vicinity of a pixel. The general approach is to use a coarse-to-fine approach to deal with these cases [8, 15].

Optical flow has a long history of active research and many other models have been proposed for estimating motion vectors. It is outside the scope of this work to review them all. Rather, the variational approach and related methods were introduced that pertained to the scene flow problem of interest here since many scene flow methods use similar optimization frameworks. While optical flow methods have seen impressive results in recent years, without 3D information there will always be cases where 2D optical flow is insufficient to discern the correct motion. The barber pole illusion is the classic example of this case, where the red and white stripes a barber pole appear to be moving in the up/down direction when reality it is spinning around an axis. Ultimately, a combination of RGB and depth can provide much more information than either one can independently.

2.2.2 Multi-view Scene Flow

Traditionally, scene flow has been studied in multi-view vision using stereo or multiple cameras. In multi-view scene flow, depth and motion are jointly estimated through concepts from multi-view geometry [31]. Vedula *et al.* [73] first defined the term "scene flow" as the estimation of 3D motion fields through imaging and since then many methods have been introduced to estimate depth and motion.

The basis behind optical methods of measuring depth is through the principles of triangulation and geometry [31]. A stereoscopic camera is a passive vision system, meaning it does not alter the scene in any way as done in active vision systems. Rather, stereo vision infers the 3D geometry of a scene by using two cameras. Typically, the two cameras are referred to as the left camera that produces the left image I^L and right camera for the right image I^R. Normally, I^L and ^{R} are rectified such that the row pixels of one image correspond to the rows in the other (i.e., corresponding points have the same y value). This is shown in Figure 2.8. Given a stereo image pair I^L, I^R, a pixel in the left image $I^L(x, y)$ corresponds to a pixel in the right image $I^R(x', y')$ through a displacement δ along the x direction:

$$\begin{aligned} x &= x' + \delta \\ y &= y' \end{aligned} \tag{2.12}$$

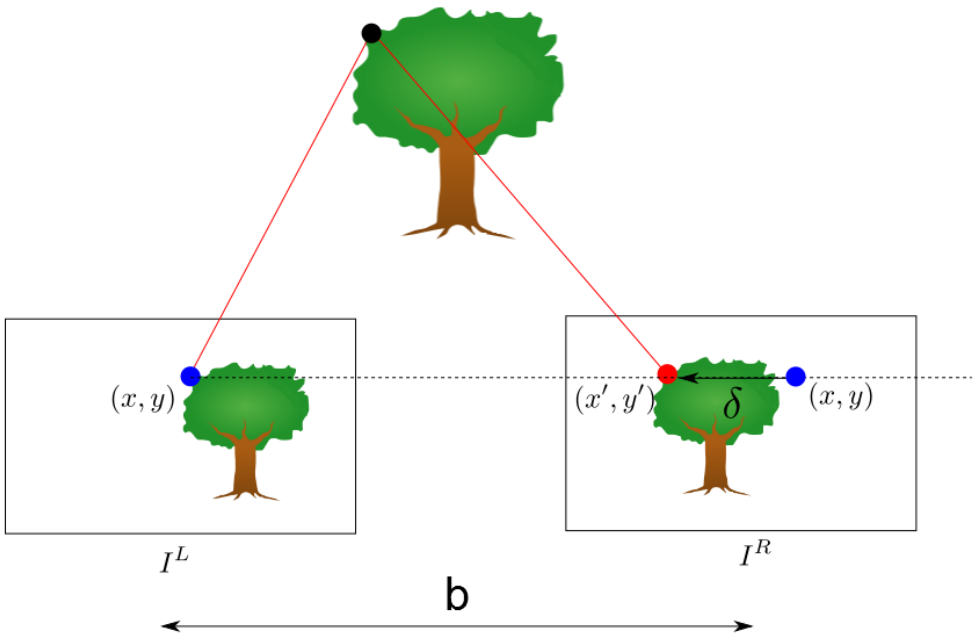

Figure 2.8: Stereo cameras imaging a scene.

The geometric relationship between a 3D world point coordinates $[X, Y, Z]$ and its 2D image projection $[x, y]$ onto the left image can be computed through the following relationship [31]:

$$\begin{bmatrix} x \\ y \\ \delta \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} Xf \\ -Yf \\ bf \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \\ 0 \end{bmatrix} \tag{2.13}$$

where b is the baseline between the two cameras. Hence, if the camera parameters are known(via calibration), the 3D points can be recovered by finding the displacements δ. The fundamental problem that stereo vision algorithms aim to solve is estimating the displacement δ for each pixel, otherwise known as the correspondence problem since the task involves finding corresponding points between the left and right images. To this end, some correlation or similarity metric is used to find corresponding points, such as the sum of squared differences (SSD) [41] or sum of absolute differences (SAD) [30]. For motion analysis, the addition of another camera allows inference of scene flow and introduces additional constraints [78]. The optical flow constraint can be applied to both the left and right cameras independently. Starting with the left camera:

$$I^L(x, y, t) = I^L(x + u, y + v, t + 1) \tag{2.14}$$

For the right camera, we can represent the pixel x coordinates in terms of the left image coordinates, with the unknown displacement δ. The optical flow motion will introduce differences only along the x direction since the y coordinates will be the same from Eq. 2.12. The change in motion along x is denoted by δ':

$$I^R(x+\delta, y, t) = I^R(x+\delta+u+\delta', y+v, t+1) \tag{2.15}$$

There is an additional constraint which states that the left image after applying the flow field should be consistent with the right image after applying the flow field:

$$I^L(x+u, y+v, t+1) = I^R(x+\delta+u+\delta', y+v, t+1) \tag{2.16}$$

These three constraints comprise the stereo motion constraints and are used in many stereo scene flow methods again in a variational or energy minimization framework [38, 58, 84]. Wedel *et al.* [78] formulated the problem in a variational framework where they decoupled the estimation of depth and motion. They employed an energy function composed of a data term and smoothness term similar to Eq.2.11, consisting of the optical flow brightness consistency constraint and the disparity constraint between left and right cameras. More recent state-of-the-art approaches in stereoscopic scene flow model a scene as a set of moving slanted planes to provide a constraint on local rigidity [80, 75, 52]. Vogel *et al.* [75] modelled a scene as a collection of rigidly moving planes, allowing them to simplify the problem to that of estimating the motion of a discrete set of segments which they infer using a discrete conditional random field (CRF). Similarly, Menze and Geiger [52] also segmented the scene into moving rigid planes but further modelled the scene as rigidly moving objects and a background. These methods have shown top performance on the KITTI benchmark dataset [23, 52].

Inherently, stereoscopic scene flow require accurate estimation of the depth of the scene (the disparity δ), which can be challenging in regions with few texture features. As such, stereoscopic scene flow methods generally rely on some form of global energy minimization, which is computationally expensive.

2.2.3 RGB-D Scene Flow

With RGB-D cameras, the problem of unreliable depth inference is alleviated since there are now depth measurements directly available. Compared to optical flow and multi-view scene flow, the field of RGB-D scene flow is relatively not well explored. Just as there are constraints on brightness and disparity for optical flow and multi-view scene flow, there is

an analogous constraint when given depth information called the range flow constraint [69] on the depth map D:

$$D_x u + D_y v + \zeta + D_t = 0 \tag{2.17}$$

where ζ is the change in depth and represents motion along the Z direction. Some early work in RGB-D scene flow were set in a variational framework that extend optical flow methods to include the depth constraint in Eq.2.17 [26, 46, 34, 40]. Similarly to optical flow, regularization is just as crucial in RGB-D scene flow. This generally meant adding an additional depth consistency cost term to Eq. 2.11 and regularizing the 3D motion field $[u, v, \zeta]$:

$$E_{global} = \int_{\Omega} E_{colour}(I_1, I_2, u, v) + E_{depth}(D_1, D_2, u, v, \zeta) + E_{reg}(u, v, \zeta) \tag{2.18}$$

This global approach has generally been favoured over local approaches in scene flow estimation. This is probably due to the fact that local structures in 3D are not distinctive and have very low discriminative power. Consider again an example of a painting where the 3D structure of one local patch is nearly identical to another patch since both are simply flat surfaces. Hence, global approaches relying on regularization have become more widely used. Herbst *et al.* [34] minimized a global energy function consisting of a data term for colour and depth consistency with an anisotropic regularization term to provide smoothness while maintaining boundaries. To work around the limitations of variational methods in estimating large motions, Letouzey *et al.* [46] used sparse feature matching (SIFT) to get prior knowledge of the motions before optimization. Quiroga *et al.* [61] used a local rigidity approach within a global weighted total variation optimization framework, where they used twist motions to model the motion field. Jaimez *et al.* [39] formulated scene flow with a spatial regularization on the 3D surface and minimized their energy function in a primal-dual framework to achieve real-time RGB-D scene flow on a GPU. Similarly, Ferstl *et al.* [20] also used the primal-dual principal on a GPU to minimize their energy function that used a ternary census transform and closest point matching as their data term. Other than the work by Jaimez *et al.* [39] and Ferstl *et al.* [20], the above methods were not designed to be computationally efficient and focused more on scene flow accuracy.

There is some work in RGB-D scene flow that step outside the variational paradigm. Hadfield and Bowden [29, 28] used a particle filter approach, hypothesizing over the possible motion vectors of the 3D data. This approach has a large number of particle candidates for each pixel and thus is computationally expensive. Hornacek *et al.* [36] proposed a patch based matching scheme using spherical patches of 3D data in combination with a variant of the patchmatch algorithm [4] to obtain a dense 6D motion field to account for

rotation and translation. There are also some methods that make use of edge information in the depth map to match larger flow vectors and handle larger motion [3, 64]. The heavy dependence on edges make the applicability of these methods more scene-specific. Finally, Sun *et al.* [70] used a layered model based on the depth of the scene to reason about occlusion and constrain the motion. They were able to achieve very high accuracy on the Middlebury optical flow dataset [65]. These methods are also not designed for fast scene flow estimation, where they are either inherently computationally expensive or use a final refinement step to smooth the motion field using a regularized objective function. Compared to existing approaches, the method presented in this work is computationally simpler and does not require explicit regularization on the motion field.

2.3 Motion Analysis Through Groupings

Part of the method presented in this work involves clustering or grouping the scene according to similar motions. As such, methods in motion analysis that use grouping techniques are reviewed here.

2.3.1 Motion Segmentation

Related to scene flow, the task of motion detection and segmentation in computer vision is of significant importance. Motion segmentation is essential for applications in robotics, video surveillance, action recognition, object segmentation, etc. [83]. For instance, a surveillance camera system would be interested in finding parts of the video where motion occurs to signify a person or car moving. As well, in sports analytics, segmentating and identifying a moving ball or puck can help analyse the actions or plays being made. Amongst the many approaches to motion detection and segmentation, a subset of methods use grouping or clustering techniques [57, 60]. These techniques generally track a sparse set of pixels across multiple frames and then cluster the scene into individual motions based on the pixel trajectories. An example is shown in Figure 2.9 for a scene with moving cars and static background. The three moving cars and static background comprise a total of four independent motions. Here, the circles represent tracked points across the two frames and are colour coded according to which cluster they belong to. Finding pixels to track can be done by finding image features using methods such as scale-invariance feature tracking (SIFT) [49].

At its core, clustering for motion analysis in computer vision revolves around the problem of subspace clustering. That is, although an image sequence provides millions of

Figure 2.9: Example of motion segmentation by tracking point trajectories (shown by the circles). The different motion clusters labelled by colour, with green circles representing the background. Images obtained from KITTI autonomous vehicle dataset [23].

available pixel data, the scene itself and the motions associated with it are represented using far fewer parameters and dimensions. This leads to using simpler, lower dimension representations of the data of interest. Conventional techniques for dimensionality reduction include principle component analysis (PCA). In motion analysis, such as motion segmentation, the interest is in the representation of pixel motions. This is normally done by representing pixel trajectories in some subspace, and clustering within the subspaces to find individual motions. We can let $\tau = [x^1, y^1, x^2, y^2, ..., x^F, y^F]$ be the trajectory of a tracked feature point over F frames. With P number of tracked points, the *measurement matrix* can be formed by stacking the trajectories together:

$$\begin{bmatrix} x_1^1 & \cdots & x_P^1 \\ y_1^1 & \cdots & y_P^1 \\ \vdots & \ddots & \vdots \\ x_1^F & \cdots & x_P^F \\ y_1^F & \cdots & y_P^F \end{bmatrix} = \begin{bmatrix} \Lambda^1 \\ \vdots \\ \Lambda^F \end{bmatrix} \begin{bmatrix} X_1 & \cdots X_P \\ Y_1 & \cdots Y_P \\ Z_1 & \cdots Z_P \\ 1 & \cdots 1 \end{bmatrix} \tag{2.19}$$

where Λ is some camera motion model, which can be the perspective camera model from Eq.2.4 or more commonly the affine camera model [31, 42, 62, 71, 74] for its linearity and simplicity. We want to segment the measurement matrix on the left according to its individual rigid motions, i.e., cluster the trajectories according to their motions. At its foundation, this is a linear subspace segmentation / clustering problem, where we want to find k independent linear subspaces of $\mathbb{R}^{2F}$. Various ways of performing this clustering have been proposed in literature, including generalized PCA [74], matrix factorizations [12],iterative methods [81],and spectral clustering methods [19]. Spectral clustering in particular has gained popularity for its efficient computations, ease of implementation, and clustering performance [76].

2.3.2 Spectral Clustering for Motion Analysis

Spectral clustering is a class of methods and algorithms for clustering data in a graph structure based on vertex affinities defined by the edge weights. Spectral clustering algorithms are derived from spectral graph theory, which studies the eigenvalues and eigenvectors of graph matrices [21]. This section will review the theory of spectral clustering used in this work.

Graph Preliminaries and Definitions

A graph $G = (\nu,\varepsilon)$ is defined by its set of vertices ν and edges ε. Here G is a weighted undirected graph, where the edge weights represent some sort of similarity measure between connected vertices. Intuitively, spectral clustering aims to partition the graph into subgraphs such that the vertices within each subgraph have high similarity with each other i.e., have high edge weights (see Figure 2.10, where high edge weights exist within the blue and red clusters which are separated by low edge weights). Letting $n = |\nu|$ represent the total number of vertices in G, the edge weights can be represented as a similarity matrix (or adjacency matrix) $\mathbf{W}$ of size $n \times n$, where each element w_{ij} of $\mathbf{W}$ holds the edge weight connecting vertices ν_i and ν_j.

