

A Thesis for the Degree of Ph.D. in Engineering

Real-time 3D Reconstruction of Dynamic Scenes
Using Moving Least Squares

August 2018

Graduate School of Science and Technology
Keio University

Siim Meerits

Abstract

Three-dimensional (3D) reconstruction is an actively researched area of computer vision. The general objective in this field is to construct digital models of real objects and scenes. With the advent of RGB-D cameras and increase in graphics card processing power, the focus of reconstruction has recently shifted from static to dynamic content and from offline to online systems. In the current state-of-the-art works many difficulties remain: multi-camera scene capture requires use of bespoke hardware, fast motions and big topology changes cannot be correctly handled and large scenes cannot be reconstructed due to memory constraints or too computationally expensive reconstruction methods. This thesis addresses all of these issues and proposes novel methods as solutions.

To avoid capturing scenes using custom and expensive hardware this thesis proposes capturing scenes for reconstruction using low-cost consumer-grade RGB-D cameras. As a drawback, the consumer devices lack capability of synchronizing camera shutters. This issue is mitigated algorithmically by developing a novel depth frame interpolation method that allows generating new temporally consistent depth data. As a byproduct, this method can be used to generate synthetic slow-motion 3D reconstruction videos.

Two novel real-time 3D reconstruction methods, ZipperMLS and FusionMLS, that allow reconstructing both highly dynamic and very large scenes are proposed. Both are based on moving least squares (MLS) surface estimation technique and produce triangle meshes as output. The methods differ considerably in the way geometry is handled. ZipperMLS reconstruction method belongs to the explicit surface reconstruction methods family. Triangle meshes are directly generated from depth maps and then merged. To smooth surfaces a new projection operator is contributed for MLS that is suitable for direct meshing of depth maps. To merge meshes, the concept of mesh zippering from previous work is re-engineered to work as highly parallelizable algorithm suitable for execution on GPUs. FusionMLS reconstruction method belongs to the volumetric reconstruction methods family. It uses MLS to estimate signed distance function (SDF) at each voxel location and marching cubes method to generate triangle mesh of the scene. The main

novelty of the method is packing surface estimation and mesh generation into a single process. That allows very low memory usage and fast reconstruction.

The developed 3D reconstruction methods can be used in various applications. A diminished reality system is demonstrated as a practical example. Diminished reality is part of mixed reality research field and its purpose is to hide, i.e. diminish, user selected objects from captured scenes. Diminishing complex objects can be demonstrated as the proposed reconstruction methods are very general.

Presented 3D reconstruction and application implementations are completely GPU-based and work in real time. The results shown in this thesis, obtained with real data, demonstrate the effectiveness of the proposed methods and its advantages compared to state-of-the-art works.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 6 |
| 1.1 | Overview of 3D reconstruction | 6 |
| 1.2 | Motivation | 12 |
| 1.3 | Thesis organization | 18 |
| 2 | Related works | 20 |
| 2.1 | Visibility methods | 20 |
| 2.2 | Volumetric methods | 21 |
| 2.3 | Indicator function methods | 24 |
| 2.4 | Point-based methods | 25 |
| 2.5 | Triangulation methods | 27 |
| 3 | Scene capture | 30 |
| 3.1 | RGB-D cameras | 30 |
| 3.2 | System topology | 34 |
| 3.3 | Camera calibration | 36 |
| 3.3.1 | RGB-D camera intrinsic and extrinsic parameters | 36 |
| 3.3.2 | Extrinsic calibration between RGB-D cameras | 38 |
| 3.4 | Time calibration | 39 |
| 4 | Motion estimation and correction | 41 |
| 4.1 | Scene flow estimation | 43 |
| 4.2 | Depth frame warping | 44 |

| | | |
|----------|---|-----------|
| 5 | 3D reconstruction | 46 |
| 5.1 | Normal estimation | 46 |
| 5.2 | Moving least squares | 50 |
| 5.2.1 | Signed distance field | 51 |
| 5.2.2 | Point projection | 52 |
| 5.3 | ZipperMLS reconstruction method | 52 |
| 5.3.1 | Overview | 53 |
| 5.3.2 | MLS projection method | 54 |
| 5.3.3 | Mesh generation | 57 |
| 5.3.4 | Performance | 63 |
| 5.4 | FusionMLS reconstruction method | 66 |
| 5.4.1 | Overview | 66 |
| 5.4.2 | Volume hierarchy | 67 |
| 5.4.3 | Block occupancy | 69 |
| 5.4.4 | Reconstruction process | 70 |
| 5.4.5 | Rendering | 72 |
| 5.4.6 | Performance | 73 |
| 5.5 | Results | 76 |
| 5.6 | Discussion | 83 |
| 6 | Applications | 86 |
| 6.1 | Overview | 86 |
| 6.2 | Diminished reality application | 87 |
| 6.2.1 | Object pose estimation | 88 |
| 6.2.2 | Rendering | 88 |
| 6.2.3 | Compositing | 90 |
| 6.2.4 | Results | 91 |
| 7 | Conclusions | 95 |

List of Figures

| | | |
|------|---|----|
| 1.1 | 3D reconstruction input and output | 10 |
| 2.1 | Volumetric 3D reconstruction examples | 22 |
| 2.2 | Indicator-function-based 3D reconstruction example | 24 |
| 2.3 | Point-based 3D reconstruction example | 26 |
| 2.4 | Triangulation-based 3D reconstruction example | 28 |
| 3.1 | Example of typical scene setup | 32 |
| 3.2 | Two major scene setup scenarios | 33 |
| 3.3 | Multi-camera capture system topology | 34 |
| 4.1 | Frame interpolation demonstration | 42 |
| 4.2 | Scene flow estimation steps | 43 |
| 4.3 | Depth frame interpolation example | 45 |
| 5.1 | Comparison of normal estimation methods | 49 |
| 5.2 | Overview of ZipperMLS reconstruction | 54 |
| 5.3 | Visualization of different surface projection methods | 56 |
| 5.4 | Comparison of MLS projection methods | 56 |
| 5.5 | Illustration of forming triangles between vertices | 58 |
| 5.6 | Illustration of mesh erosion | 59 |
| 5.7 | Illustration of mesh merging | 61 |
| 5.8 | Illustration of final mesh generation | 62 |
| 5.9 | Mesh merging example | 63 |
| 5.10 | Comparison of mesh merging methods | 64 |
| 5.11 | Overview of FusionMLS reconstruction | 67 |

| | | |
|------|--|----|
| 5.12 | Visualization of occupancy volume | 70 |
| 5.13 | Visualization of voxel block reconstruction | 74 |
| 5.14 | Visualization of edge rendering methods | 75 |
| 5.15 | Results of reconstructing highly dynamic scenes | 77 |
| 5.16 | Comparison of 3D reconstruction methods | 78 |
| 5.17 | Reconstruction results of human actors | 79 |
| 5.18 | Reconstruction results of humans with background | 80 |
| 5.19 | Example of reconstructing large scenes | 81 |
| 6.1 | Visualization of input images and region masks | 89 |
| 6.2 | Visualization of composition post-processing steps | 92 |
| 6.3 | Diminishing human actor | 93 |
| 6.4 | Diminishing spherical object | 94 |

List of Tables

| | | |
|-----|--|----|
| 1.1 | Comparison of recent 3D reconstruction methods | 13 |
| 3.1 | RGB-D camera specifications | 31 |
| 5.1 | ZipperMLS performance measurements | 64 |
| 5.2 | ZipperMLS recommended parameter values | 65 |
| 5.3 | FusionMLS performance measurements | 74 |
| 5.4 | FusionMLS recommended parameter values | 75 |

Chapter 1

Introduction

This chapter introduces main 3D reconstruction concepts and directions of development. The motivation of this thesis is explained by showing what features state-of-the-art methods have and what they have so far lacked. Finally the content of other chapters is outlined.

1.1 Overview of 3D reconstruction

3D reconstruction research is part of computer vision field and is also closely related to computer graphics research. The purpose of 3D reconstruction is to generate digital models of real-world scenes. Next paragraphs give a short overview of developments in the field. After that, 3D reconstruction methods are generalized and discussed as series of steps.

Brief history of the field. 3D reconstruction research is strongly connected to available scene capture methods. Simplest way to acquire information about surroundings is to use classical imaging sensors. This is the basis of stereo and multi-view stereo methods (MVS) that use, respectively, two or more cameras at different viewpoints to capture surroundings. Initial MVS methods were demonstrated in the mid 90s (Laurentini, 1994). Unfortunately, the problem of obtaining 3D models from 2D images is very complex. High number of images need to be captured from various viewpoints to get a good sense of object shapes. Even then, the problem is typically underdefined. Both the scene capture

and computational requirements mean that the methods are generally designed to work offline to reconstruct static scenes i.e. scenes without movement. The MVS methods are still actively developed (Seitz et al., 2006) and are yet to become real-time. Visually fairly high-quality results, however, have been well demonstrated.

To make 3D reconstruction task easier, range-sensing devices were developed that can capture so-called point clouds of surroundings. The points in these clouds represent surface locations sampled by sensors. The reconstruction task then becomes reconstructing models from points instead of images. This problem is also difficult due to various types of noise in point clouds, but still easier than purely image-based reconstruction. Initially, small-scale objects less than 1 meter in size were scanned with stationary and bulky devices. Famous examples include the the Stanford 3D Scanning Repository (Turk and Levoy, 1994) that became de-facto the main 3D reconstruction benchmarking dataset.

Armed with point cloud capture devices, many of the fundamental 3D reconstruction methods were developed from the end of 80s to mid 90s. For example, volumetric reconstruction (Hoppe et al., 1992; Curless and Levoy, 1996), triangle mesh generation from volumes (Lorensen and Cline, 1987; Doi and Koide, 1991) and direct meshing (Turk and Levoy, 1994; Hilton et al., 1996) methods. All of them worked offline due to lack of computational power.

For a long time, range or depth sensing devices were expensive and limited in their capabilities. However, around year 2010, with the development of Microsoft Kinect and other so-called RGB-D cameras, the technology became widely available. Here RGB-D refers to devices which include both standard color image sensor and depth sensing apparatus. This combination of sensors is popular due to application requirements. In terms of 3D reconstruction, it allows filtering point clouds using color info and allows coloring reconstructed scene models.