For a given vertex with m number of edges, the degree d of a vertex is defined as the sum of incident edge weights:

$$d_i = \sum_{j=1}^{m} w_{ij} \tag{2.20}$$

The degree matrix, $\mathbf{\Psi}$, is defined to be a diagonal matrix with the diagonal values comprised of d_i.

For a given subgraph $S \subset \nu$, its 'size 'can be measured as the volume of the subgraph *vol(S)*, which is defined as the sum of the degrees of all vertices in S:

$$vol(S) = \sum_{i \in S} d_i \tag{2.21}$$

A cut, which partitions the graph into two disjoint subgraphs, can be quantized as the sum of the edge weights that separate the two partitions. For two subgraphs $S_1, S_2 \subset \nu$, define $W(S_1, S_2)$ as the sum of edge weights connecting vertices from S_1 to S_2:

$$W(S_1, S_2) = \sum_{i \in S_1, j \in S_2} w_{ij} \tag{2.22}$$

For k subgraphs $S_1 ... S_k$, a cut is defined as

$$cut(S_1, ..., S_k) = \sum_{i=1}^{k} W(S_i, \overline{S_i}) \tag{2.23}$$

where $\overline{S_i}$ is the complement of S_i.

Spectral Clustering Algorithm

As mentioned before, the goal of spectral clustering on a graph is to cluster the graph into subgraphs such that the edge weights within each subgraph are high, signifying high similarity. Put another way, clustering the graph vertices means finding a desired *mincut*, which is a cut to partition the graph into subgraphs such that the connecting edge weights between subgraphs are minimized. In practise, finding the mincut is relatively simple, but often leads to solutions where a single vertex is isolated from the rest of the graph as a subgraph [76]. Hence, at the same time, the subgraphs should be of relatively equal size relative to each other i.e., we don't want a partition that is insignificantly smaller than the rest. To this end, the popular paper by Shi and Malik [67] proposed a normalized cut (Ncut) defined as:

$$Ncut(S_1, ..., S_k) = \sum_{i=1}^{k} \frac{cut(S_i, \overline{S_i})}{vol(S_i)} \tag{2.24}$$

Minimizing the normalized cut in Eq. 2.24 takes into consideration the volume (or size) of each graph partition S_i in the denominator, thus discouraging very small volumes. An example of a graph structure and Ncut is shown in Figure 2.10, with nodes shown as circles and edges shown as lines with the corresponding edge weights beside each edge. The two subgraphs, S and its complement $\overline{S}$ are colour coded blue and red.

An exact solution to finding the minimum Ncut is NP-hard [76]. Spectral clustering allows for an approximate solution in polynomial time. The core of spectral clustering algorithms involves the eigenvalue decomposition of the Laplacian matrix [21, 76]. The unnormalized Laplacian matrix $\mathbf{L}$ is defined as [76]:

$$\mathbf{L} = \mathbf{\Psi} - \mathbf{W} \tag{2.25}$$

where $\mathbf{\Psi}$ is the degree matrix and $\mathbf{W}$ is the adjacency matrix defined above. Note that in literature there are various proposed Laplacian matrices [21] to accomplish different goals.

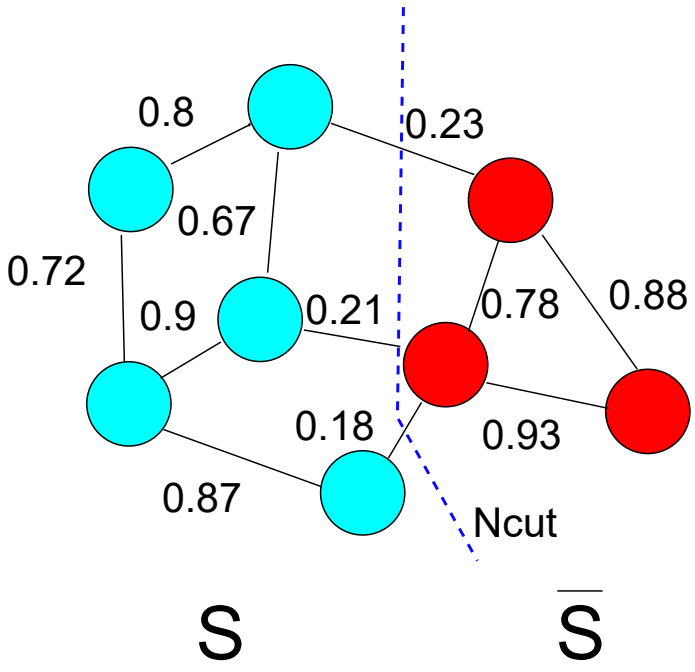

Figure 2.10: Toy example of an Ncut on a sample graph. The circles represent the vertices ν, and the lines and numbers represent the edges ε and edge weights. The graph is partitioned into two subgraphs, S (blue) and $\overline{S}$(red), where it is seen that the edge separating the two subgraphs have smaller weight than the edge weights within each subgraph. Ncut aims to minimize the edge weights connecting the subgraphs and maximize the edge weights contained within each subgraph.

A commonly used Laplacian matrix is the normalized symmetric Laplacian, $\mathbf{L}_{sym}$, introduced in the seminal work by Shi and Malik [67] and has improved clustering performance over the unnormalized Laplacian matrix [76]:

$$\begin{aligned} \mathbf{L}_{sym} &= \mathbf{\Psi}^{-\frac{1}{2}}\mathbf{L}\mathbf{\Psi}^{-\frac{1}{2}} \\ &= \mathbb{I} - \mathbf{\Psi}^{-\frac{1}{2}}\mathbf{W}\mathbf{\Psi}^{-\frac{1}{2}} \end{aligned} \tag{2.26}$$

where $\mathbb{I}$ is the identity matrix. Closely related to $\mathbf{L}_{sym}$ is the normalized Laplacian referred to as $\mathbf{L}_{rw}$ due to its relation to random walks in literature [76]:

$$\begin{aligned} L_{rw} &= \mathbf{\Psi}^{-1}\mathbf{L} \\ &= \mathbb{I} - \mathbf{\Psi}^{-1}\mathbf{W} \end{aligned} \tag{2.27}$$

A few important properties of $\mathbf{L}_{sym}$ and $\mathbf{L}_{rw}$ to this work are listed below.

1. For every vector $\overrightarrow{h} \in \mathbb{R}^n$:

$$\overrightarrow{h}'\mathbf{L}_{sym}\overrightarrow{h} = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}\left(\frac{h_i}{\sqrt{d_i}} - \frac{h_j}{\sqrt{d_j}}\right)^2 \tag{2.28}$$

Proof

$$\begin{aligned}
\overrightarrow{h}'\mathbf{L}_{sym}\overrightarrow{h} &= \overrightarrow{h}'(\mathbb{I} - \mathbf{\Psi}^{-\frac{1}{2}}\mathbf{W}\mathbf{\Psi}^{-\frac{1}{2}})\overrightarrow{h} \\
&= \overrightarrow{h}'\mathbb{I}\overrightarrow{h} - \overrightarrow{h}'\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{W}\mathbf{\Psi}^{-\frac{1}{2}}\overrightarrow{h} \\
&= \sum_{i=1}^{n} h_i^2 - \sum_{ij}^{n} w_{ij}\frac{h_i}{\sqrt{d_i}}\frac{h_j}{\sqrt{d_j}} \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} h_i^2 + \sum_{j=1}^{n} h_j^2 + 2\sum_{ij}^{n} w_{ij}\frac{h_i}{\sqrt{d_i}}\frac{h_j}{\sqrt{d_j}}\right) \\
&= \frac{1}{2}\sum_{i,j=1}^{n} wij\left(\frac{h_i}{\sqrt{d_i}} - \frac{h_j}{\sqrt{d_j}}\right)^2
\end{aligned} \tag{2.29}$$

2. λ is an eigenvalue of $\mathbf{L}_{rw}$ with eigenvector $\overrightarrow{e}$ if and only if λ is an eigenvalue of $\mathbf{L}_{sym}$ with eigenvector $\overrightarrow{e}' = \mathbf{\Psi}^{\frac{1}{2}}\overrightarrow{e}$.

Proof

$$\begin{aligned}
\mathbf{L}_{sym}\overrightarrow{e}' &= \lambda\overrightarrow{e}' \quad \text{multiply both sides by } \mathbf{\Psi}^{-\frac{1}{2}} \\
\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{\Psi}^{-\frac{1}{2}}L\mathbf{\Psi}^{-\frac{1}{2}}\overrightarrow{e}' &= \lambda\mathbf{\Psi}^{-\frac{1}{2}}\overrightarrow{e}' \quad \text{sub. } \overrightarrow{e} = \mathbf{\Psi}^{-\frac{1}{2}}\overrightarrow{e}' \\
\mathbf{\Psi}^{-1}\mathbf{L}\overrightarrow{e} &= \lambda\overrightarrow{e} \\
\mathbf{L}_{rw}\overrightarrow{e} &= \lambda\overrightarrow{e}
\end{aligned} \tag{2.30}$$

3. λ is an eigenvalue of $\mathbf{L}_{rw}$ with eigenvector $\overrightarrow{e}$ if and only if λ and $\overrightarrow{e}$ solve the general eigenproblem $\mathbf{L}\overrightarrow{e} = \lambda\mathbf{\Psi}\overrightarrow{e}$.

Proof

$$\begin{aligned}
\mathbf{L}_{rw}\overrightarrow{e} &= \lambda\overrightarrow{e} \quad \text{multiple both sides by } \mathbf{\Psi} \\
\mathbf{\Psi}\mathbf{\Psi}^{-1}\mathbf{L}\overrightarrow{e} &= \lambda\mathbf{\Psi}\overrightarrow{e} \\
\mathbf{L}\overrightarrow{e} &= \lambda\mathbf{\Psi}\overrightarrow{e}
\end{aligned} \tag{2.31}$$

Suppose the goal is to find the minimum Ncut for k number of subgraphs $S_1 ... S_k$. First define k indicator vectors $\overrightarrow{h}_s \in \mathbb{R}^n, s = 1...k$, with each indicator vector being of the form:

$$h_{si} = \begin{cases} \frac{1}{\sqrt{vol(S)}} & \text{if } \nu_i \in S_s \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1...n; s = 1...k) \tag{2.32}$$

These indicator vectors are the desired variables to be found as they specify which vertex belongs to which subgraph. The specific form of $\overrightarrow{h}$ in Eq. 2.32 is chosen such that:

$$\begin{aligned}
\overrightarrow{h}'_s \mathbf{L} \overrightarrow{h}_s &= \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(h_{si} - h_{sj})^2 \qquad \text{(from Property 1, Eq. 2.28)} \\
&= \frac{1}{2}\sum_{i\in S_s, j\in \overline{S_s}} w_{ij}\left(\frac{1}{\sqrt{vol(S_s)}}\right)^2 + \frac{1}{2}\sum_{i\in \overline{S_s}, j\in S_s} w_{ij}\left(-\frac{1}{\sqrt{vol(S_s)}}\right)^2 \\
&= \sum_{i\in S_s, j\in \overline{S_s}} \frac{w_{ij}}{vol(S_s)} + \sum_{i\in \overline{S_s}, j\in S_s} \frac{w_{ij}}{vol(S_s)} \\
&= \frac{cut(S_i, \overline{S_s})}{vol(S_s)}
\end{aligned} \tag{2.33}$$

From Eq. 2.33 and referring to the definition of Ncut in Eq. 2.24 it is seen that $\text{Ncut}(S_1 ... S_k) = \sum_s^k \overrightarrow{h}'_s \mathbf{L} \overrightarrow{h}_s$. Therefore, Ncut minimization is equivalent to minimizing $\sum_s^k \overrightarrow{h}'_s \mathbf{L} \overrightarrow{h}_s$.

The indicator vectors can be concatenated to form the columns of a matrix $H \in \mathbb{R}^{n\times k}, \mathbf{H} = [\overrightarrow{h}_1, \overrightarrow{h}_2, ..., \overrightarrow{h}_k]$. It can be shown that

$$\mathbf{H}'\mathbf{\Psi}\mathbf{H} = \mathbb{I} \tag{2.34}$$

This can be verified by directly multiplying the terms and noticing the diagonal terms are $(\sum_{i,\nu_i\in S_i} d_i)/vol(S_i) = vol(S_i)/vol(S_i) = 1$. Similarly, it can be checked that:

$$\overrightarrow{h}'_s \mathbf{L} \overrightarrow{h}_s = (\mathbf{H}'\mathbf{L}\mathbf{H})_{ss} \tag{2.35}$$

Hence, $\sum_s^k \overrightarrow{h}'_s \mathbf{L} \overrightarrow{h}_s = \sum_s^k (\mathbf{H}'\mathbf{L}\mathbf{H})_{ss} = Tr(\mathbf{H}'\mathbf{L}\mathbf{H})$ where Tr indicates the trace of a matrix and the Ncut minimization problem can be restated as solving the following equation:

$$\arg\min_{S_1 ... S_k} Tr(\mathbf{H}'\mathbf{L}\mathbf{H}) \text{ subject to } \mathbf{H}'\mathbf{\Psi}\mathbf{H} = \mathbb{I} \tag{2.36}$$

This discrete optimization problem, in which the desired indicator vectors in $\mathbf{H}$ can take on finite discrete set of values from Eq. 2.32, is still NP hard [76]. We can relax this optimization problem by allowing the indicator vectors to take on real values. Substituting $\mathbf{M} = \mathbf{\Psi}^{\frac{1}{2}}\mathbf{H}$ (reason explained shortly), Eq. 2.36 is rewritten as:

$$\arg\min_{\mathbf{M}\in\mathbb{R}^{n\times k}} Tr(\mathbf{M}'\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{L}\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{M}) \text{ subject to } \mathbf{M}'\mathbf{M} = \mathbb{I} \tag{2.37}$$

The reason for the substitution of $\mathbf{M}$ is that now Eq. 2.37 can be solved by directly applying the following theorem derived from the Courant Fisher characterization [43]:

Given a symmetric matrix $\boldsymbol{A}$ *of size* $n \times n$ *and an arbitrary matrix* $\boldsymbol{V} \in \mathbb{R}^{n \times k}$*, then the trace of* $\boldsymbol{V'AV}$ *is minimized when* $\boldsymbol{V}$ *is an orthogonal basis of the eigenspace associated with the smallest eigenvalues. If* $\overrightarrow{e}_1, ..., \overrightarrow{e}_k$ *are eigenvectors associated with the increasing eigenvalues* $\lambda_1, ..., \lambda_k$ *and* $\boldsymbol{E} = [\overrightarrow{e}_1, ..., \overrightarrow{e}_k]$ *then:*

$$\min Tr(\mathbf{V}'\mathbf{A}\mathbf{V}) = Tr(\mathbf{E}'\mathbf{A}\mathbf{E}) = \lambda_1 + ... + \lambda_k, \qquad \mathbf{V} \in \mathbb{R}^{n \times k}, \mathbf{V}'\mathbf{V} = \mathbb{I} \tag{2.38}$$

In other words, since $\mathbf{\Psi}^{-\frac{1}{2}}\mathbf{L}\mathbf{\Psi}^{-\frac{1}{2}} = \mathbf{L}_{sym}$, the solution of $\mathbf{M}$ to Eq. 2.37 is the matrix containing the first k eigenvectors of $\mathbf{L}_{sym}$ as columns. For computational convenience, it can also be shown that the solution to Eq. 2.37 is the first k eigenvectors of $\mathbf{L}\overrightarrow{e} = \lambda\mathbf{\Psi}\overrightarrow{e}$ by applying Property 2 and 3 of $\mathbf{L}_{sym}$ and $\mathbf{L}_{rw}$ above.