3D reconstruction requires processing huge volume of data. It is difficult to achieve real-time performance with purely CPU-based algorithms. From around year 2001, graphics processing units (GPUs) started to become more flexible, allowing user-written code to be executed instead of a fixed pipeline. Today, this is known as general-purpose computing on graphics processing units (GPGPU). It took many years for mature frameworks to appear. For example Nvidia CUDA was released in 2007, OpenGL version 3.0 was released

in 2008 and OpenCL was released in 2009. Currently, using GPU can provide orders of magnitude faster 3D reconstruction processing than CPU in case of algorithms that are well parallelizable.

The change in both RGB-D camera and GPGPU hardware availability spurred a revision of earlier reconstruction work. Popular methods of the 80s and 90s were adapted to GPUs and could now handle RGB-D camera specific noise. Systems such as KinectFusion (Izadi et al., 2011) allowed capturing surroundings in real-time. This method and most other methods at the time were strictly aimed at static scene reconstruction. Lot of effort was put into making the reconstruction scalable to large scenes. Typically a scene is scanned using a single device incrementally. This can result in camera position tracking errors (otherwise known as drift) to accumulate, leading to corrupted models. The research into solutions, such as loop closure, is still continuing.

After real-time high-quality static scene reconstruction was demonstrated around 2011, the research focus started to shift towards dynamic scene reconstruction. Initially only small movements in the scene were allowed (Keller et al., 2013). Arguably, only from 2015 did more general systems appear that allowed arbitrary movements (Dou et al., 2016). Most of the methods build upon technologies developed for static scene reconstruction. As a fairly recent development, lot of issues remain to be solved.

Applications and use cases. Early 3D reconstruction applications were in field of medicine. For example, computer tomography scanned data had to be reconstructed (Lorensen and Cline, 1987) to visualize results. Some other areas that have required static scene reconstruction include archeology, earth observation and reverse engineering. In archeology (Oliveira et al., 2014) excavated sights can be recorded so that object positions are preserved exactly. Scanned artifacts can also be further compared and analyzed digitally. In earth observation the task is to reconstruct terrain shape, buildings and other objects from aerial photos, satellite images or terrain laser scans.

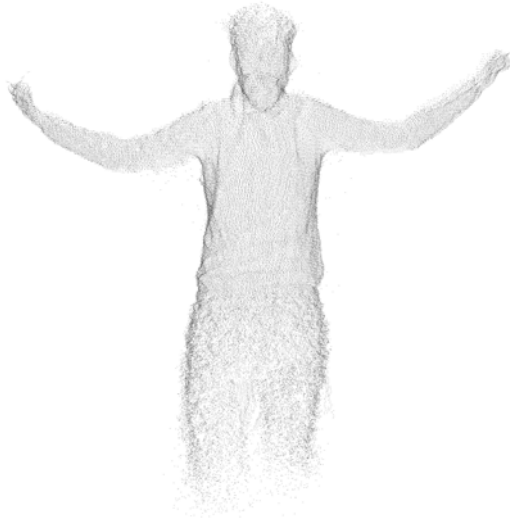
Arguably, more interesting applications need reconstruction in dynamic scenes. Some examples include cinematography, robotics, telepresence or mixed reality. In cinematography (Ronfard and Taubin, 2010) the aim of 3D reconstruction is to help with special effects. For example, actors can be scanned and digitized. Filmed scenes have to be

reconstructed to some degree to add realistic looking special effects which fit with scene lighting or obey other physics rules. Robotics (Kostavelis and Gasteratos, 2015) typically uses reconstruction to create a map of visited areas or uses already reconstructed models for determining robot position. Telepresence, in general, tries to immerse users in some other place than their real location. By far the most common use case is videoconferencing (Maimone and Fuchs, 2011; Collet et al., 2015; Plüss et al., 2016) where the objective is to allow humans to communicate over telecommunication systems. Here reconstruction helps to immerse users in a 3D scene. Finally, in mixed reality field (Orts-Escolano et al., 2016; Lindlbauer and Wilson, 2018), the objectives of 3D reconstruction can vary wildly. In many cases the surroundings need to be reconstructed so that new virtual objects can be placed into the scene correctly.

Reconstruction stages. For most 3D reconstruction systems, the process can be divided conceptually into following stages:

1. **Data acquisition.** Data about surroundings needs to be captured using some devices. In case of multi-view stereo methods, color image sensors are used. Recent systems almost exclusively rely on cameras that produce point clouds of surroundings.
2. **Surface reconstruction.** The surface reconstruction consolidates available 3D information to a single consistent surface.
3. **Geometry extraction.** After a surface has been defined, it should be converted to a geometric representation that is useful for a particular application. Some commonly used formats are point clouds, triangle meshes and depth maps.
4. **Application layer.** In most applications, obtained geometry is rendered and presented to user in some form. In mixed reality field, synthetic content may be added to the scenes.

A wide variety of methods have been developed to capture raw data about scene surroundings. The most relevant for real-time 3D reconstruction are RGB-D devices that can sample point clouds of scenes. In terms of underlying point cloud capture technology, there are two major branches of range sensing methods – triangulation and Time-of-Flight



(a) Point cloud



(b) Triangle mesh



(c) Shaded model



(d) Textured model

Figure 1.1: Example 3D reconstruction inputs and outputs: (a) point cloud captured with RGB-D sensors, (b) triangle mesh of reconstructed model, (c) shaded model of the triangle mesh, and (d) model with textures from RGB cameras.

(ToF). The triangulation methods obtain points by triangulating object positions from multiple viewpoints. For example, stereo devices contain two image sensors at fixed positions to capture images of scene from slightly different viewpoints. Corresponding points are found in both images which allows triangulating point positions. Finding correspondences can be difficult task, especially when objects lack texture. For this reason, structured light technologies were developed which illuminate scene with known light pattern. This pattern can be triangulated directly or be used to provide more texture for textureless objects. ToF technology is based on fairly different concept. While there are multiple sub-categories of ToF technology, the general idea is that camera devices emit light which is reflected from objects and then captured again at source. By measuring the time between light emission and capture the object distance can be calculated. It is possible to directly measure the flight time, but this requires very complex electronics due to the high speed of light. Most devices emit light as a continuous amplitude modulated wave. The distance from an object can then be derived from a phase shift of emitted and captured signals.

Surface reconstruction determines positions of surfaces in the scene. Typically, the input to this process consists of point clouds from capture system. Points represent locations of the surfaces, but tend to contain errors due to noise. In multi-camera capture systems the point clouds from different cameras might not perfectly align. Reconstruction methods have to be able to consolidate the separated clouds into a single surface yet detect when two surfaces are simply close to each other. Noise has to be removed without oversmoothing. Holes in models should be filled, but care has to be taken not to introduce incorrect surfaces. It can be seen that surface reconstruction has many conflicting goals and balance has to be found in many issues to achieve high-quality surfaces.

Geometry extraction deals with turning reconstructed surfaces into some standard representation useful for applications. Some typical options are: point clouds, volumes or triangle meshes. All of them have benefits and drawbacks. Point clouds are easy to manipulate, but do not make up a watertight surface of objects. Additionally, rendering requires splatting techniques (Zwicker et al., 2001) which is not very well suited for GPUs. Volumes are another possibility which are natural choice for many reconstruction methods. Unfortunately the memory cost of storing volumes is considerable and may

become too high if scenes are large. Additionally rendering such scenes requires raycasting techniques which again have non-trivial computational cost. Triangle meshes have the lowest rendering cost compared to other methods and have fairly low memory footprint.

Some stages of 3D reconstruction are visualized in Fig. 1.1. The point cloud is the result of data acquisition stage. Reconstruction process itself is hard to visualize, but results of geometry extraction can be shown. Both triangle mesh and its shaded version are presented. For applications that need visualizing reconstruction results it is possible to texture the reconstructed model with RGB camera images.

The work in this dissertation involves all layers of the 3D reconstruction with contributions in all parts. Arguably, the most complex and important parts are surface reconstruction and geometry extraction steps. For this two novel 3D reconstruction methods called ZipperMLS and FusionMLS were developed.

1.2 Motivation

3D reconstruction methods have many aspects that can be compared. In the following, important capabilities of reconstruction systems are summarized to see the motivation behind this work.

- **Dynamic content is allowed.** Scenes that are dynamic contain moving objects. This makes reconstruction considerably more difficult as it is not possible to accumulate data over multiple frames without tracking movements. Alternatively scenes have to be reconstructed using single frame of information.
- **Topology changes are allowed.** While dynamic scene reconstruction methods can handle movements, there may be many restrictions on what movements are allowed. Hence it is reasonable to separately consider which systems are general enough to allow any kinds of movements including topology changes. A topology change means change in shape that cannot be achieved with just deformation of a model. For example an object is split into two parts.
- **Entering-exiting is allowed.** Many methods can track deformations and even larger topological changes. However, there might be no capability of starting tracking

Table 1.1: Comparison of recent 3D reconstruction methods

| | <i>Dynamic content</i> | <i>Topology changes</i> | <i>Entering-exiting</i> | <i>No priors</i> | <i>Any shapes</i> | <i>Backgrounds</i> | <i>Large scenes</i> | <i>Mutli-camera</i> | <i>Commodity</i> | <i>Real-time</i> | <i>Triangle meshes</i> |
|-------------------------|------------------------|-------------------------|-------------------------|------------------|-------------------|--------------------|---------------------|---------------------|------------------|------------------|------------------------|
| Keller et al. (2013) | ○ | | ● | ● | ● | ● | | ● | ● | | |
| Zollhöfer et al. (2014) | ● | | | ● | | | | | ● | | ● |
| Newcombe et al. (2015) | ● | | | ● | | | | ● | ● | | ● |
| Innmann et al. (2016) | ● | | | ● | | | | ● | ● | | ● |
| Wang et al. (2017) | ● | ○ | ● | ● | | ○ | | ● | | | ● |
| Guo et al. (2017) | ● | | ● | ● | | | | ● | ● | | ● |
| Yu et al. (2017) | ● | | | | | | | ● | ● | | ● |
| Li et al. (2018) | ● | | | | | ○ | | ● | | | ● |
| Dou et al. (2016) | ● | ● | ○ | ● | ● | | ● | | ● | | ● |
| Dou et al. (2017) | ● | ● | ○ | ● | ● | | ● | | ● | | ● |
| Collet et al. (2015) | ● | ● | ● | ● | ● | ○ | ● | | ○ | | ● |
| Alexiadis et al. (2017) | ● | | | | | | ● | ● | ● | | ● |
| Alexiadis et al. (2018) | ● | | | | | | ● | ● | ● | | ● |
| Kuster et al. (2014) | ● | ● | ● | ● | | ● | ○ | ● | ○ | | |
| Plüss et al. (2016) | ● | ● | ● | ● | | | | ● | ● | | ● |
| This work | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |

● - demonstrated in prior work

○ - has limitations or not fully demonstrated

of new objects as they enter scenes or the tracking method may get confused when objects leave scenes.

- **No priors are needed.** Reconstruction of scenes becomes simpler if the shapes of objects are known a priori. The objects may be pre-scanned or hand-designed models. This method is particularly helpful for generating high-quality results if camera coverage of the scene is limited. The clear drawback of this approach is that the scene objects have to be known beforehand.
- **Any shapes are acceptable.** In the presence of sensor noise and lack of camera coverage, reconstructing arbitrary objects may be difficult. Hence by restricting the types of objects allowed to be reconstructed, higher quality output may be achieved. Most common restriction is to allow only human actors in the scene. However, this severely limits applications where the reconstruction can be used.
- **Backgrounds can be reconstructed.** Many reconstruction methods are incapable of reconstructing scene backgrounds. This limitation typically arises from the technical choices of reconstruction algorithms. For example, the so-called visibility methods need to extract a silhouette of objects. Background objects either lack silhouette or it is difficult to extract them. Again, many practical application might need the backgrounds in results.
- **Large scenes can be reconstructed.** Development of static scene reconstruction has moved in the direction of allowing bigger reconstruction areas. Today, models of whole buildings can be generated by scanning the rooms one-by-one. Typical dynamic scene reconstruction methods, however, are currently limited to couple cubic meters of space. Generally, volumes of 50m^3 or more are considered to be large in this work.
- **Multi-camera input is possible.** To obtain good coverage of scene, multiple cameras could be used. This, however, comes with a set of issues that need to be solved such as calibration of camera poses, synchronization of shutters and fusing multiple sources of data.

- **Commodity devices can be used.** Many systems require the captured input data to have certain properties. For this reason it might be necessary to use custom-built capture devices to satisfy those requirements. However, consumer-grade RGB-D cameras are preferred for their price and ease of use.
- **Real-time execution.** Many applications need to use reconstruction results immediately. For example a free-viewpoint video broadcast or a mixed reality application. In those scenarios the scene should be reconstructed at 30 frames per second or more.
- **Triangle meshes produced as output.** The reconstruction result can take many different forms. Popular choices are point clouds, depth maps or triangle meshes. Point clouds require splatting techniques to render and depth maps are view dependent. Triangle meshes are the most traditional representation allowing watertight model rendering. Additionally meshes are very well supported by graphics hardware.

This list was compiled on the basis of features compared in related works. The objective was to make a comprehensive list of capabilities of reconstruction methods in terms of input properties, ability to handle various types of scenes and output data features. Comparison of reconstruction methods can also be found in two recent survey papers. Berger et al. (2017) compare methods on the basis of input point cloud artifacts and requirements. Recent dynamic scene reconstruction work, however, almost exclusively relies on RGB-D devices that have similar input point cloud characteristics. Zollhöfer et al. (2018) mostly compare methods in terms of technologies used. This can be of interest to researchers to see latest trends in 3D reconstruction. However, it does not give a good picture of the capabilities of reconstruction methods.

An overview of state-of-the art reconstruction systems in terms of their features can be seen in Tab. 1.1. While all the methods allow at least some movement in scene, very few methods can allow arbitrary topological changes or free entering and exiting of scenes. At least half of the methods specialize in some types of objects such as humans, whether using scanned prior or a generic model. While many of the newest works use multiple cameras, the area of reconstruction is still small. In fact, most systems can only reconstruct few

isolated foreground objects. The need for real-time performance is well understood and only few systems using complex tracking or reconstruction methods work offline. With the proliferation of volumetric reconstruction methods, recent works almost exclusively produce triangle meshes.

The proposed reconstruction methods were designed to satisfy all the previously mentioned features. This requires some rethinking of the way to approach 3D reconstruction. In the following paragraphs some crucial observations about prior art are discussed together with ideas that allow overcoming the limitations of state-of-the-art works.

Scene capture. To obtain good 3D reconstruction results, scenes should be captured from multiple viewpoints. Unfortunately, this has typically required use of custom-made capture devices that can be explicitly synchronized. In turn this has led to only few recent systems to utilize multiple cameras. This work proposes using much more widely available consumer-grade RGB-D devices for scene capture. These cameras do not have a way to synchronize shutters. This issue is solved algorithmically. To obtain temporally consistent data, both time calibration and depth map motion compensation techniques are developed.

Data accumulation. Most prevalent method of reconstructing scenes is to continuously accumulate received camera data to a global 3D model. This has been very successful when dealing with static scenes. The sensor noise can be eliminated by averaging measurements over multiple frames. Problems arise when dealing with dynamic scenes. The scene movements have to be tracked with high precision so that data would be correctly accumulated. In fact, due to the limited camera visibility of scenes, sensor noise, and unlimited number of possible scene geometry changes, it is unlikely that completely accurate scene tracking can be achieved. The solution proposed by Dou et al. (2016) in their seminal Fusion4D reconstruction system is to reset accumulated data periodically. This approach can achieve truly high quality results, but even in this case very fast movements can corrupt reconstructed models. This work proposes going even further by reconstructing full scene geometry from scratch on each frame. This allows higher degree of dynamic movements without the danger of corrupting reconstructed model.

Reconstructing full scene on each frame might seem expensive, but counterintuitively this proves to be an advantage. The main benefit is that there is no need to store reconstruction data for use in next frames. This results in both lower memory cost and reduced memory bandwidth which increases processing speed. Execution time is also reduced by not having to track object motion.

No priors and any shapes. Not accumulating data over multiple frames means that handling sensor noise simply by averaging measurements over time is no longer possible. Consequently, another way of consolidating and smoothing data from RGB-D cameras is needed. Many methods in the literature use templates. By detecting pose of known objects, the input data can be replaced with high-quality models known a priori. Some methods go further and restrict reconstruction to only human actors. While the quality of results can be high in these cases, many applications require more general approach that allows any objects in reconstruction.

This work takes a different, direct approach, and proposes using moving least squares methods to smooth and consolidate RGB-D data. This method considerably increases computational cost, but it is not a problem due to memory and execution time savings detailed in the previous data accumulation paragraph.

Geometry representation. As noted by Zollhöfer et al. (2018) in a recent summary of 3D reconstruction methods, almost all state-of-the-art works use truncated signed distance function (TSDF) volumes for storing scene geometry. This method has notoriously high memory footprint. Without memory usage reduction strategies it does not allow reconstructing large scenes. The only method capable of reconstructing very large scenes in Tab. 1.1 by Keller et al. (2013) uses point-based geometry that is very compact. Following this lead, the methods proposed in this dissertation also avoid storing TSDF volumes. Instead, the geometry is mostly represented as triangle meshes or point clouds.

Reconstruction method features in this work. In summary, the developed 3D reconstruction methods have following properties. Any type of movements including topology changes and objects entering-exiting scenes can be handled by reconstructing full scenes on each frame. The methods do not need any prior information and can reconstruct

arbitrary shapes due to the use of moving least squares based smoothing techniques. This already implies that scene backgrounds can also be reconstructed. Multiple cameras are used to increase scene coverage such that large scenes can be reconstructed. The result of reconstruction are triangle meshes as in most previous work. Finally, the proposed reconstruction methods work in real time so that they could be used in as wide variety of applications as possible.

1.3 Thesis organization

Rest of the thesis is divided into chapters as follows.

- Chapter 2 gives an overview of related 3D reconstruction literature. The focus is on works capable of real-time operation, that can produce triangle meshes as output and can handle dynamic scenes. Additionally, an overview of major 3D reconstruction applications is given.
- Chapter 3 details how real-life scenes are captured using RGB-D devices. The topology of cameras and computers is explained together with information on how compressed image data is transferred over network using custom protocols. Both temporal and spatial calibration methods are discussed including a novel way of calibrating consumer-grade RGB-D camera clocks.
- Chapter 4 deals with estimating non-rigid motion of objects for the purpose of compensating unsynchronized shutters of multiple cameras. The process of estimating scene flow from consecutive depth frames is detailed. Finally, interpolation method that utilizes calculated scene flow is described.
- Chapter 5 starts from detailing processes necessary for successful 3D reconstruction such as initial surface normal estimation. In normal estimation two methods suitable for MLS-based reconstruction are given with tradeoffs in speed and quality. Next, details of generic MLS surface estimation are given together with two specific methods called ZipperMLS and FusionMLS. Finally, comparison of proposed reconstruction methods is given using real data in various scenes.

- Chapter 6 discusses applications of the proposed 3D reconstruction methods. A diminished reality system is presented in detail, leveraging the unique capabilities of the reconstruction approach.
- Chapter 7 concludes the thesis with an overview of the major contributions and methods used.

Chapter 2

Related works

The related literature overview is constrained to 3D reconstruction methods that can work with range image data from RGB-D sensors and provide explicit surface geometry outputs, such as point clouds or triangle meshes. As such, view-dependent methods are not discussed. For an overview of 3D reconstruction algorithms in general please refer to Berger et al. (2017). Reconstruction methods using RGB-D cameras are surveyed by Zollhöfer et al. (2018).

2.1 Visibility methods

Visual hull methods, as introduced in Laurentini (1994), reconstruct models using an intersection of object silhouettes from multiple viewpoints. Polyhedral geometry (Matusik et al., 2001; Franco and Boyer, 2003) has become a popular representation of hull structure. To speed up the process, Li et al. (2002); Duckworth and Roberts (2014) developed GPU-accelerated reconstruction methods. The general drawback with those approaches is that they need to extract objects from an image background using silhouettes. In a cluttered and open scene, this is difficult to do. In many applications it might also be necessary to include backgrounds in reconstructions, but this is generally not supported.