Once the real valued vectors of $\mathbf{M}$ have been solved for, they must still be converted back to the discrete indicator vectors in order to group the vertices of the graph. This can be done by treating the rows of $\mathbf{M}$ as data points and performing k-means clustering on the rows of $\mathbf{M}$ since vertices belonging in same group will have similar indicator vectors.

The entire spectral clustering algorithm is summarized in Algorithm 1, and popular spectral clustering algorithms generally follow this algorithm [76, 67, 54].

Algorithm 1 Spectral Clustering Algorithm

For input Similarity matrix W and k clusters:

1: Compute Laplacian $\mathbf{L} = \mathbf{\Psi} - \mathbf{W}$
2: Compute first k eigenvectors $\overrightarrow{e}_1, ... \overrightarrow{e}_k$ to the eigenproblem $\mathbf{L}\overrightarrow{e} = \lambda\mathbf{\Psi}\overrightarrow{e}$
3: Construct matrix $\mathbf{E}$ using $\overrightarrow{e}$ vectors as columns
4: The rows of $\mathbf{E}$ are the data points in R^k. Run k-means on the data points to obtain k clusters.

For motion analysis, spectral clustering is appealing due to its low dependence on the structure of the data, where clustering can be done on motion trajectories rather than simply image points. In 2D motion analysis, spectral clustering has been employed for tasks such as motion segmentation based on point trajectories [57, 60, 45, 19]. Mateus *et al.* [51] use spectral clustering to track 3D trajectories using a multi-camera setup to estimate sparse scene flow. These methods perform motion segmentation in 2D images and perform clustering to generate a sparse motion field rather than the dense motion field in scene flow.

2.4 Summary

The primary drawback to most existing scene flow methods is the inherent computational complexity in their variational framework. Optical flow has had a long history of research with many different methods and also frameworks for more efficient computations. Scene flow on the other hand has seen significantly less work and as such there exist little methods that efficiently estimate scene flow. The lack of computationally efficient RGB-D scene flow methods is the driving motivation behind this work. Rather than using an optimization approach, grouping of similar motions is performed using concepts from spectral clustering. Spectral clustering has seen success in motion analysis for 2D images, in particular for motion segmentation by clustering sparse point trajectories. We apply similar concepts for grouping RGB-D data for approximate dense scene flow.

Chapter 3

Method

The motivation behind the method presented here is from observing that most scenes encountered in everyday life consist of rigidly moving parts. This assumption has been explored in recent state-of-the-art works in multi-view stereo [52, 75] decomposing the scene into rigidly moving planes, and also in RGB-D scene flow [61] that constrains rigid movements locally. In contrast to previous methods that are formulated in a variational framework with explicit data and regularization term, we group these rigid motions using concepts from spectral clustering. The data term, which consists of some defined cost relating the difference in intensity and depth values based on the constraints in Eq. 3.1 and 3.2 is used as edge weights of a graph connecting matching nodes between frame 1 and frame 2 to define the matching affinity between nodes. Instead of a regularization terms, we use a pairwise affinity between nodes based on intensity and motion similarity to group like-motions and enforce every point within the group to have the same rigid motion. The reasoning behind this again follows the basic assumption that most scenes can be decomposed into rigidly moving parts. Similarly to how existing methods use a coarse-to-fine scheme to accommodate larger motions [8, 15], the presented method here uses a multi-resolution voxelization approach to hierarchically group motions together.

The following sections of this chapter are as follows. Section 3.1 introduces the scene flow problem formulation and model and an overview of the method is given in Section 3.2. Sections 3.3 and 3.4 present the core of the algorithm, namely the multi-scale voxelization and affinity graph construction. Computational complexity is analysed in Section 3.6.

3.1 Problem Formulation and Model

We formulate scene flow as a matching problem between two sets of 3D points measured at two different time steps by an RGB-D camera. Let I_1, I_2 represent the intensity images and D_1, D_2 represent the depth images measured at frame 1 and frame 2 respectively. For a given pixel at $\vec{p} = (x, y), \vec{p} \in I, D$ in the image domain, scene flow is the estimation of its 3D translation vector $\vec{o} = (u, v, \zeta), \vec{o} \in \mathbb{R}^3$ from one frame to the next. Here, (u, v) is the optical flow velocity along the axis in the image plane, and ζ represents the change in depth. Estimating $\vec{o}$ is generally done under the following constraints:

$$I_1(x, y) = I_2(x + u, y + v) \tag{3.1}$$

$$D_1(x, y) = D_2(x + u, y + v) - \zeta \tag{3.2}$$

Eq. 3.1 is the brightness constancy assumption commonly used in optical flow (see Eq. 2.6 in Section 2.2.1) and Eq. 3.2 is the analogous depth constancy constraint (see Eq. 2.17 in Section 2.2.3). Rather than estimating $\vec{o}$, where u, v are motions in the image plane, we match points directly in 3D space to better represent the physical scene. That is, we want to find the 3D translation motion $\overrightarrow{T} \in \mathbb{R}^3$:

$$\overrightarrow{T} = \begin{bmatrix} \frac{Z}{f} & 0 & \frac{X}{Z} \\ 0 & \frac{Z}{f} & \frac{Y}{Z} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ \zeta \end{bmatrix} \tag{3.3}$$

where X, Y, Z are the 3D spatial coordinates of the observed point, readily obtained from the pinhole camera model in Eq. 2.4.

Let $P^1 = \{\overrightarrow{P}_i^1\}_{i=1\ldots n}, \overrightarrow{P}_i^1 = (X_i^1, Y_i^1, Z_i^1) \in \mathbb{R}^3$ represent the set of n 3D points of frame 1, with intensity values $Q^1 = \{Q_i^1\}_{i=1\ldots n}, \overrightarrow{Q}_i^1 \in \mathbb{R}$. Similarly, let P^2 represent the set of 3D points of frame 2 with intensity values Q^2. The goal is to find the set of 3D velocity vectors, $T^1 = \{\overrightarrow{T}_i^1\}_{i=1\ldots n}, \overrightarrow{T}_i^1 = (\Delta X_i^1, \Delta Y_i^1, \Delta Z_i^1) \in \mathbb{R}^3$ where $\overrightarrow{T}_i^1$ is the translation of $\overrightarrow{P}_i^1 \in P^1$ to its corresponding point in P^2. This is shown in Figure 3.1.

The 3D data are modelled as a graph $G(\nu, \varepsilon)$ with the data represented by vertices ν. The term ν contains the 3D data from both frame 1 and frame 2. Under this graphical model, scene flow is estimation is performed through grouping ν into k groups. In each group, the vertices from frame 1 correspond to the vertices from frame 2. The 3D data points can then be registered together according to the groupings to find the rigid transformations (rotation and translation) for each group. Figure 3.2 shows this model with two groups as an example. With G, the realization of the groupings is the fundamental problem and the next sections describe the method for creating these groupings.

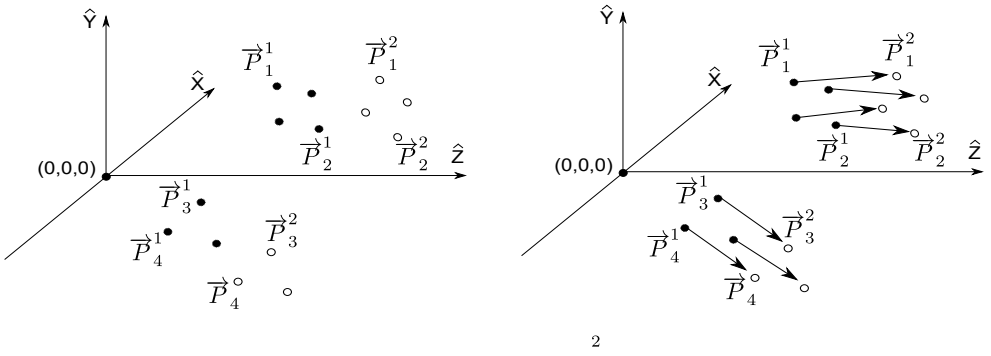

Figure 3.1: Scene flow is finding the 3D translation vectors for each point. Black dots represent points from frame 1 and white circles from frame 2.

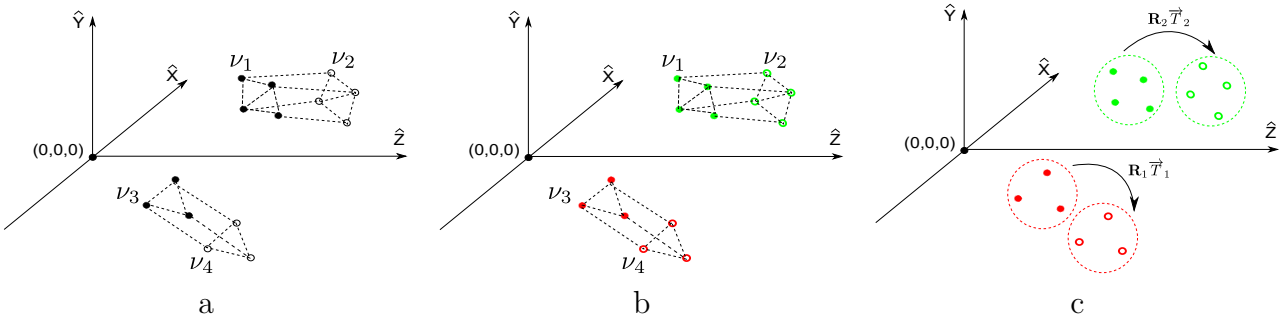

Figure 3.2: Graphical model of the 3D data. a) ν is the union of the 3D data from both frames. b) Groups are created according to motions. Shown here are two groups, colour coded red and green. c) Once the groups have been created, the rigid transformations $\mathbf{R}\vec{T}$ (rotation and translation) can be computed via registration of the 3D data within each group.

3.2 Method Overview

The overall pipeline on a sample scene is shown in Figure 3.3. We use a hierarchical, coarse-to-fine approach by voxelizing the 3D data. Voxelization discretizes the 3D data into a 3D grid, which is a much simpler representation and allows for multiple grid resolutions. The voxelization process is described in Section 3.3. P^1 and P^2 are voxelized at different resolution levels where at each level the voxels are used as vertices to construct a similarity graph (or affinity matrix). The affinity matrix is constructed based on colour and motion similarity between voxels. Using the affinity matrix, the voxels are split into groups via spectral grouping, shown in the middle column of Figure 3.3 with each group colour coded. The construction of the affinity matrix and voxel spectral grouping is described in Section 3.4. Voxels from frame 1 that are in the same group as voxels from frame 2 are presumed to match each other. Matched voxels are shown on the very right column of Figure 3.3, where red voxels are from frame 1 and green voxels are from frame 2 with blue lines linking matched voxels together. The motion field is estimated by computing the rigid transformation between matched voxels via iterative closest point (ICP), described in Section 3.5. Data of matched voxels are propagated forward to the next voxel resolution, thus establishing a hierarchical grouping and matching framework. This process is iterated until a specified final voxel resolution is reached. The procedure is explained further in the next sections.