Curless and Levoy (1996) uses the visual hull concept together with range images and defines a space carving method. Essentially, an object is carved out of volume with the help of depth maps recorded from multiple viewpoints. Thanks to the range images, background segmentation becomes a simple task compared to visual hull methods. Original Curless

and Levoy method was used to reconstruct small-scale static objects. Saito et al. (2003) show reconstruction of room-sized scenes with the use of multibaseline stereo. Dynamic scene reconstruction is made possible by independently reconstructing consecutive frames captured from cameras. Unfortunately, the system requires high number of cameras to be available and works offline. Yaguchi and Saito (2004) demonstrate reconstruction in very large scenes using again shape-from-silhouette info, but reconstruct objects in projective grid space. This approach allows developing free-viewpoint video applications, but the underlying reconstructed meshes have limited quality.

2.2 Volumetric methods

Volumetric reconstruction methods represent 3D data as grids of voxels. Each volume element can contain binary space occupancy data (Connolly, 1984; Chien et al., 1988) or samples of some continuous function that describes geometry in the vicinity of the voxel (Hoppe et al., 1992; Curless and Levoy, 1996). Nowadays almost all volumetric methods use either signed distance function (SDF) or truncated signed distance function (TSDF) for this purpose. In both cases the value is essentially a distance to the nearest surface and the sign of the value represents whether the voxel is located outside or inside solid objects. TSDF differs from SDF by truncating distances at some pre-determined minimum and maximum value.

One of the earliest examples of using the volumetric reconstruction approach in room-sized scenes was presented by Narayanan et al. (1998) who used high number of RGB cameras to capture objects from multiple viewpoints. Multibaseline stereo was used to generate depth maps of the scene which could then be used in volumetric reconstruction to generate triangle meshes. Additionally, this method also demonstrated direct meshing of depth maps to achieve free viewpoint video.

After commodity RGB-D sensors became available, the work of Izadi et al. (2011) spawned a whole family of 3D reconstruction algorithms based on TSDF volumes. In its core, the method uses Curless and Levoy (1996) work to capture static scenes, but accumulates data in real time using GPU based acceleration. The strength of these methods is their ability to integrate noisy input data to produce high-quality scene models.

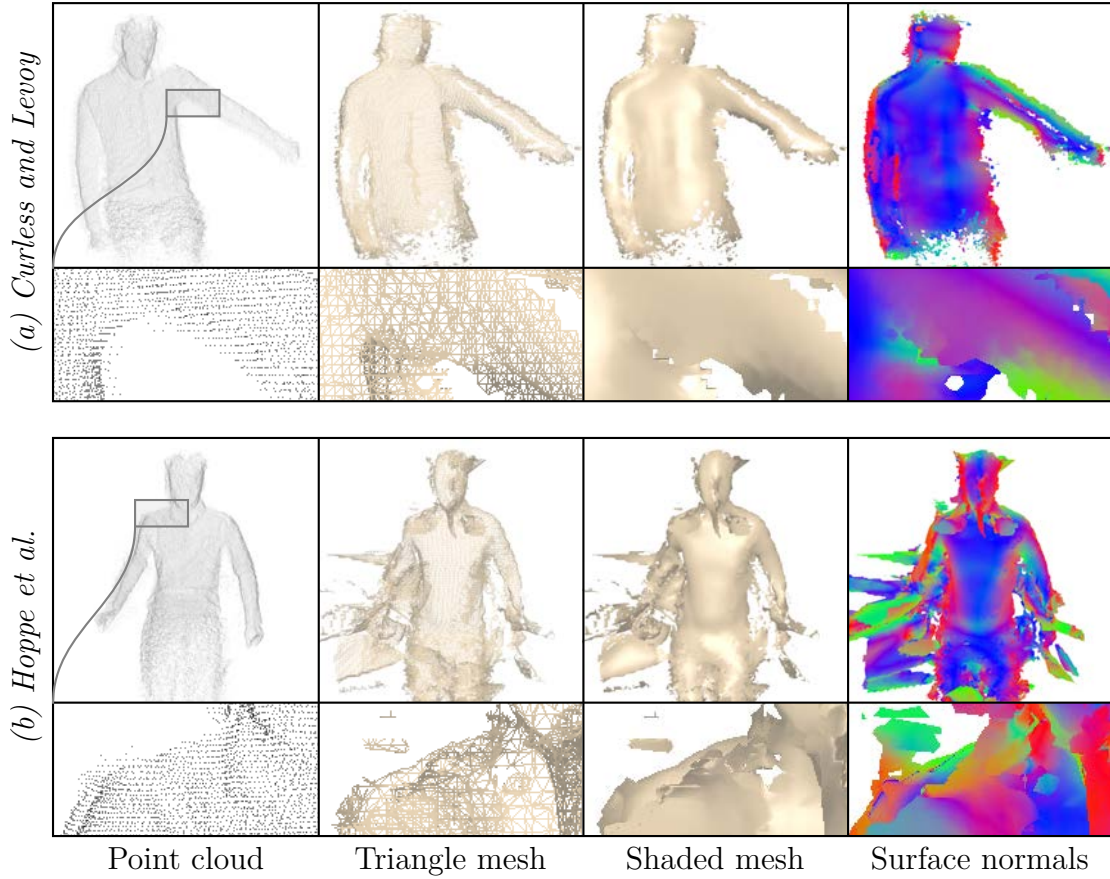


Figure 2.1: Examples of two classic volumetric 3D reconstruction methods: (a) Curless and Levoy (1996), and (b) Hoppe et al. (1992). Both methods have noisy results. In case of (a) there are not enough points in a single frame of data to obtain smooth surfaces and in case of (b) the method cannot properly handle outliers and noise in general.

However, a major drawback is their high memory consumption. Whelan et al. (2012); Chen et al. (2013) propose out-of-core approaches where reconstruction volume is moved around in space to lower system memory requirements. Nießner et al. (2013) provide an alternative solution that compresses volume space using spatial hashing scheme. While boasting good performance, the method increases the complexity of reconstruction pipeline considerably.

Natively, TSDF methods such as KinectFusion by Izadi et al. (2011) cannot capture dynamic scenes. One way to handle dynamic content is to segment scene to dynamic and static parts. For example Dou and Fuchs (2014) propose using a pre-scanned static backgrounds or more recently the Co-Fusion approach by Rünz and Agapito (2017) allow

both multiple moving objects and online capture of scene backgrounds.

Using TSDF volumes for dynamic scene reconstruction is considerably more complicated than in static scene situation. The voxel data has to be relocated when objects move. This requires very accurate tracking of object movements. Otherwise this quickly leads to corrupted scene geometry. Newcombe et al. (2015) introduced DynamicFusion, wherein depth data is accumulated into a canonical model of a scene, which is subsequently deformed using a warp field to match scene changes in real time. Innmann et al. (2016) improved DynamicFusion by estimating a more dense warping field, and Guo et al. (2017) made use of albedo information in motion tracking. Zhang and Xu (2018) added an option to reconstruct scene backgrounds by segmenting dynamic and static content. While the reconstruction quality is high, these methods can fail when scene topology changes considerably from that of the initially estimated model. Moreover, these methods were designed to be used with a single RGB-D camera. Dou et al. (2016) proposed resetting the deformed model periodically to allow more extensive scene topology changes. The method relies on multiple custom-built capture devices. Follow-up work (Dou et al., 2017) has improved the reconstruction and motion estimation accuracy via machine learning techniques but retains a similar hardware setup.

Fusion-based methods track the motion of objects in the scene over time and accumulate the captured data into a canonical representation of the scene. Following this strategy, parametric reconstruction methods display impressive results by deforming models of objects known a priori. Performance capture systems track motion by deforming a model of a human Yu et al. (2017); Li et al. (2018). Zollhöfer et al. Zollhöfer et al. (2014) present a way to scan any 3D object template, which can then be deformed in real time. The obvious drawback of fixed templates and parameterized models is their inability to deal with unexpected scene topology changes or appearance of unknown objects in the scene. Wang et al. (2017) propose templateless reconstruction method that can efficiently track non-rigid motions. However, the method does not work in real time and complex topology changes may not correctly be tracked.

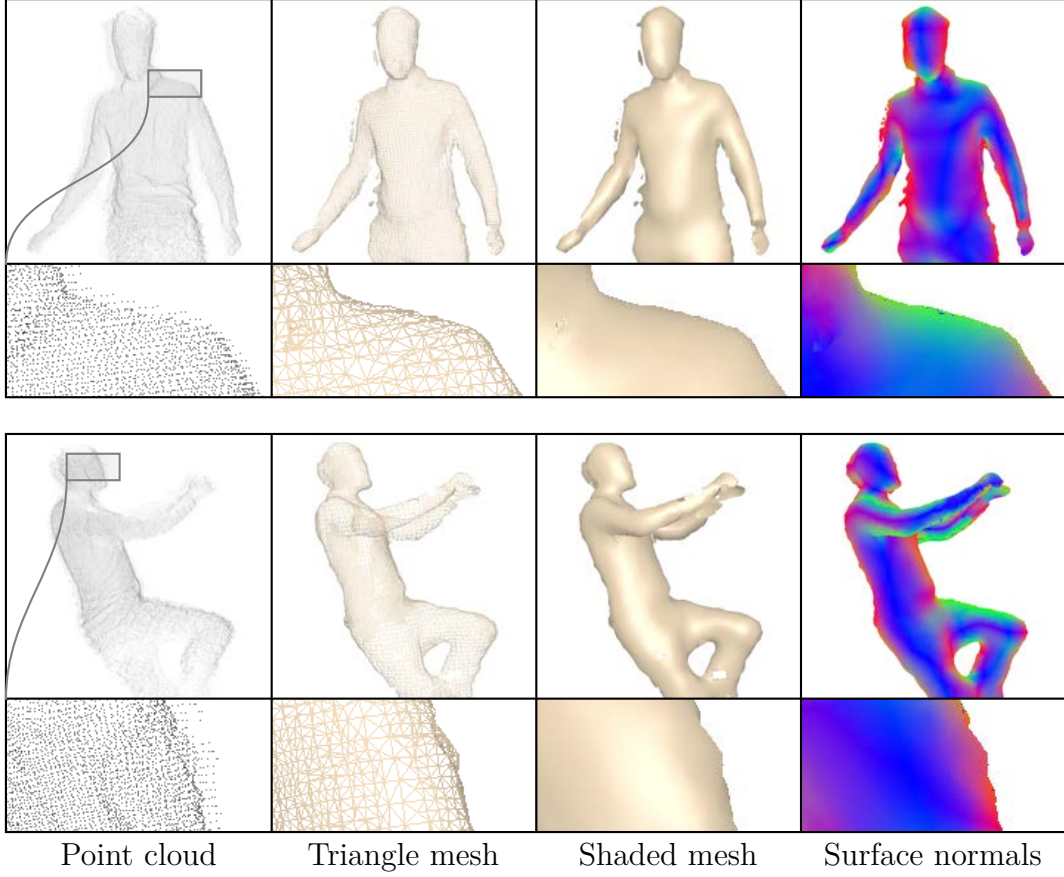


Figure 2.2: Example of indicator-function-based Poisson reconstruction (Kazhdan et al., 2006). The reconstruction quality is fairly high but some erroneous surfaces appear due to point cloud noise.