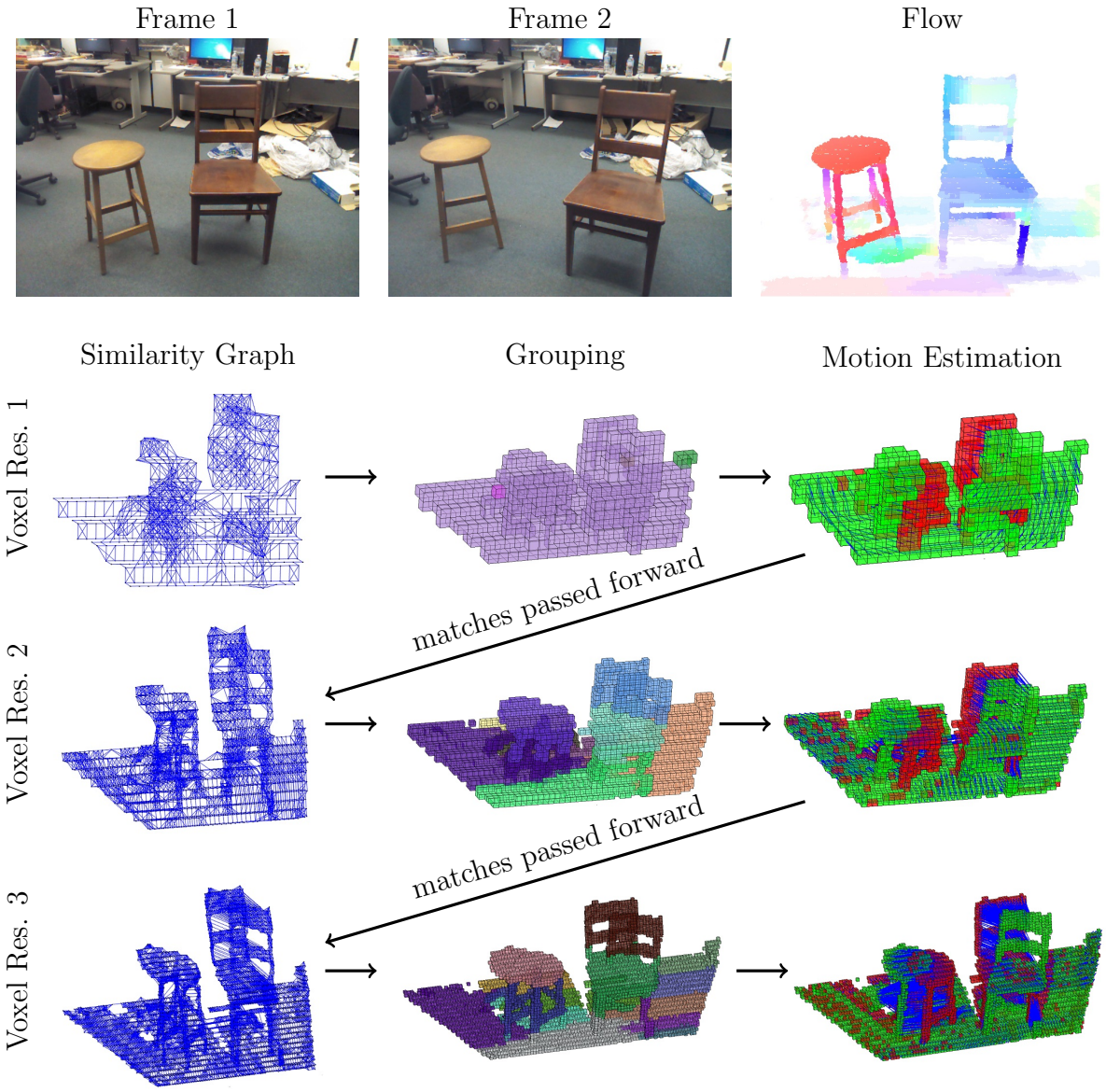

Figure 3.3: Our method pipeline on sample scene. Excluding the top row, each row represents an iteration of the method at a different voxel level and resolution, starting with the coarsest resolution at the top. At each level, information from matching voxels at previous levels are used as priors to help guide the grouping of voxels that exhibit similar motion and object properties.

3.3 Multiscale Voxelization

Voxelization is the process in which a 3D object is converted from its continuous geometric representation to a discrete space representation i.e. 3D grid. It is analogous to how a scene is pixelized to a 2D grid in a digital image. Here, voxelization quantizes the set of 3D points captured by the RGB-D camera to integer values on a 3D grid within some defined volume with dimensions $\Gamma_X \times \Gamma_Y \times \Gamma_Z$. The size of the 3D volume should be large enough such that it encapsulates all 3D data points. A voxel is defined as an element/cube of the 3D grid that can either be occupied or not, depending on how many 3D points lie within that voxel's volume. In addition, an occupied voxel takes on the value of the mean intensity values of all 3D points within the voxel's space. Given $\Gamma_X, \Gamma_Y, \Gamma_Z$ and the width of each voxel cube (the resolution) r, a voxel at coordinates (a, b, c) in the voxel grid V is defined to be:

$$V(a,b,c) = \frac{1}{N} \sum_{1}^{N} Q_i \qquad (3.4)$$
$$[a,b,c]\times r < \vec{P_i} < [a+1,b+1,c+1]\times r$$

where N is the number of 3D points within the voxel's volume. A visualization of a voxel grid's dimensions is shown in Figure 3.4.

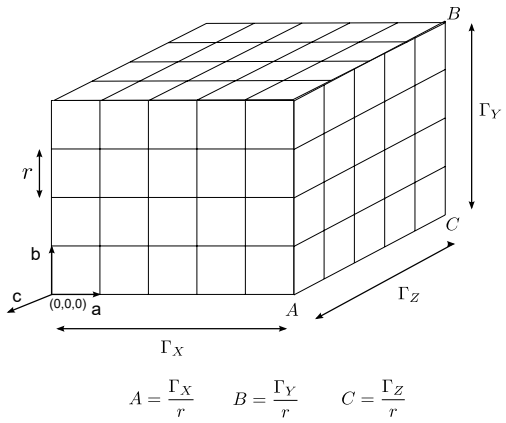

Figure 3.4: Visualization of a voxel grid and its dimension definitions. Γ_X, Γ_Y, Γ_Z, and r are in world units whereas A, B, C are in voxel coordinate units (here $A = 5, B = 4, C = 5$). a, b, c are the coordinate axis of the voxel grid.

Voxelization not only simplifies the data, but it also allows for a coarse-to-fine approach. We voxelize the 3D points at multiple levels, each level representing a different resolution

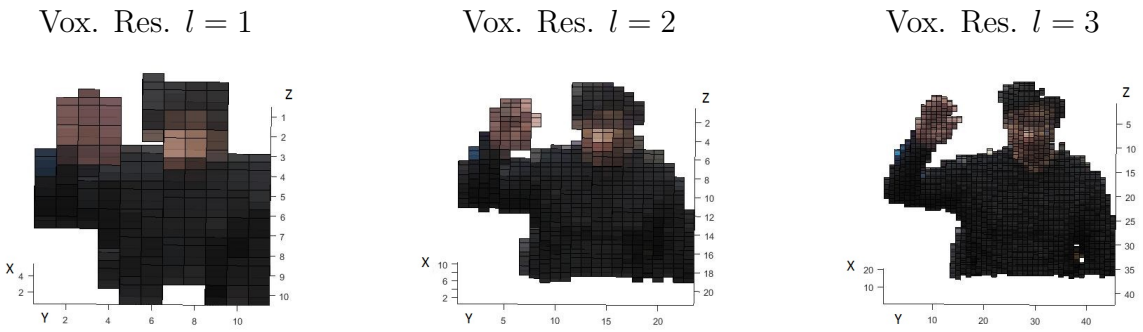

Figure 3.5: Example of voxelization of 3D data at three different resolutions.

of the voxel grid. An example voxelization at three different resolutions is visualized in Figure 3.5 and the voxelization procedure is listed below.

1. Define the volume of interest in 3D space as $\Gamma_X \times \Gamma_Y \times \Gamma_Z$ in world units. The volume should encapsulate all the points in P^1 and P^2.

2. Starting at a level index l, define r_l to be the current level's resolution i.e., the length of the sides of a voxel cube. Define V_l^1, V_l^2 to be $A_l \times B_l \times C_l$ voxel grids of P^1, P^2, where $A_l = \frac{\Gamma_X}{r_l}$, $B_l = \frac{\Gamma_Y}{r_l}$, $C_l = \frac{\Gamma_Z}{r_l}$. The values of each voxel $V_l(a, b, c), a = 1...A_l, b = 1...B_l, c = 1...C_l$ are the mean intensities of the 3D points inside the voxel's volume.

3. In the next voxel level $l + 1$, the voxel grid is at a finer resolution: $r_{l+1} = r_l/2$, $A_{l+1} = A_l \times 2$, $B_{l+1} = B_l \times 2$, $C_{l+1} = C_l \times 2$. V_{l+1} is created once again using Step 2 with the updated dimension values.

Starting at a low resolution voxel grid, we can create groupings between larger motion segments and refine the groupings by passing on matches from a parent node to its children nodes as we iterate to the next level. Using blocks or patches of 3D data in previous works include SphereFlow by Hornacek *et al.* [36] and supervoxels [55]. The framework here can use these patches for improved matching accuracy, but is not necessary. The basic voxelization scheme will be used here for its simplicity, computational efficiency, and lends itself easily to implementation data structures such as an octree.

3.4 Graph Construction and Spectral Grouping

The voxels are used as the data units for the graphical model introduced in Section 3.1. At a voxel level l, let ν_l^1 represent a set of vertices of a graph, where each vertex in ν_l^1 is associated with an occupied voxel of V_l^1. Similarly, let ν_l^2 represent a set of vertices representing occupied voxels in V_l^2. We construct the graph $G_l = (\nu_l, \varepsilon_l)$ that combines vertices from both frames i.e., $\nu_l = \nu_l^1 \cup \nu_l^2$. We want to partition ν_l into groups such that given a grouping $S \subset \nu_l$, the vertices in S that came from ν_l^1 are assumed to match the vertices in S that came from ν_l^2. In this way, the voxels from V_l^1 are matched to the voxels in V_l^2 through these groups. Examples of these groups are shown in the second column of Figure 3.3, where each group is colour coded differently. To carry out this grouping, we employ spectral grouping to partition G. Spectral grouping techniques group vertices in a graph according to their connective relationships described by the edge weights ε. Hence, an appropriate affinity matrix for G must be constructed.

3.4.1 Graph Affinities

Spectral grouping performance is heavily reliant on the affinity matrix, therefore the affinity between data must be set up accordingly. Given the graph $G_l = (\nu_l, \varepsilon_l)$, ε_l is represented as an adjacency matrix $\mathbf{W} = (w_{ij})_{i,j=1...m}$ which describes the values of the weighted edges (or affinities) for m vertices. W is constructed to be composed of two types of affinities, which we call the pairwise affinity and matching affinity, and are described in the following subsections.

Pairwise Affinity

The pairwise affinity, denoted w_{ij}^P, encapsulates the colour and motion similarity between voxel $\nu_{l,i}$ and its neighbouring voxel $\nu_{l,j}$ of the same frame. The assumption here is that if a voxel shares similar colour with its neighbours then it should share a similar motion. The pairwise affinity is decomposed into the colour similarity measure $s_c(\nu_{l,i}, \nu_{l,j})$ and motion similarity measure $s_m(\nu_{l,i}, \nu_{l,j})$:

$$w_{ij}^P = \alpha s_c(\nu_{l,i}, \nu_{l,j}) + \beta s_m(\nu_{l,i}, \nu_{l,j}) \tag{3.5}$$

where α, β are weighting parameters.

The colour similarity measure used is a Gaussian weight parametrized by its variance σ on the mean intensity difference between voxels $\nu_{l,i}, \nu_{l,j}$:

$$s_c(\nu_{l,i}, \nu_{l,j}) = exp\left(\frac{-(\nu_{l,i} - \nu_{l,j})^2}{2\sigma^2}\right) \tag{3.6}$$

Any normalized colour similarity measure from literature can be used here to obtain possibly more robust matches, such as the data terms used in optical flow [22]. However, in practise we found the Gaussian weighted measure to be sufficient. In literature, previous methods have successfully used Gaussian weighted distance metrics [76].

The motion similarity measure weighs the difference in motion vector estimates between voxels $\nu_{l,i}, \nu_{l,j}$. The motion vector of $\nu_{l,i}$, denoted $\vec{m}_{l,i}$, is the coordinate difference between $\nu_{l,i}$ and its matching node. Similarly, $\vec{m}_{l,j}$ is the motion vector for $\nu_{l,j}$. How these matches are found is described in the following subsection under matching affinities. Again a Gaussian weight is used to measure this difference:

$$s_m(\nu_{l,i}, \nu_{l,j}) = exp\left(\frac{-(|\vec{m}_{l,i} - \vec{m}_{l,j}|)^2}{2\sigma^2}\right) \tag{3.7}$$

Together the colour similarity encourages grouping on similar objects, while the motion similarity encourages grouping on similar motion.

Matching Affinity

The matching affinity, denoted w_{ik}^M, is assigned to the edge connecting a voxel $\nu_{l,i}^1 \in V_l^1$ to a voxel $\nu_{l,i}^2 \in V_l^2$. The edge weight connecting the matching voxels is the same as the colour similarity measure:

$$w_{ik}^M = exp\left(\frac{-(\nu_{l,i} - \nu_{l,k})^2}{2\sigma^2}\right) \tag{3.8}$$

Matches are found and propagated in a hierarchical fashion from coarse-to-fine voxel resolution levels. Starting at the first level (coarsest), the matching vertex for $\nu_{1,i}^1$ is the nearest neighbour in V_1^2 to $\nu_{1,i}^1$, Then, the matches are recalculated after applying grouping and motion estimation (see Section 3.5). The matches are then propagated to the next voxel level to its children. That is, given a vertex $\nu_{l,i}$ corresponding to voxel $V_l(a_i, b_i, c_i)$ its parent vertex is the vertex in ν_{l-1} corresponding to voxel $V_{l-1}(\text{floor}(\frac{a_i}{2}, \frac{b_i}{2}, \frac{c_i}{2}))$. This relationship is shown in Figure 3.6.

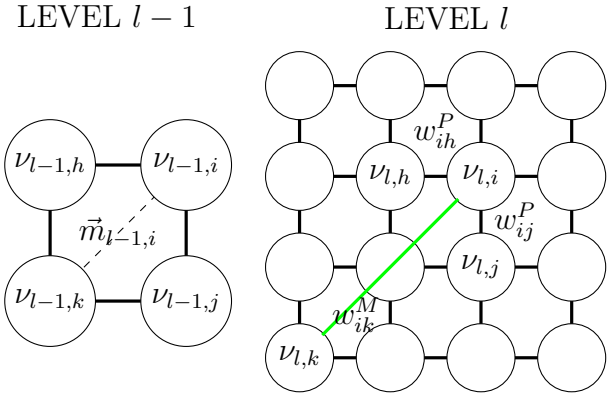

Figure 3.6: Example graph connectivities, shown as a 2D grid, of a voxel level l and $l-1$ for the edge weights of the vertex $\nu_{l,i}$. In this representation, each node from ν_{l-1} subdivides to four nodes to form ν_l. The pairwise weights w^P_{ij}, w^P_{ik} connect $\nu_{l,i}$ to its neighbours and the matching vertex weight w^M_{ik} between $\nu_{l,i}$ and $\nu_{l,k}$ (shown in green) is from the matching of their parent nodes from the previous level $l-1$, shown by the vector $\vec{m}_{l-1,ik}$

.