2.3 Indicator function methods

The goal of indicator function methods is to label all parts of reconstruction space either as interior or exterior of objects. It is difficult to directly label interior-exterior space from point clouds. However, due to the way RGB-D cameras and most other range scanning techniques work, rough surface normals for each point in point cloud can be estimated. Crucially, there exists a connection between point cloud and indicator function. Namely, the gradient of the indicator function should match point cloud normals. That problem can be stated in the form of Poisson partial differential equation. After solving the differential equation it is possible to generate a 3D model by locating the boundary between interior and exterior space of the indicator function.

The indicator function can be found through various means. Kazhdan (2005) first proposed Fourier transform based method which uses traditional 3D volume to store frequency domain data of indicator function gradient. Alexiadis et al. (2017) employ this method to reconstruct human actors in dynamic scenes. The Poisson equation can also be solved using multigrid approach Kazhdan et al. (2006). To solve some over-smoothing issues Kazhdan and Hoppe (2013) propose an enhanced version called screened Poisson reconstruction. Collet et al. (2015) used this method with some modifications to reconstruct various dynamic scenes, albeit utilizing an incredible amount of computational power and high number of cameras. Indeed, the global nature of the optimization comes with a great computational cost, making it infeasible in most situations with consumer-grade hardware. Wang et al. (2016) proposed another Poisson-based reconstruction system for dynamic scenes that utilizes a much simpler camera setup. Unfortunately this method fails under rapid motions.

Fig. 2.2 shows Poisson reconstruction Kazhdan et al. (2006) examples. While slow to compute, the results of human reconstruction have fairly high quality. In fact, the general benefit of indicator function methods is that they can create watertight models of objects even when some parts of point clouds are missing or noisy. From the negative side, parts of the scene that cannot be enclosed, such as scene backgrounds, may result in additional incorrect surfaces. This is due to indicator function trying to form closed surfaces. Additionally, temporal stability issues can arise when some parts of models lack point cloud coverage. The missing model areas can be interpolated in many ways from neighboring surfaces. Due to the presence of noise this means that such areas can vibrate or flicker in time.

2.4 Point-based methods

The general idea behind point-based methods is to represent surfaces only using point clouds as proposed by Pfister et al. (2000). This means that the reconstruction results are not continuous surfaces, but disjoint points without connectivity information.

Keller et al. (2013); Whelan et al. (2016) propose reconstruction systems using only point primitives. In this method, points are stored in a single large buffer without spatial

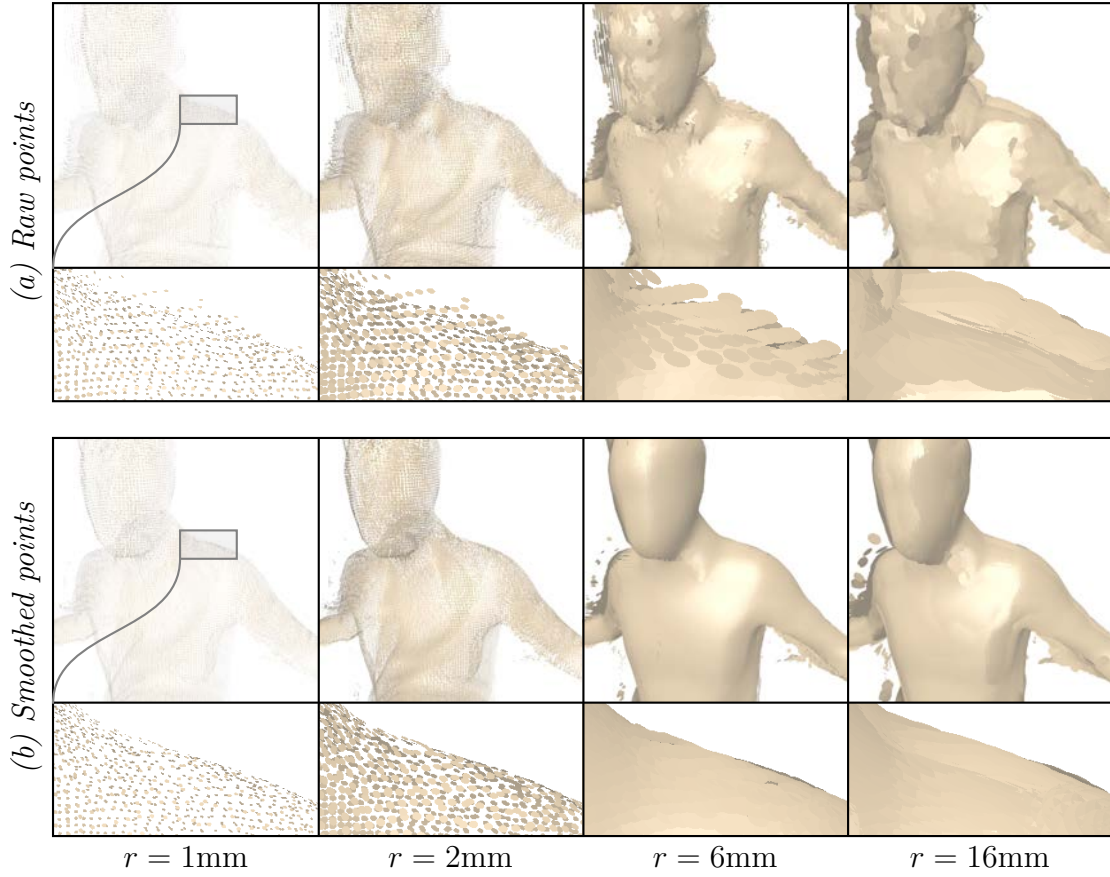


Figure 2.3: Example of point-based 3D reconstruction. All points are rendered as circular planar surfaces called splats. Four different splat radiuses, 1mm, 2mm, 6mm and 16mm, are shown for two point clouds: (a) unfiltered point cloud from RGB-D cameras, and (b) point cloud smoothed using MLS technique.

hierarchy. On each frame, points from depth map are added to the buffer. Points added to the model are initially categorized as unstable. When the points have been seen over multiple frames they will be promoted to stable category. This acts to reduce noise in the model. Additionally, the point positions can be averaged over local neighborhood to achieve smoother surfaces. Storing surfaces as points allows compact memory usage and therefore very large scenes can be reconstructed. Unfortunately, this method has very limited support for dynamic scenes. Additionally, the point clouds can become unnecessarily dense in some areas and still waste memory.

MLS methods have a long history in data science as a tool for smoothing noisy data. Alexa et al. (2001) used this concept in computer visualization to define point set surfaces

(PSS). These surfaces are implicitly defined and allow points to be refined by reprojecting to them. Since then, a wide variety of methods based on PSS have appeared – see Cheng et al. (2008) for a partial summary. In the classical formulation, Levin (2004); Alexa et al. (2003) approximate local surfaces around a point as a low-degree polynomial. Alexa and Adamson (2004) simplify the approach by formulating a signed distance field of the scene from oriented normals. To increase result stability, Guennebaud and Gross (2007) formulate higher-order surface approximation while Fleishman et al. (2005); Wang et al. (2013) add detail-preserving MLS methods. Kuster et al. (2014) introduce temporally stable MLS for use in dynamic scenes.

Most works to date utilize splatting (Zwicker et al., 2001) for visualizing MLS point clouds. While fast, this approach cannot easily handle texturing without blurring, so it is not as well supported in computer graphics as traditional triangle meshes are. It has been considered difficult to generate meshes on top of MLS processed point clouds. Regarding MLS, Berger et al. (2017) note that “it is nontrivial to explicitly construct a continuous representation, for instance an implicit function or a triangle mesh”. Scheidegger et al. (2005) and Schreiner et al. (2006) propose advancing front methods to generate triangles on the basis of MLS point clouds. These methods can achieve good results, but they come with high computational costs and are hard to parallelize. Plüss et al. (2016) directly generate triangle meshes on top of refined points. However, the result generates multiple disjointed meshes and does not deal with mesh stability.

Fig. 2.3 shows classical point-based MLS reconstruction and splatting-based rendering techniques in action. The radius of the splats has considerable impact on the quality of rendering and should be selected appropriately. As evident from the figure, points also need to be resampled or filtered to avoid very noisy rendering results.

2.5 Triangulation methods

Point clouds can be meshed directly using Delaunay triangulation and its variations (Cazals and Giesen, 2006). These approaches are subject to noise and uneven distances between points. Amenta and Bern (1999) and Amenta et al. (2001) propose robust variants with the drawback of being slow to compute.

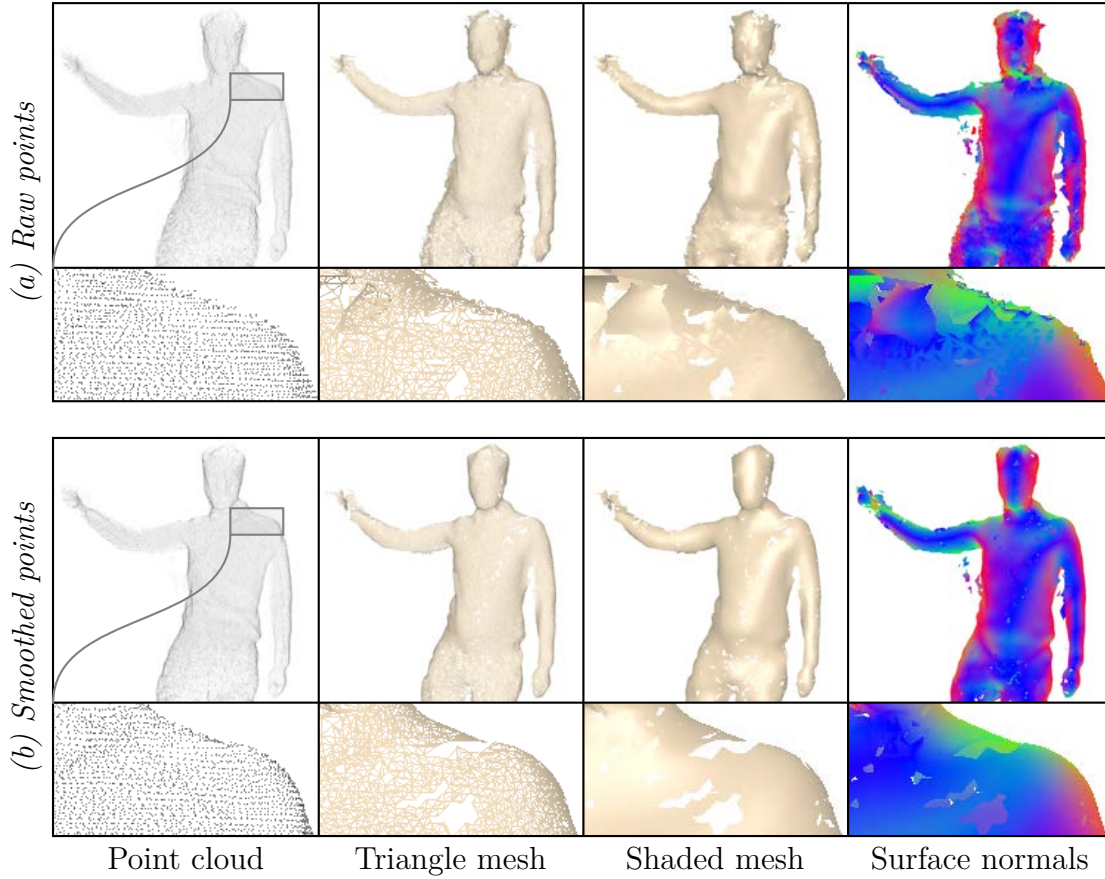


Figure 2.4: Example of triangulation-based 3D reconstruction (Marton et al., 2009) of two point clouds: (a) unfiltered point cloud from RGB-D cameras, and (b) point cloud smoothed using MLS technique. The number of triangles is very high and the meshes contain errors such as missing areas even when using smoothed point clouds.

Direct triangulation of input point clouds can result in very dense meshes. Digne et al. (2014) propose a method to iteratively simplify meshes initially created using Delaunay triangulation without losing sharp features. The method is robust to outliers and noise but is very expensive to compute. Another approach to control mesh density and achieve robustness to noise is to build meshes incrementally as proposed by Marton et al. (2009); Scheidegger et al. (2005); Schreiner et al. (2006). Also in this scenario, parallelization of mesh building task is difficult and results in limited performance.

Due to the computational cost of Delaunay triangulation and problems parallelizing mesh building the number of real-time and online methods is limited. Most systems make use of the inherent structure of point clouds captured using RGB-D devices. For

example, Maimone and Fuchs (2011) create triangle meshes for multiple cameras separately by connecting neighboring range image pixels to form triangles. The meshes rendered separately and then resulting renderings are merged using image manipulation. Alexiadis et al. (2013) take the idea further and merge triangle meshes before rendering. While those methods can achieve high frame rates, the output quality could be improved due to lack of smoothing or consolidation of data from different cameras.

Fig. 2.4 shows triangulation of point clouds using method proposed by Marton et al. (2009). In this work meshes are grown by starting from a random point and joining together neighboring points to form triangles. This method is shown in action both with and without pre-smoothing input point clouds. Typically the number of triangles is very high due to the high density of input clouds. It can be seen that triangulation methods are sensitive to noise and also tend to fail in regions with complex geometry.

Chapter 3

Scene capture

This chapter details how raw data about real-world scenes can be captured using RGB-D cameras. Multi-camera capture presents several difficulties: devices have to be connected to form a single system, huge amount of captured data has to be effectively transmitted and received in real time and both temporal and spatial calibration of cameras is needed for 3D reconstruction. These issues are discussed in detail in the following sections.


3.1 RGB-D cameras

The geometric information about scenes can be captured in various ways. Classically, multi-view stereo reconstruction methods (Seitz et al., 2006) have only used photometric and colorimetric info from traditional cameras. Despite the advances in algorithms and computational capabilities, the methods are still too expensive to work in real time. Hence it is necessary to use special capture devices that can reduce the computational load by giving at least some estimate of scene geometry directly.

To obtain geometric details about surroundings in real time, RGB-D devices can be used. Those devices typically consist of a standard color camera, representing the ‘RGB’ part of RGB-D, and some apparatus to capture a range image or so-called depth map of a scene, which represents the letter ‘D’ in the acronym. A number of different technologies exist to generate depth maps with each having its own performance characteristics. Most importantly, types of noise and their magnitude can vary wildly between approaches.

The focus is on using consumer-grade off-the-shelf RGB-D devices for scene capture.

Table 3.1: RGB-D camera specifications

| | | |
|---|---------------------------|---------------------------|
|  | Camera | Microsoft Kinect 2 |
| | Color resolution | 1980×1080 pixels |
| | Depth resolution | 512×424 pixels |
| | Frame rate | 30 fps |
| | Ranging technology | Time-of-flight |
| | Connectivity | USB 3.0 |

The motivation is that for using 3D reconstruction in applications, building custom capture devices can be prohibitively expensive and complex. One of such device that was also used in experiments of this dissertation is Microsoft Kinect 2. The specifications for this device can be seen in Tab. 3.1.

The RGB-D cameras in the scene to be captured should be set up in such a way that the area covered by cameras is maximized. An example of two-camera capture setup is seen in Fig. 3.1. Positioning of cameras is dependent on many variables that are difficult to quantify. In many cases this means placing devices using heuristic rules and evaluating results of capture. If there are issues with captured data then cameras are repositioned and results are evaluated again. As to the placement rules, protocols for two basic scenarios were developed: capturing an object observable from many sides and capturing a large scene with backgrounds. These scenarios are visualized in Fig. 3.2 and discussed in next paragraphs.

When capturing lone standing objects the cameras should be placed around it in circular fashion with equal spacing. Typically cameras should be placed at same heights for initial experiment. Human actor capture is most common application of this scenario. Four cameras were found to be sufficient to achieve good point cloud coverage. When increasing the number of cameras, it is best to place cameras at different heights. For example Dou et al. (2016) use 8 cameras with four placed at low height and another four at high position to observe human actors from four sides.

When capturing large scenes, such as indoors environments, cameras are typically placed sparsely. For seamless reconstruction, the camera views should overlap to some degree. This is also necessary to calibrate the positions of cameras using known calibration