3.4.2 Grouping Motions

Once the affinity matrix has been constructed from the graph structure, spectral clustering techniques can be deployed to group the data according to motions. We follow the popular spectral grouping method by Shi and Malik [67] to create the groupings in G_l. We effectively approximate a normalized graph cut along the edges of G_l, where we have defined our affinities. To this end, Eigen decomposition is performed on the normalized Laplacian:

$$\mathbf{L} = \mathbf{\Psi}^{-1/2}\mathbf{L}\mathbf{\Psi}^{-1/2} \tag{3.9}$$

where $\mathbf{L}$ is the unnormalized Laplacian matrix and $\mathbf{\Psi}$ is the degree matrix (see Chapter 2.3). Then, k-means clustering is performed on the resulting row data of the matrix with its columns constructed from the first k eigenvectors of $\mathbf{L}$ according to Algorithm 1. The question arises now on how to choose k. Our problem is concerned with finding the motions of individual groups, and so as long as the groups are large enough to contain enough depth features then performance is not sensitive to the choice of k. k should be chosen to loosely represent the expected number of rigid motions in the scene. If a high k value is chosen, the scene is over-grouped or over-segmented with the size of each group being roughly the same due to the normalized cut. Motions for smaller parts can still be estimated so long as there is enough 3D structure within each group. Spectral grouping is

convenient because of its linear nature and, generally, the similarity matrix will be sparse and hence we can make use of efficient methods of finding the eigenvectors of large sparse matrices [76].

3.5 Group Motion Estimation

Once groups of matching voxels have been created, the rigid motions of points from frame 1 to frame 2 within each group can be found by point cloud registration methods such as iterative closest point (ICP) [9]. Given a group that contains n_k number of 3D points $\overrightarrow{P}^1_{i=1...n_k}$ from frame 1 and $\overrightarrow{P}^2_{i=1...n_k}$ from frame 2, we find the 3×3 rotation matrix $\hat{\mathbf{R}}$ and 3×1 translation vector $\hat{\overrightarrow{T}}$ that minimizes the squared Euclidean distance between $\overrightarrow{P}^1_{i=1...n_k}$ and $\overrightarrow{P}^2_{i=1...n_k}$:

$$\hat{\mathbf{R}}, \hat{\overrightarrow{T}} = \arg\min_{\mathbf{R}, \overrightarrow{T}} \sum_{i=1}^{n_k} \|(\mathbf{R}\overrightarrow{P}^1_i + \overrightarrow{T}) - \overrightarrow{P}^2_i\|^2 \tag{3.10}$$

As the name implies, ICP finds $\hat{\mathbf{R}}, \hat{\overrightarrow{T}}$ iteratively by assuming the closest points of $\overrightarrow{P}^1_{i=1...n_k}$ to $\overrightarrow{P}^2_{i=1...n_k}$ correspond to each other and finds the optimal $\mathbf{R}, \overrightarrow{T}$ applied to $\overrightarrow{P}^1_{i=1...n_k}$ based on these correspondences. Then a new set of closest points are found and the process is repeated until a convergence threshold or iteration limit is reached.

After applying the rigid transformation from ICP, a match is made based on the nearest neighbour of the node in frame 1 from the nodes in frame 2. This serves to accommodate for slight non-rigid motions, and it also helps generate some flow estimates when there are no distinct object matches. The 3D vector difference between matched nodes is set as $\vec{m}$ for the next iteration of voxel resolutions, multiplied by the upscale factor.

After the last iteration, the voxel flow vectors from matched nodes are reprojected to the point cloud P^1, and the 3D translation motion $\overrightarrow{T}^1$ is recovered.

3.6 Computational Complexity

The computational time for the grouping method can be broken down as follows. The main steps in the method are creating the similarity matrix, clustering, and ICP. For N number of voxels in ν and k clusters at the finest voxel resolution, the order of the run-time is:

$$\mathcal{O}(Time_{sim} + Time_{cluster} + k \times Time_{ICP}) \tag{3.11}$$

Creating the similarity matrix is straightforward, it requires checking the neighbourhood for each occupied voxel in the voxel grid for the pairwise affinity and adding the matching affinity edge. The maximum number of neighbours in a cubic grid is 16, plus 1 for the matching edge.

$$Time_{sim} = 17N \tag{3.12}$$

The spectral grouping method relies on an Eigen solver. The run-time analysis for a normalized cut is [67]:

$$Time_{cluster} = N \times m \tag{3.13}$$

where m is the number of iterations required for an iterative Eigen solver. Finally, while ICP is not known for its computational efficiency, there exists much work in the field of point cloud registration with variants of ICP to efficiently register point clouds [63, 27]. In general, ICP utilizes nearest neighbour search through a certain number of iterations. Each group runs ICP on at most N points, for i iterations in ICP the run-time is approximately:

$$k \times \log N \times N \tag{3.14}$$

Hence, the order of the total run-time for the algorithm is

$$\mathcal{O}(17N + N \times m + k \times \log N \times N) \tag{3.15}$$

Typically, accurate results can be obtained with fewer than 10000 voxels. For an image resolution of 640 $\times$ 480 pixels = 307200, the proposed method will run in approximately the same amount of time it takes to iterate through all pixels in the image.

Chapter 4

Experimental Results

Evaluation was performed on two RGB-D datasets and the KITTI dataset [23]. The first RGB-D dataset is the UW-VIP RGB-D Scene Flow Dataset [47] (referred to as the UW dataset for short), and will be used for quantitative evaluation. Qualitative results are obtained on selected sequences from the GraphFlow RGB-D dataset [3] which consists of larger motions. Lastly, the KITTI dataset, though technically not using an RGB-D camera, contains recorded data containing RGB and depth information of real scenes from a moving vehicle. For comparison, we evaluate our method against some leading methods in RGB-D scene flow by Sun *et al.* [70] (source code acquired online [1]), which uses a layered RGB-D model, and Ferstl *et al.* [20] (source code acquired online[2]), which is set in a variational framework called CP-Census.

For all experiments, four layers were used in the layered RGB-D method by Sun *et al.* [70] and four voxel levels were used in our method, starting with a voxel grid size of ten. The hyper parameters that need to be set are the weights α, β in Eq. 3.5 to specify the contribution of colour and motion similarity to the affinity matrix respectively and σ for the Gaussian weight. The selection of α, β is intended to find a balance between clustering based on colour and motion. If α is high, more emphasis is placed on neighbouring colour similarity and hence voxels belonging to the same object are more likely to be clustered together and labelled with the same motion. However, if an object contains multiple different motions, such as the different segments of an arm, relying solely on colour similarity will cluster the entire arm as one group and not separate the segments. On the other hand, motion estimates are not as reliable as colour measurements and relying solely on

[1] http://people.seas.harvard.edu/ dqsun/

[2] https://rvlabs.org.icg.tugraz.at/project_page/project_tofusion/project_sceneflow.html

motion estimates will create groups of voxels with holes. Hence, a balance between colour and motion is needed to find individual moving parts of the scene. With the colour and depth image normalized to have max value 1, the parameters were tuned empirically and assigned $\alpha = 0.8$, $\beta = 0.2$, $\sigma = 0.2$ for all of experiments.

4.1 UW Dataset - Quantitative Evaluation

With the lack of ground truth RGB-D scene flow data, some authors have used the Middlebury stereo dataset [65] to perform quantitative analysis [70, 61, 36]. This is not a good representation of scene flow as the camera motion is only along the X direction in a static scene. The UW dataset is composed of pairs of RGB-D frames at two time steps of various scenes. All motion in the scenes are rigid motions with combinations of object and camera movement. Ground truth data of the motions were obtained by manually segmenting out individual moving components in the scene and for which flow fields were determined by performing ICP on the individually segmented components. Image sequences from the UW dataset are shown in Figure 4.2 with their ground truth flow fields projected onto 2D image plane. In this work, the 3D flow fields are visualized by projecting the 3D motion vectors onto the 2D image plane i.e., they are viewed the same way as optical flow. For qualitative visual assessment, the 2D flow vectors are colour coded according to the following colour wheel:

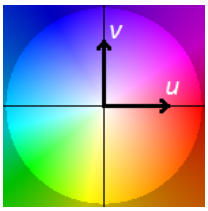

Figure 4.1: 2D flow colour wheel. E.g., left motion ($-u$) is blue, right motion ($+u$) is red.

4.1.1 Evaluation Metric

For quantitative evaluation, our method was run on the UW dataset and the accuracy metrics used are the normalized root mean square error (NRMS) of 3D point differences and average angular error (AAE) [5]. The NRMS valuation measures the deviation of the

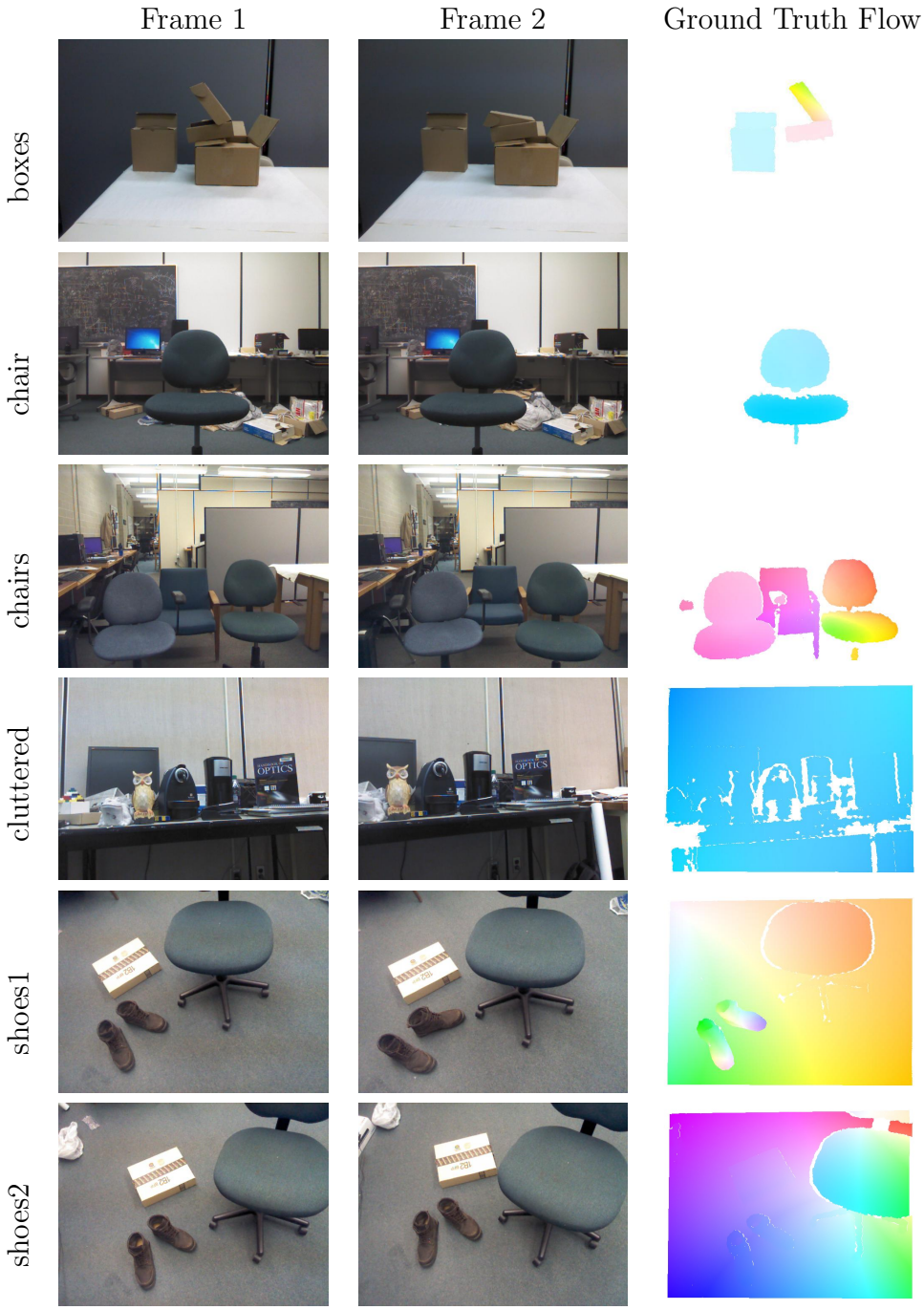

Figure 4.2: Image sequences from the UW-VIP RGB-D Scene Flow image pairs. Shown are the colour images and ground truth flow fields projected to the image domain.

estimated 3D point $\overrightarrow{P}(x,y)$ after applying the estimated scene flow and its true 3D point $\overrightarrow{P}_o(x,y)$ from the true motion.

$$NRMS = \frac{\sqrt{\frac{1}{n}\sum_{\Omega} \|\overrightarrow{P}(x,y) - \overrightarrow{P}_o(x,y)\|^2}}{max\|\overrightarrow{P}_o(x,y)\| - min\|\overrightarrow{P}_o(x,y)\|} \tag{4.1}$$

where Ω is the image domain and n is the number of points. The denominator in Eq. 4.1 is a normalization factor allowing NRMS to be evaluated as a percentage of RMS error from the range of possible motions indepedent of the absolute depth value. The original motivation behind this metric was that errors in 2D flow do not necessarily correlate with 3D motion errors, since 3D errors will be of different scale than the 2D error and certain errors in 3D (such as motions along the Z direction) are not captured in 2D [5].

As well, we use the average angular error (AAE), defined in radians, between $\overrightarrow{P}(x,y)$ and $\overrightarrow{P}_o(x,y)$

$$AAE = \frac{\sum_{\Omega} cos^{-1} \frac{\overrightarrow{P}(x,y)\cdot\overrightarrow{P}_o(x,y)}{|\overrightarrow{P}(x,y)||\overrightarrow{P}_o(x,y)|}}{n} \tag{4.2}$$

The angular error measures the angular difference between the estimated flow vectors and the true flow vectors.