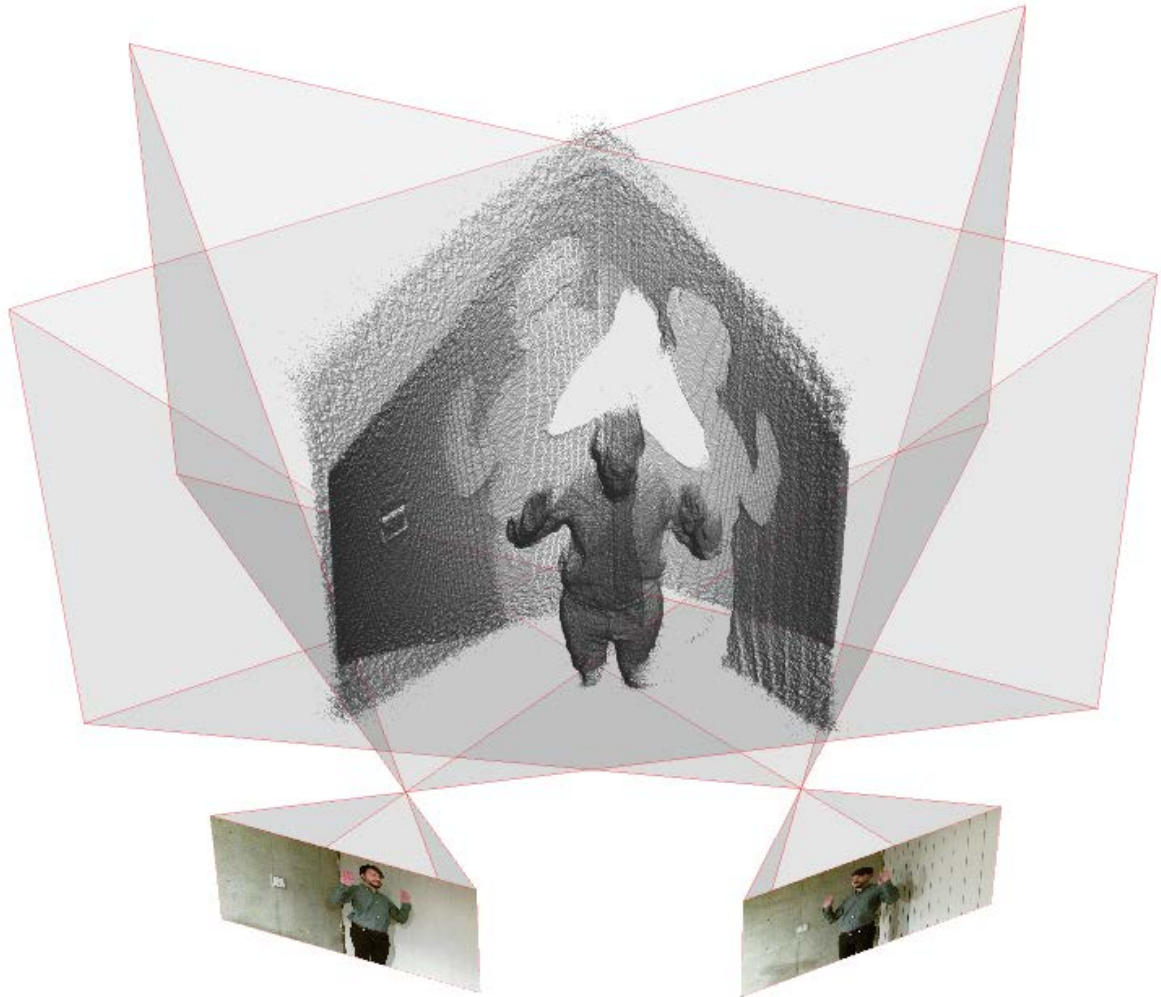
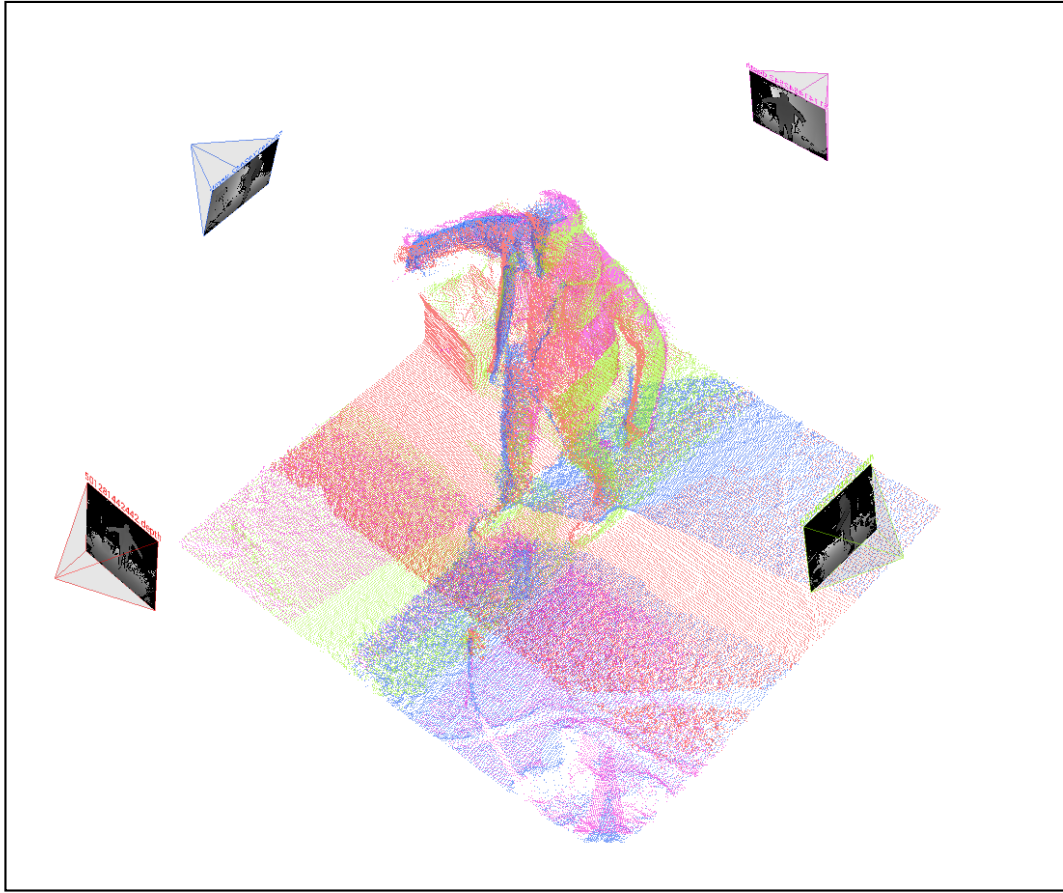
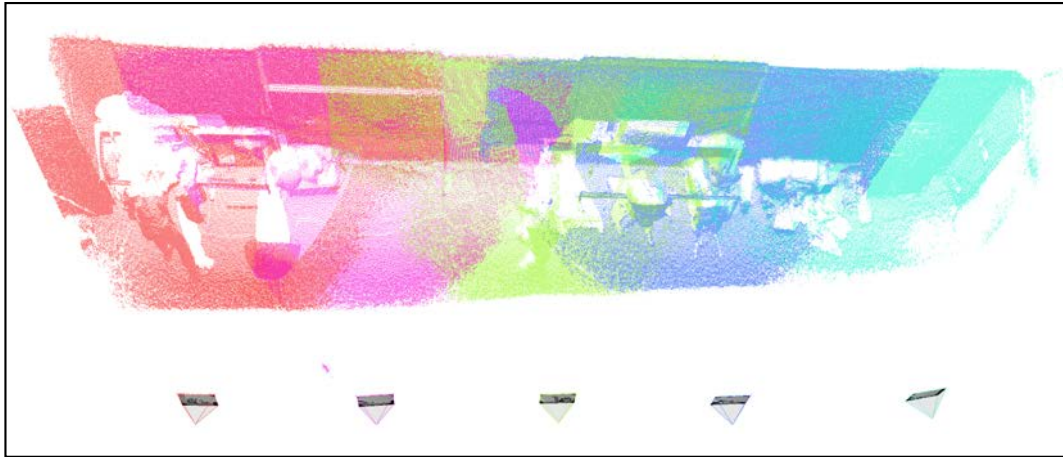


Figure 3.1: Example of typical scene setup. Two RGB-D cameras, illustrated with color images, are set up. Shaded boxes show camera field of view and typical depth camera range. Points represent the point cloud captured using camera depth sensor.



(a) High camera overlap



(b) Low camera overlap

Figure 3.2: Two major scene setup scenarios: (a) camera views have high degree of overlap with many areas covered by four cameras, and (b) camera views have limited overlap with no area being observed with more than two cameras.

object. Typically RGB-D camera depth maps have highest quality in the center of depth maps and considerable noise in the edge areas. For this reason, quality of reconstruction can be increased by introducing more camera field-of-view overlap.

3.2 System topology

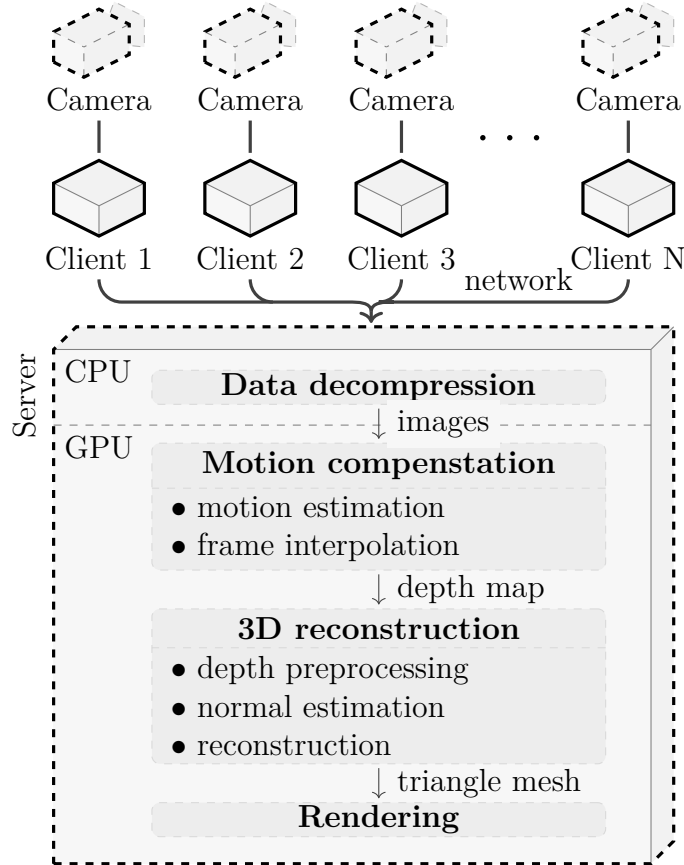


Figure 3.3: Multi-camera capture system topology with main processing steps.

The proposed capture system contains multiple layers of devices. Currently available consumer-level RGB-D devices are mostly connected to computers over USB. This puts strict limits of cable length and capturing large scenes becomes difficult. For this reason, a small computer, called client, is bundled together with each camera. The client computer task is to receive images from cameras, compress them, and transmit them over the local Ethernet network. The data is received by a single server machine that is tasked with 3D reconstruction. First, the received information is decompressed and uploaded

to the graphics card memory. Rest of the processing, including motion estimation and compensation, 3D reconstruction and rendering or application specific processing takes place in GPU. All system devices, their connectivity and basic flow of data is shown in Fig. 3.3.

It is not possible to transmit camera images over the network without compression as it would result in 1.5Gbit/s bandwidth per camera that exceeds Gigabit Ethernet capabilities. The solution is to use JPEG compression for color images and Zstandard compression for depth images. Typical color and depth stream combined bandwidth in that scenario is around 150Mbit/s, but very cluttered scenes with lots of fine details can increase the bandwidth up to 300Mbit/s.

The image data is transferred over User Datagram Protocol (UDP) to server using custom format. Every frame is fragmented into packets that do not exceed the local network maximum transmission unit (MTU), which is normally 1500 bytes. Each packet contains high-precision timestamp, that uniquely identifies each camera frame, and both total compressed image size and offset of the current fragment data inside the frame. The reason for attaching such header to each packet comes down to the fundamentals of UDP. Namely, the packets do not have guaranteed order of arrival and there are no retransmissions in case of packet drops. Regardless of what packet arrives first for a frame, enough information is available to allocate a new frame with specified timestamp and size. The amount of data that has been received is also being kept track for each frame on the server side. If any packet is lost in transmission, the reassembly of the respective frame is never completed. In other words, loss of a single packet will cause a loss of the full frame. Fortunately, this event is very rare in the local area network setting and this situation can simply be ignored. Any incomplete frame is allowed to be buffered up to a certain timespan after which it becomes stale and the data is discarded. This guarantees that any partially received frames do not inflate the memory usage of the program indefinitely.