4.1.2 Results

The results of our method, layered RGB-D [70], and CP-Census [20] are reported in Table 4.1. While the layered RGB-D method of Sun *et al.* performed the best overall in accuracy, our results were comparable and in some cases outperformed the other two methods. Visualizations of the 2D flow fields are shown in Figure 4.3. It can be seen that the scenes where our performance dropped were for scenes with motion on a complex background. In these situations, wrong voxels can be grouped together that leads to error in matching. As well, the aperture problem persists for planar surfaces causing ambiguity in the scene. CP-Census and the layered RGB-D methods were seen to perform well in these cases due to their global energy minimization approach with smoothness constraint. It is interesting to note that despite the 2D flow visualizations looking more favourable for the layered method on the *shoes1* sequence, our method had lower NRMS. Indeed, the 3D point differences were smaller for our method. This is probably because, unlike Sun *et al.*, we do not use optical flow for initialization. Ultimately our method was very comparable to other two methods in terms of accuracy on the UW dataset.

Table 4.1: Quantitative results of scene flow methods on the UW dataset. NRMS in %, AAE in radians

Sequence	Ours		Layered		aTVG	
	NRMS	AAE	NRMS	AAE	NRMS	AAE
boxes	2.05	**0.005**	**1.30**	0.007	1.63	0.007
chair	0.52	0.006	**0.50**	**0.004**	6.96	0.043
chairs	**2.21**	**0.018**	4.02	0.019	3.59	0.020
cluttered	10.85	0.090	**5.38**	**0.008**	18.86	0.159
shoes1	**4.67**	0.053	5.15	**0.006**	8.61	0.094
shoes2	7.06	0.053	**3.25**	**0.014**	11.22	0.104

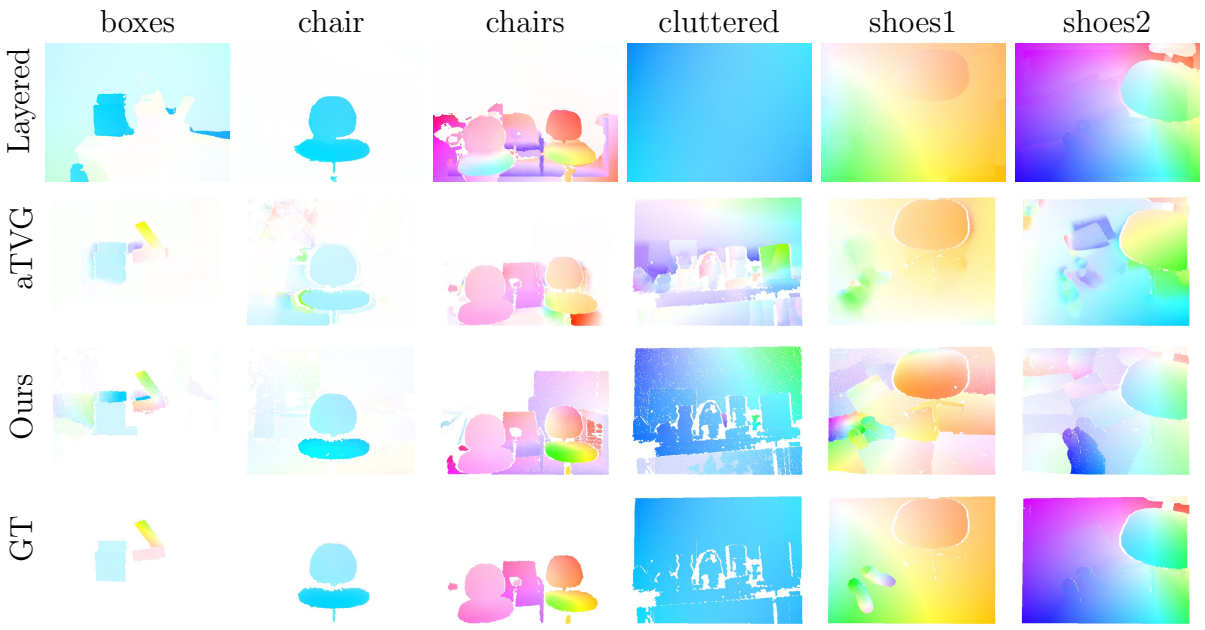

Figure 4.3: Visualizations of the estimated flow fields on the UW dataset. We can see that our method has difficulty matching homogeneous areas of the background motion, but can reliably estimate the flow of individual objects in the foreground. As expected, variational methods perform well to smooth out background motions.

4.2 GraphFlow - Qualitative Evaluation

This section shows some qualitative results of our method on scenes from the GraphFlow dataset [3] that was used to evaluate their method on large motion scene flow. The results are visualized in Figure 4.4. Looking at the 2D representation of the scene flow and the flow circle in the bottom left corner we can see that the estimated motion is consistent with the expected flow. For comparison, we also show the layered RGB-D method [70] that showed the highest accuracy on our UW dataset.

In the *Walking* scene, the blue silhouette indicates the person moving to the left, which is expected. The errors in the background are due to improper groupings caused by occlusion, which are not explicitly handled yet. The individual motions in the *Tea* scene were also correctly grouped. Specifically, the kettle, arm, and head movements were captured, which are the most important movements in the scene. The layered method does shower smoother motion for the *Party*. Observing more closely, we can see that our method was able to distinctly pick out the arm movement which was lost in the layered method due to smoothing. Of particular interest is the *Hammer* sequence, in which the layered RGB-D method appears to have gotten the incorrect motion. The *Hammer* scene has a very large motion of a relatively small object, and it is seen that our method is still able to handle this case. Visually, we see that the general motions of each scene were estimated correctly by our method.

4.3 KITTI Dataset

We further evaluate the presented method on the KITTI dataset [23]. This dataset is taken from a vehicle moving through different outdoor environments and is equipped with stereo cameras and a Velodyne LiDAR sensor for depth measurements. While the system is not an RGB-D camera, it provides calibrated RGB and depth images. However, the depth image is not dense and instead quite sparse, which may not be typical of RGB-D depth images. Despite this, this dataset is representative of real world scenarios of scenes that may be encountered by an autonomous vehicle and where data may be missing or noisy. The results are shown below, where the RGB image from two time steps, our flow estimate, the flow estimate by Sun *et al.* [70], and ground truth flow maps are shown. The error metric used for evaluation in this dataset is based on 2D flow and is the percentage of pixels whose flow estimates are beyond a certain error threshold compared to the ground

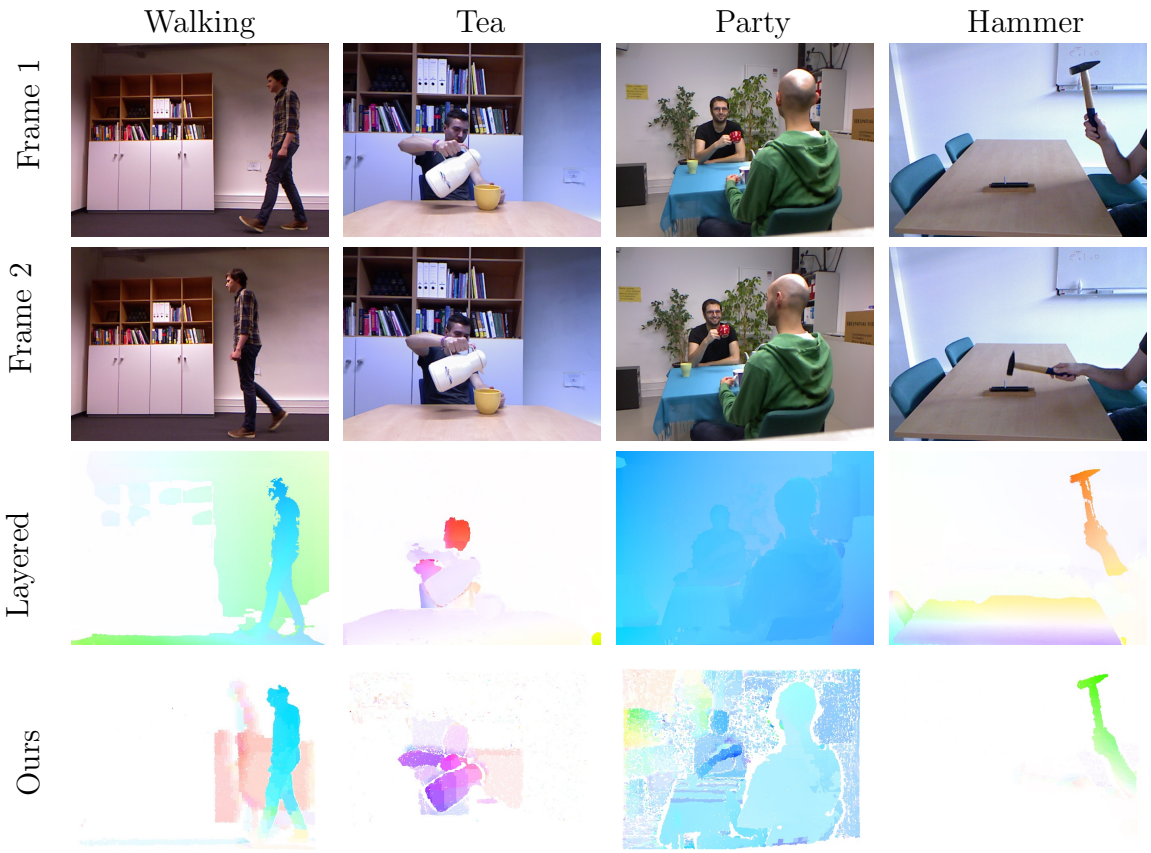

Figure 4.4: Qualitative results of running our method on the GraphFlow dataset. It is seen that the reported 2D Flow are consistent with the expected flow looking at the image sequence from Frame 1 to Frame 2.

truth i.e.,

$$error = \frac{\#\text{ bad pixels}}{\#\text{ total pixels}} \tag{4.3}$$

Let $\overrightarrow{F} = [u, v]$ represent a pixel's 2D flow estimate and let $\overrightarrow{F_o} = [u_o, v_o]$ represent the ground truth flow for that pixel. Then a pixel is determined bad if:

$$|\overrightarrow{F} - \overrightarrow{F_o}| > \tau_1 \text{ and } \frac{|\overrightarrow{F} - \overrightarrow{F_o}|}{|\overrightarrow{F_o}|} > \tau_2 \tag{4.4}$$

for error thresholds τ_1, τ_2. Reported here are results for $\tau_1 = 3$ pixels and $\tau_2 = 10\%$. As well, since the Velodyne data is highly sparse and unusable for scene flow for long distances, we truncate the evaluation to measurements of the physical world within 20m.

In Figure 4.5, the scene flow sequences are of the car moving on city streets with other cars on the road. Here, the rigid motions of other vehicles were correctly estimated by our method. However, on flat surfaces such as the road, our method had difficulty, which accounts for the large errors since the road takes up a large portion of the image. This is because our method uses ICP, which depends on there being depth features to estimate rigid motion. A flat surface such as the road provides ambiguity in matching patches.

In Figure 4.6, the vehicle is stationary and other moving vehicles are observed. Here, our method is able to correctly estimate the motions of the other vehicles by correctly grouping their rigid motions. Some incorrect groupings can lead to wrong motions, such as the traffic light post in sequence 000046. This can happen when there are multiple possible good matches, as is the case for two poles.

Finally, Figure 4.7 shows sequences of the car moving and with traffic moving in perpendicular motion to the vehicle's motion. Once gain, the motion of the other vehicles are correctly captured. Large errors here are due to two factors. One is error in the road motion, as before. The other source of error is due to the fact that our method estimates approximate motion, that is, the motion is more discretized due to our voxelization process. Since the evaluation metric used in this dataset is based on a strict error threshold, even the correct general motion may be deemed as erroneous.

In summary, many object rigid motions were correctly recovered by our method on the KITTI dataset. We see that even the layered method by Sun *et al.* struggled with this dataset. The depth information is much sparser than regular RGB-D cameras, and generally this dataset is not entirely suitable for RGB-D methods. However, it provides as close to real life data as possible in an application that would realistically have similar depth information on autonomous vehicles.

4.4 Run-time Performance

One of the major advantages to using the proposed hierarchical grouping approach to estimating scene flow is a major reduction in run-time and computational complexity. Working with groups and reasoning about the motion of independently moving groups of points inherently scales better with image size and resolution, since the number of moving parts in the scene is limited. The hierarchical approach also allows for information

Table 4.2: Run-time (s) for experimental data sequences.

UW DATASET			
Sequence	Ours	Sun *et al*[70]	Ferstl *et al.*[20]
boxes	**81**	664	1935
chair	**76**	1001	2320
chairs	**29**	819	1949
cluttered	**78**	997	1856
shoes1	**51**	906	1900
shoes2	**64**	738	1869
Average	**63**	854	1971
KITTI DATASET			
000005	**53**	876	NA
000020	**61**	712	NA
000046	**35**	680	NA
000047	**41**	675	NA
000086	**54**	702	NA
000167	**58**	645	NA
Average	**51**	715	NA

to be passed from one level to the next, constraining the search space to find matches between voxels. Compared to optimization-based methods with an energy function defined at the pixel level, the issue of convergence is not an issue. While there is a risk of finding mismatches, the performance behaviour of our method is predictable and can be designed around.

All experiments were run on a Intel i7-4770 3.4GHz CPU using Matlab implementation. The run-time for each data sequence is reported in Table 4.2. Our method clearly outperforms the other two methods in computation time, averaging around 1 minute compared to Sun *et al.*'s 14 minutes and 30 minutes for Ferstl *et al.* Of course, any algorithm's run-time performance will depend on its implementation. It should be noted that the current implementation of the method is in pure Matlab code with minimal optimization, leaving plenty of room for improvements. At the same time, a majority of the heavy computations in our method involve computing the edge weights for each voxel, which is highly parallelizable and can take advantage of parallel architectures such as multi-core GPUs.

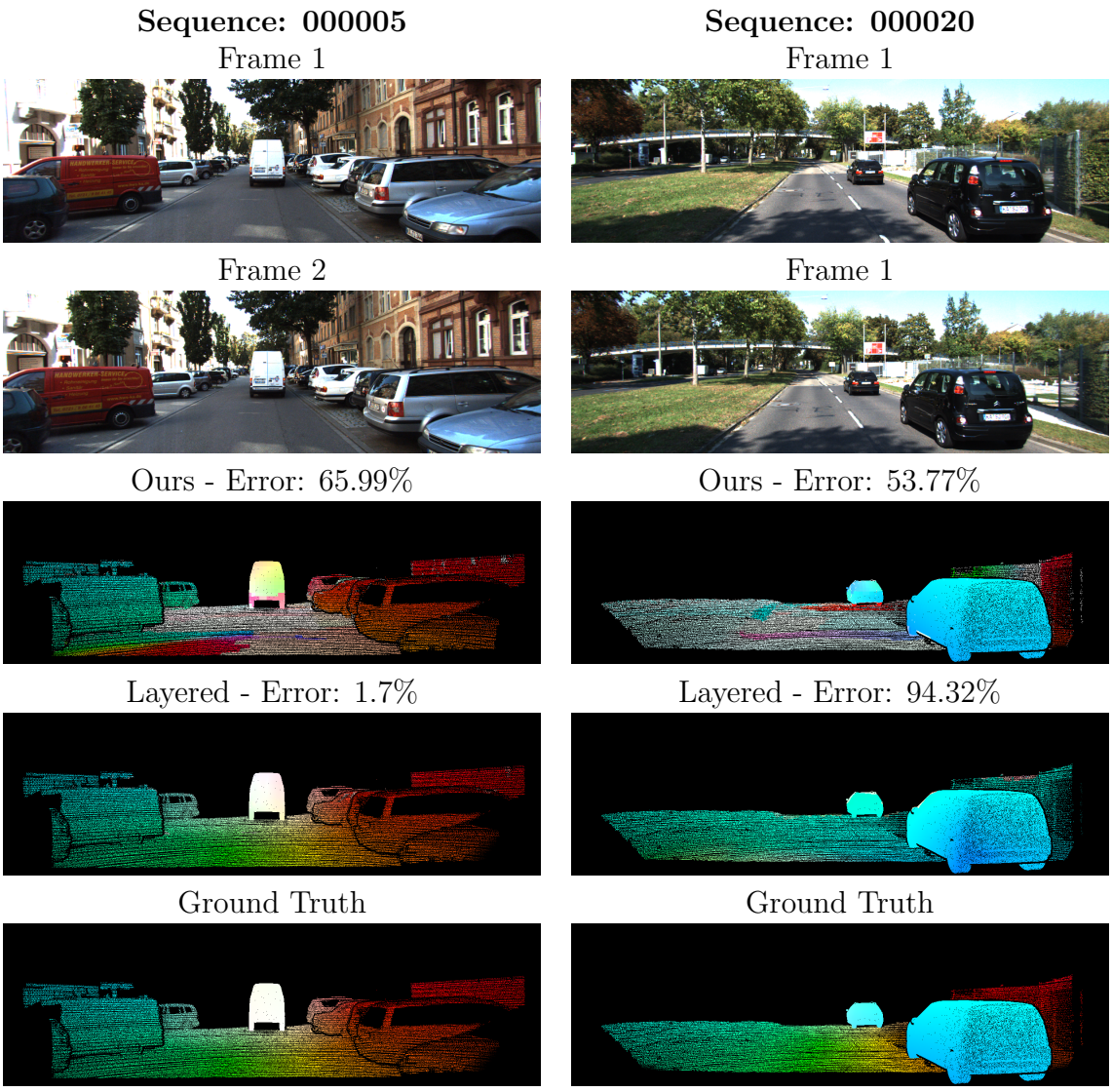

Figure 4.5: KITTI sceneflow dataset sequences 000005 and 000020 of moving car. Our method struggles with recover motion on flat planes, such as the road, due to lack of depth features. However, the motion of other objects and vehicles are estimated correctly.

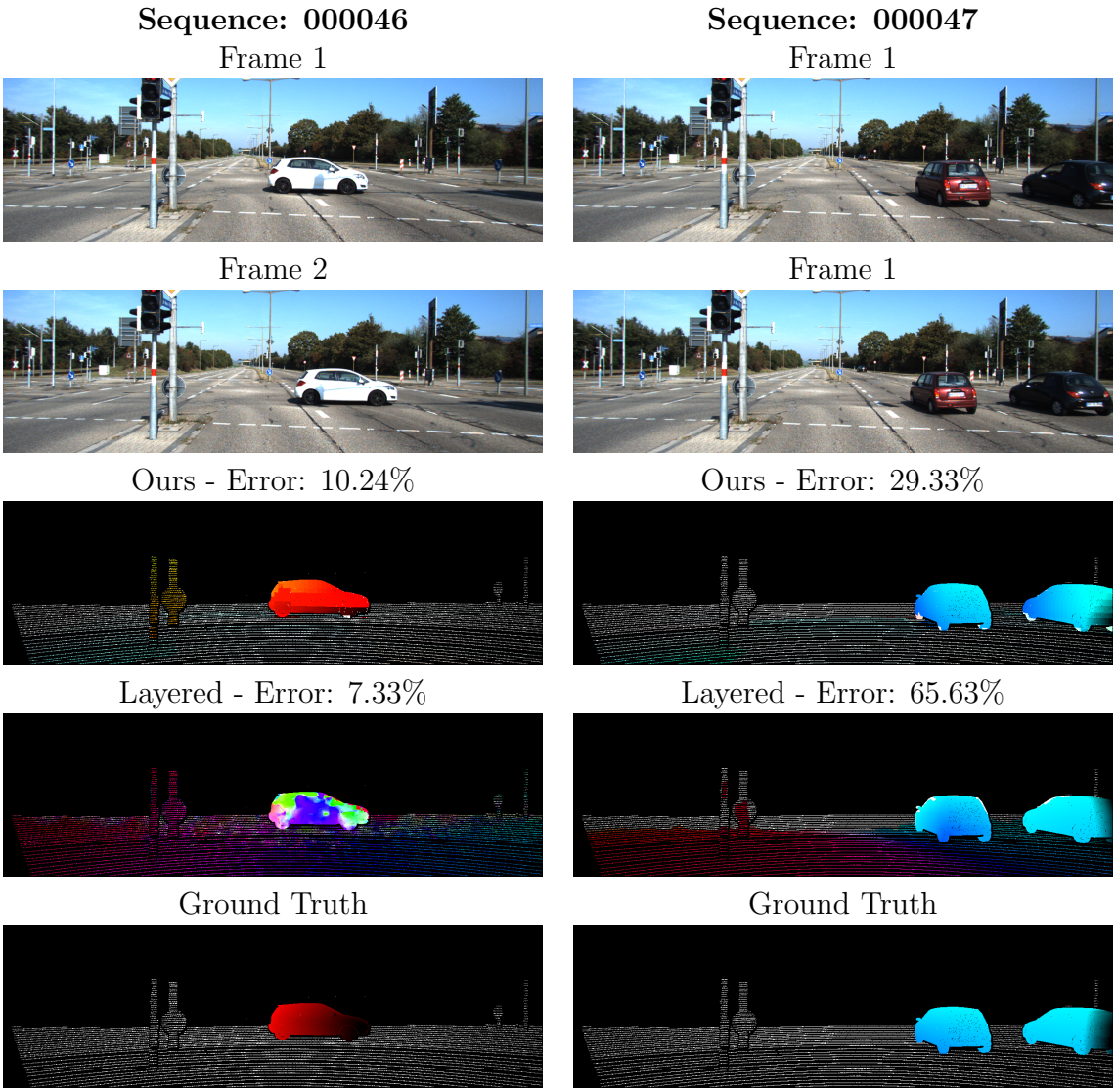

Figure 4.6: KITTI sceneflow dataset sequences 000046 and 000047 of other moving cars. It is seen that individual vehicle motions are estimated correctly.

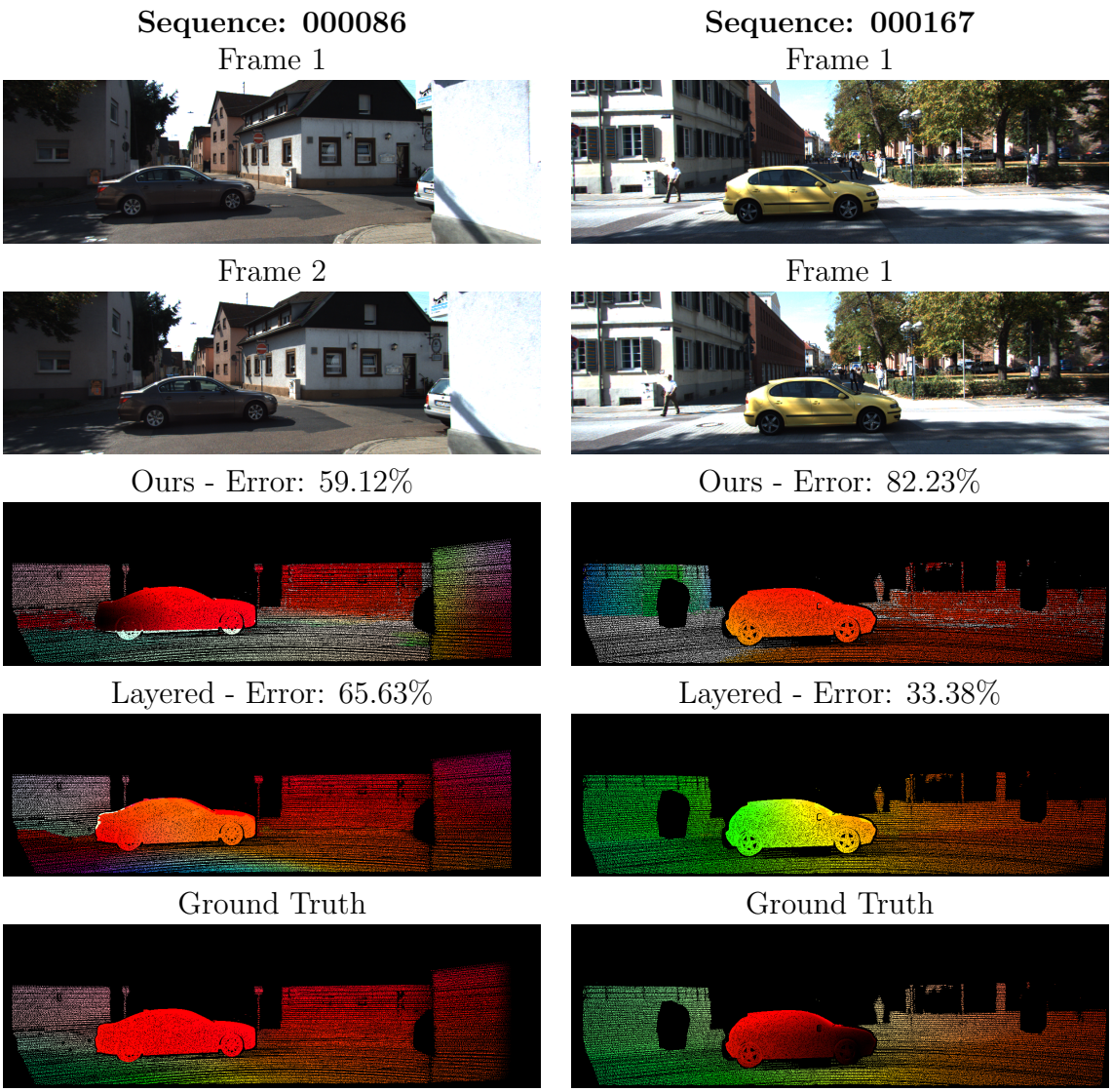

Figure 4.7: KITTI sceneflow dataset sequences 000086 and 000167 of moving car. Again, featureless regions are more erroneous but the motion of individual objects are estimated well.

Chapter 5

Conclusion and Future Work

The advancement of RGB-D camera technology facilitates a slew of computer vision and robotic vision tasks. This work dealt with the fundamental problem of scene flow using RGB-D cameras. Scene flow allows robots to better understand their dynamic environments, and is also an incredibly powerful tool that can be leveraged by other vision tasks. Existing RGB-D scene flow methods have heavily focused on scene flow accuracy rather than efficiency. As such, most state-of-the-art methods are not suitable for real-time applications. To address the lack of efficient RGB-D scene flow methods, this work introduced a new method of estimating RGB-D scene flow that performs significantly faster than most existing methods while maintaining decent accuracy. The proposed method groups objects and rigid motions together using spectral grouping techniques to match 3D voxels between frames and handles large object motion through a hierarchical coarse-to-fine approach.

Chapter 3 formulated the method as a way of finding matching correspondences in 3D data between two frames using spectral grouping techniques to group similar motions together. The method used a hierarchical voxelization approach to simplify the large amount of 3D data produced by RGB-D cameras. The construction of the affinity graph for spectral grouping was also described for grouping the data. The final motions were estimated using standard point cloud registration techniques, such as ICP, on the groups of 3D data. The run-time analysis illustrated how the proposed method is not only tractable but also holds promise for real-time performance.

Chapter 4 demonstrated the viability of the method on different datasets. Experimental results showed significantly faster run-times with slightly lower accuracy than state-of-the-art methods in RGB-D scene flow on the VIP dataset and showed good qualitative results on scenes with larger motion. On the custom VIP dataset, individual object motions were

estimated with very comparable accuracy to state-of-the-art methods. Qualitative results on the Graphflow dataset showed visually consistent motion estimates. On the KITTI benchmark dataset, which consisted of real scenes that can be encountered by a moving vehicle, our method was able to recover the motions of individual objects in the scene but struggled with homogeneous backgrounds such as the road. It was shown that the run time of our method is an order of magnitude faster than other RGB-D scene flow methods.

The main contribution of this work was to present a new and novel RGB-D scene flow method that demonstrated significantly faster run times than existing state-of-the-art while maintaining high accuracy. The main findings are summarized as follows:

1. Independent 3D motions in many different types of scenes can be grouped together using spectral clustering techniques by assigning appropriate edges weights in the affinity matrix based on colour and motion similarity.

2. Voxelization of 3D data can greatly simplify and speed up 3D data processing. A coarse-to-fine resolution approach can account for greater motions in RGB-D data.

3. Rigid motions of objects can be accurately recovered via groupings provided enough depth features are present. Homogeneous regions in colour / depth are still ambiguous and difficult to estimate motions for.

5.1 Future Work

Evidently the main advantage of the method presented is the increased run-time performance. Taking into consideration the unoptimized Matlab implementation and the run-time analysis presented in Section 3.6, real-time performance seems very promising. Also, much of the computation lies in the voxel operations of generating the similarity graph and finding matches. These operations are easily parallelizable on a GPU to operate on each voxel individually. Finally, accuracy does not suffer for rigid object motions, as shown by the quantitative results on the VIP dataset.

There are some limitations to the method. Firstly, it is not suitable for scenes consisting of mostly non-rigid motions such as fluids. As well, homogeneous regions in both colour and depth is still challenging without a global regularization model. This was shown in the errors on the road segments of the KITTI dataset. Finally, very fine motions may be lost due to discretization in the voxelization procedure.

The method presented here shows promise as a viable approach to accurate real-time scene flow estimation. There are still many opportunities in which the current method can be improved. One potential task is accounting for noise and occlusion in the depth measurements, as this could lead to poor grouping results. Noise is an existing issue with RGB-D cameras, especially around object boundaries. Explicit models can be used or more robust grouping methods can be explored based on local 3D statistics. A pre-processing step to denoise the point cloud could also be used, such as statistical outlier filters. To account for occlusion, a forwards-backwards matching process can be employed to identify regions that are not seen by both frames. Other occlusion models borrowed from stereo vision literature can also be explored.

To further improve voxel matches, we can mitigate the inherent ambiguity in homogeneous scenes by incorporating a global regularization as a post processing step, where higher level smoothness term between voxel groups can be explored. The added benefit of a refinement step is to account for finer motions that may have been lost due to voxelization. For quicker refinement, guided image filters may be used to smooth the errors rather than using a regularization framework.

As well, explicit constraints can be added to the spectral clustering algorithm, such as forcing matches based on image feature matching between frames, or dynamically sizing the clusters based on local depth features. While the method presented here makes use of spectral clustering algorithms, other grouping methods can also be explored, such as hierarchical K-means, nearest neighbour, or agglomerative algorithms.

Various other aspects of the method can also be improved. The voxelization procedure presented here was a basic voxelization scheme for the purpose of facilitating a hierarchical approach. In the field of 3D discrete topology, other voxelization algorithms can be explored to create more consistent surface voxels and separation to determine its effects on motion estimation performance. As mentioned in this work, other methods other than voxels can be used as 'patches'of 3D data, such as spherical patches to provide better matching performance. Similarly to optical flow, there is still room to explore correlation metrics to use for determining similarity between patches of 3D data. The metric should be a hybrid of colour similarity and 3D geometric similarity and account for other factors such as noise and changes in illumination from one frame to the next.

Additional future work involves exploring applications of real-time scene flow using the proposed method as input. A suitable application for our method includes action recognition, since arm and hand motions can be estimated accurately using our method. As well, the scene flow estimation can be used for navigating and avoiding obstacles for aerial robotics in indoor environments.

References

[1] Google tango (online). http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.htmlre, 2016.

[2] Xtion pro (online). https://www.asus.com/3D-Sensor/Xtion_PRO/, 2016.

[3] Hassan Abu Alhaija, Anita Sellent, Daniel Kondermann, and Carsten Rother. Graphflow–6d large displacement scene flow via graph matching. In *Pattern Recognition*, pages 285–296. Springer, 2015.

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics-TOG*, 28(3):24, 2009.

[5] Tali Basha, Yael Moses, and Nahum Kiryati. Multi-view scene flow estimation: A view centered variational approach. *International journal of computer vision*, 101(1):6–21, 2013.

[6] Basler. Basler 3d cameras (online). http://www.baslerweb.com/en/products/cameras/3d-cameras, 2016.

[7] Florian Becker, Bernhard Wieneke, Stefania Petra, Andreas Schroder, and Christoph Schnorr. Variational adaptive correlation method for flow estimation. *IEEE Transactions on Image Processing*, 21(6):3053–3065, 2012.

[8] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *European conference on computer vision*, pages 237–252. Springer, 1992.

[9] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[10] Gabriele Bleser and Gustaf Hendeby. Using optical flow as lightweight slam alternative. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 175–176. IEEE, 2009.

[11] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Learning hierarchical sparse features for rgb-(d) object recognition. *The International Journal of Robotics Research*, 33(4):581–599, 2014.

[12] Terrance E Boult and L Gottesfeld Brown. Factorization-based segmentation of motions. In *Visual Motion, 1991., Proceedings of the IEEE Workshop on*, pages 179–186. IEEE, 1991.

[13] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pages 25–36. Springer, 2004.

[14] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):500–513, 2011.

[15] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International Journal of Computer Vision*, 61(3):211–231, 2005.

[16] Antonin Chambolle and Thomas 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.

[17] Zhuoyuan Chen, Hailin Jin, Zhe Lin, Scott Cohen, and Ying Wu. Large displacement optical flow from nearest neighbor fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2443–2450, 2013.

[18] Isaac Cohen. Nonlinear variational method for optical flow computation. In *PROCEEDINGS OF THE SCANDINAVIAN CONFERENCE ON IMAGE ANALYSIS*, volume 1, pages 523–523. PROCEEDINGS PUBLISHED BY VARIOUS PUBLISHERS, 1993.

[19] Liangjing Ding, Adrian Barbu, and Anke Meyer-Baese. Motion segmentation by velocity clustering with estimation of subspace dimension. In *Asian Conference on Computer Vision*, pages 491–505. Springer, 2012.

[20] David Ferstl, Gernot Riegler, Matthias Ruether, and Horst Bischof. Cp-census: A novel model for dense variational scene flow from RGB-D data. In *BMVC*, 2014.

[21] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern recognition*, 41(1):176–190, 2008.

[22] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: a survey. *Computer Vision and Image Understanding*, 134:1–21, 2015.

[23] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[24] Gene H Golub and Charles F Van Loan. Matrix computations. johns hopkins studies in the mathematical sciences, 1996.

[25] Paulo FU Gotardo, Tomas Simon, Yaser Sheikh, and Iain Matthews. Photogeometric scene flow for high-detail dynamic 3d reconstruction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 846–854, 2015.

[26] Jens-Malte Gottfried, Janis Fehr, and Christoph S Garbe. Computing range flow from multi-modal kinect data. In *Advances in Visual Computing*, pages 758–767. Springer, 2011.

[27] Michael Greenspan and Mike Yurick. Approximate kd tree search for efficient icp. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pages 442–448. IEEE, 2003.

[28] Simon Hadfield and Richard Bowden. Kinecting the dots: Particle based scene flow from depth sensors. In *2011 International Conference on Computer Vision*, pages 2290–2295. IEEE, 2011.

[29] Simon Hadfield and Richard Bowden. Scene particles: Unregularized particle-based scene flow estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(3):564–576, 2014.

[30] Rostam Affendi Hamzah, Rosman Abd Rahim, and Zarina Mohd Noh. Sum of absolute differences algorithm in stereo correspondence problem for stereo matching in computer vision application. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 1, pages 652–657. IEEE, 2010.

[31] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[32] Patrick Héas, Cédric Herzet, and Etienne Mémin. Bayesian inference of models and hyperparameters for robust optical-flow estimation. *IEEE Transactions on Image Processing*, 21(4):1437–1451, 2012.

[33] Heptagon. Sensing (online). http://hptg.com/product/#imaging, 2016.

[34] Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D flow: Dense 3-d motion estimation using color and depth. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2276–2282. IEEE, 2013.

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

[36] Michael Hornacek, Andrew Fitzgibbon, and Carsten Rother. Sphereflow: 6 dof scene flow from rgb-d pairs. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3526–3533. IEEE, 2014.

[37] Intel. Realsense overview (online). http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.htmlre, 2016.

[38] Michael Isard and John MacCormick. Dense motion and disparity estimation via loopy belief propagation. In *Asian conference on computer vision*, pages 32–41. Springer, 2006.

[39] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers. A primal-dual framework for real-time dense RGB-D scene flow. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[40] Mariano Jaimez, Mohamed Souiai, Jörg Stückler, Javier Gonzalez-Jimenez, and Daniel Cremers. Motion cooperation: Smooth piece-wise rigid scene flow from rgb-d images. In *3D Vision (3DV), 2015 International Conference on*, pages 64–72. IEEE, 2015.

[41] Takeo Kanade, Hiroshi Kano, Shigeru Kimura, Atsushi Yoshida, and Kazuo Oda. Development of a video-rate stereo machine. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, pages 95–100. IEEE, 1995.

[42] Kenichi Kanatani and Yasuyuki Sugaya. Multi-stage optimization for multi-body motion segmentation. In *Australia-Japan Advanced Workshop on Computer Vision*, volume 2, page 7, 2003.

[43] Effrosini Kokiopoulou, Jie Chen, and Yousef Saad. Trace optimization and eigenproblems in dimension reduction methods. *Numerical Linear Algebra with Applications*, 18(3):565–602, 2011.

[44] Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions using graph cuts. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 508–515. IEEE, 2001.

[45] Fabien Lauer and Christoph Schnörr. Spectral clustering of linear subspaces for motion segmentation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 678–685. IEEE, 2009.

[46] Antoine Letouzey, Benjamin Petit, and Edmond Boyer. Scene flow from depth and color images. In *BMVC 2011-British Machine Vision Conference*, pages 1–11. BMVA Press, 2011.

[47] Francis Li, Alexander Wong, and John Zelek. Hierarchical grouping approach for fast approximate rgb-d scene flow. In *Conference on Computer and Robot Vision*. IEEE, 2016.

[48] Ce Liu, William T Freeman, Edward H Adelson, and Yair Weiss. Human-assisted motion annotation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[49] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[50] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[51] Diana Mateus and Radu Horaud. Spectral methods for 3-d motion segmentation of sparse scene-flow. In *Motion and Video Computing, 2007. WMVC'07. IEEE Workshop on*, pages 14–14. IEEE, 2007.

[52] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[53] Microsoft. Kinect hardware (online). https://developer.microsoft.com/en-us/windows/kinect/hardware, 2016.

[54] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[55] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmentation-supervoxels for point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2027–2034, 2013.

[56] Jaesik Park, Tae Hyun Oh, Jiyoung Jung, Yu-Wing Tai, and In So Kweon. A tensor voting approach for multi-view 3d scene flow estimation and refinement. In *ECCV*, 2012.

[57] JinHyeong Park, Hongyuan Zha, and Rangachar Kasturi. Spectral clustering for robust motion segmentation. In *European Conference on Computer Vision*, pages 390–401. Springer, 2004.

[58] Ioannis Patras, Nicolas Alvertos, and Georgios Tziritas. Joint disparity and motion field estimation in stereoscopic image sequences. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 1, pages 359–363. IEEE, 1996.

[59] PMD. Pmd (online). http://www.pmdtec.com/, 2016.

[60] Huaijun Qiu and Edwin R Hancock. Robust multi-body motion tracking using commute time clustering. In *European Conference on Computer Vision*, pages 160–173. Springer, 2006.

[61] Julian Quiroga, Thomas Brox, Frédéric Devernay, and James Crowley. Dense semi-rigid scene flow estimation from RGBD images. In *Computer Vision–ECCV 2014*, pages 567–582. Springer, 2014.

[62] Shankar R Rao, Roberto Tron, René Vidal, and Yi Ma. Motion segmentation via robust subspace separation in the presence of outlying, incomplete, or corrupted trajectories. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[63] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[64] Kyosuke Sato, Haiyuan Wu, and Qian Chen. High-speed and high-accuracy scene flow estimation using kinect. *Procedia Computer Science*, 22:945–953, 2013.

[65] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–195–I–202. IEEE, 2003.

[66] Loren Arthur Schwarz, Artashes Mkhitaryan, Diana Mateus, and Nassir Navab. Human skeleton tracking from depth data using geodesic distances and optical flow. *Image and Vision Computing*, 30(3):217–226, 2012.

[67] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[68] David Shulman and J-Y Herve. Regularization of discontinuous flow fields. In *Visual Motion, 1989., Proceedings. Workshop on*, pages 81–86. IEEE, 1989.

[69] Hagen Spies, Bernd Jähne, and John L Barron. Range flow estimation. *Computer Vision and Image Understanding*, 85(3):209–231, 2002.

[70] Deqing Sun, Erik B Sudderth, and Hanspeter Pfister. Layered RGBD scene flow estimation. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 548–556. IEEE, 2015.

[71] Roberto Tron and René Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[72] Markus Unger, Manuel Werlberger, Thomas Pock, and Horst Bischof. Joint motion estimation and segmentation of complex scenes with label costs and occlusion modeling. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1878–1885. IEEE, 2012.

[73] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 722–729. IEEE, 1999.

[74] Rene Vidal, Yi Ma, and Shankar Sastry. Generalized principal component analysis (gpca). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1945–1959, 2005.

[75] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

[76] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[77] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011.

[78] Andreas Wedel and Daniel Cremers. *Stereo scene flow for 3D motion analysis*. Springer Science & Business Media, 2011.

[79] Li Xu, Jiaya Jia, and Yasuyuki Matsushita. Motion detail preserving optical flow estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1744–1757, 2012.

[80] Koichiro Yamaguchi, David McAllester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *Computer Vision–ECCV 2014*, pages 756–771. Springer, 2014.

[81] Jingyu Yan and Marc Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *European conference on computer vision*, pages 94–106. Springer, 2006.

[82] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pages 214–223. Springer, 2007.

[83] Luca Zappella, Xavier Lladó, and Joaquim Salvi. Motion segmentation: a review. In *Proceedings of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 398–407. IOS Press, 2008.

[84] Ye Zhang and Chandra Kambhamettu. On 3d scene flow and structure estimation. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–778. IEEE, 2001.

[85] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[86] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[87] Henning Zimmer, Andrés Bruhn, and Joachim Weickert. Optic flow in harmony. *International Journal of Computer Vision*, 93(3):368–388, 2011.