On the server side, all successfully received and reassembled frames are queued for decompression. To speed up the process, decompression is heavily multi-threaded. The color image data goes through an extra step, which is to recompress it using S3 Texture Compression algorithm DXT1. The resulting image is fairly compact, storing 16 pixels in just 64 bits of data. The purpose of such compression is to reduce the required PCI

Express bandwidth to send images to a graphics card.

It takes variable amount of time for image data to be read from RGB-D devices, compressed, transmitted over network, decompressed and, depending on image type, recompressed. For this reason all images are stored in buffers after traversing the pipeline. All GPU processing of frames happens with a delay, typically 500ms, from actual capture time t . Essentially, the system defines a deadline $t' = t + 500\text{ms}$ at which point all image data has to be received and processed. At the deadline moment t' the buffers are searched for frames just before and after time t for all cameras. The proposed reconstruction methods require four consecutive frames for each camera – two frames before time t and two frames after t to be available. As it can be prohibitively expensive to upload four frames per camera to the GPU on each reconstruction pass, an image caching mechanism is utilized based on ring buffers. When uploading frames, for each camera separately, a position corresponding to time t is located from a four frame ring buffer. Any frames that do not match in the buffer are overwritten by new data. The result is that when the reconstruction frame rate is same as RGB-D device frame rate, only one image per stream needs to be uploaded on each reconstruction pass.

3.3 Camera calibration

Each camera, a single sensor and lens combination, has intrinsic parameters that include focal length, principal point coordinates and distortion parameters. The relative positions of cameras, typically given as Euclidean rotation and translation data, make up the extrinsic information. Intrinsic information for all cameras and the extrinsic transformations between them need to be known with high precision for the reconstruction method to work well.

3.3.1 RGB-D camera intrinsic and extrinsic parameters

Consumer RGB-D devices are typically factory-calibrated and the intrinsic parameters and extrinsic transformation between sensors can be read from cameras. Unfortunately some information can be stored in non-standard or proprietary formats. For example, the Microsoft Kinect 2 device has focal length and principal point data for both depth

and color sensors, but distortion info is only available for depth sensor. Also there is no standard extrinsic transformation data available.

Fortunately, the Kinect device includes a proprietary function that can project points from depth camera coordinate space to color camera image plane. Technically this is done using a high-degree polynomial that takes 3D point in depth sensor reference frame as input and produces 2D point on the color sensor image plane as output. This polynomial is non-trivial to decompose into linear Euler transformation and non-linear color sensor distortion parts. Nevertheless, the camera calibration algorithms can be applied to reveal that data. In fact, the developed calibration method is general and works as long as there exists a function that takes points from depth camera reference frame and projects them to color image plane.

The actual calibration works as follows. First step is to generate thousands of random 3D points in depth camera space. The camera field of view should be uniformly covered and the points should lie uniformly between minimum and maximum depth camera sensing range. Then the proprietary RGB-D camera transformations are applied to find corresponding 2D points on the color camera image plane. Next, the extrinsic transformation as linear rotation and translation between sensors is estimated. This can be done using various methods, but since the data is algorithmically generated and free of outliers and noise, a simple Levenberg-Marquardt optimization was found to work best. The optimizer is initialized with identity rotation matrix and zero translation vector. This is because both sensors are roughly looking at the same direction and situated very near to each other. The optimizer then iteratively changes the initial guess by minimizing reprojection error of the 3D to 2D projection. Finally, Levenberg-Marquardt optimization can be reused to find out color camera distortion parameters. Brown-Conrady distortion model is used with all parameters initialized as zeros. Again, the optimization minimizes reprojection error iteratively, yielding distortion parameters as a result. Most of these operations can be found in computer vision libraries such as OpenCV (Bradski, 2000). To get an estimate of the calibration accuracy, this calibration can be run multiple times to measure standard deviation of the calibration results.

3.3.2 Extrinsic calibration between RGB-D cameras

The relative pose of all RGB-D cameras placed in the scene needs to be known in terms of Euclidean rotation and translation matrices. In theory, the 3D reconstruction method allows moving cameras as long as pose info is available at all times. However, in practice, this requires very accurate camera tracking and there is no real need for moving the devices. Therefore the cameras in this dissertation were placed in fixed positions for experiments.

Relative positions between cameras were estimated using color images. Because the transformations between color and depth sensors of each RGB-D device were already derived in Sec. 3.3.1, it is possible to chain transformations and determine relative positions of different depth sensors.

Due to the high calibration precision required, instead of relying on automatically detected scene features, a more reliable calibration object was preferred. For convenience, a checkerboard object with 7 rows and 11 columns was utilized, giving a total of 77 calibration points per captured frame. Unfortunately, checkerboard has a weakness in terms of symmetry and its position cannot be uniquely determined. For any board which has different number of rows and columns the board has 180° rotational symmetry. Fortunately all of the cameras have been placed in scene with similar orientation – the ‘up’ direction in images always points to ceiling. This is sufficient constraint to resolve the rotational symmetry ambiguity.

The calibration protocol is as follows. First, system user selects one camera that is fixed to a global coordinate origin. Poses of other cameras are then estimated relative to the initially fixed device. The cameras can be placed in various configurations in the scene. The only constraint is that the field-of-view of cameras has to overlap in such way that calibration object can be seen simultaneously. All the cameras do not have to see the calibration object at the same time, it is sufficient that there exists a chain of transformations from every camera to any other camera. An interactive calibration tool was built to allow user to select which camera positions to fix in which order.

Care has to be taken to accurately estimate camera poses. First, pose information is collected over multiple frames. The pose estimation happens in real time and user is

presented with both numerical and visual results of calibration that helps verify the quality of results. Chaining transformations between cameras can result in accumulation of errors. Therefore it is best to estimate rotation and translation between cameras multiple times. To average transformations, the rotation should be represented using quaternions. This also allows user to estimate standard error of the transformations to get a sense of how accurate the calibration is.

3.4 Time calibration

Consumer-grade RGB-D devices typically do not have the possibility to control when shutters are triggered. This presents a problem for multi-camera capture systems as moving objects may be captured at different times depending on device. The first step to compensate for different shutter times is to find accurate timestamps all frames captured by devices. The time calibration consists of two parts: server–client computer time synchronization and client–camera time calibration.

Previously, Alexiadis et al. (2017) proposed an offline method for calibrating times of RGB-D devices using audio. Namely, all devices record both audio and video during scene capture. Then, in the offline step, audio is used to calibrate frame capture times. This approach could achieve around 15 ms accuracy. A more direct time calibration technique is proposed that works online and achieves at least 1 ms accuracy.

Synchronizing time between server and client machines is carried out using Network Time Protocol (NTP). Initial time for the server is obtained from nearby stratum 2 clock. Next, all clients synchronize with the server machine over the local Ethernet network. Standard NTP synchronization can only achieve 1 ms accuracy. However, all network adapters within the proposed system support hardware timestamping feature. This allows clocks to be synchronized to sub-microsecond accuracy. The client systems consistently reported the estimated time synchronization accuracy below 1 μ s relative to the server machine.

Calibrating time between RGB-D device and client computer is made difficult due to the fact that there is no direct method to synchronize clock of RGB-D device. Typically cameras have precise internal timers that are started on device initialization. The internal

time is exposed in image frames transmitted to the computer. In case of Microsoft Kinect 2 the timestamp has 125 μs precision. The cameras are normally connected over USB which has variable latency of packet delivery. On the computer side the arrival times of the USB packets are recorded using high precision timers. Time calibration can be achieved through statistical modeling of received timestamps.

Let t_c and t_d be the values of computer and camera device internal timers, respectively. The relation between these timers is modeled as

$$t_c = (1 + s_d)t_d + o_d + c_{dc}, \quad (3.1)$$

where s_d is the clock skew, o_d is the offset between the timers, and c_{dc} is the average time between image capture and delivery of the image to the computer. It is possible to recover s_d and o_d using a linear least squares regression analysis. The constant c_{dc} , however, is dependent on the hardware and software being used. Because the capture setup consists of homogeneous hardware, it is assumed that this time delay is constant across devices. Hence, the relative timestamps of any captured frames remain valid.

From a sample of 3000 timestamps captured over 100 seconds, a significant time skew s_d of $-179.2 \pm 0.4\text{PPM}$ (confidence level of 95%) was found. If the time calibration is repeated after every 100 seconds, the time uncertainty of $\pm 40 \mu\text{s}$ is obtained from skew. The timer offset o_d has a confidence interval of $\pm 23 \mu\text{s}$. It is possible to conclude that the Kinect 2 RGB-D camera has a reliable internal timer that can be calibrated to sub-millisecond precision. However, the calibration should be repeated periodically, for instance, every 100 seconds, to reduce the time uncertainty from clock skew estimation error.

Chapter 4

Motion estimation and correction

As detailed in Sec. 3.4, all the image frames received from the cameras are assigned precise timestamps. Due to the consumer-oriented nature of the hardware, it is not possible to control shutter trigger times. The result is that the cameras capture fast moving objects at different times. A naive 3D reconstruction of such data would result in one object appearing at slightly different locations simultaneously.

The proposed solution is to generate new synthetic camera frames such that it appears as if the scene were captured at the same time by all RGB-D devices. Essentially, consecutive camera frames are interpolated to produce new images. Since the interpolation method is continuous, it is possible to generate data for any timepoint. This also leads to interesting applications such as generating synthetic slow motion videos. An example of captured point clouds with and without interpolation can be seen in Fig. 4.1.

The basic strategy for interpolating depth data from cameras is to estimate scene flow for every camera separately. The depth can then be warped to interpolated time by applying scaled scene flow vectors to depth points. For an overview of scene flow estimation works please refer to Yan and Xiang (2016).

Unfortunately, the available methods tend to be computationally too expensive to be practical for estimating multiple flow maps in real time. Therefore, a new scene flow estimation method was designed which trades some estimation quality for speed. In a nutshell, scene flow is estimated by finding correspondences in consecutive depth maps. A mesh of depth points is generated for one frame which is then warped iteratively to the closest points on the second depth map frame. Regularization is achieved by imposing

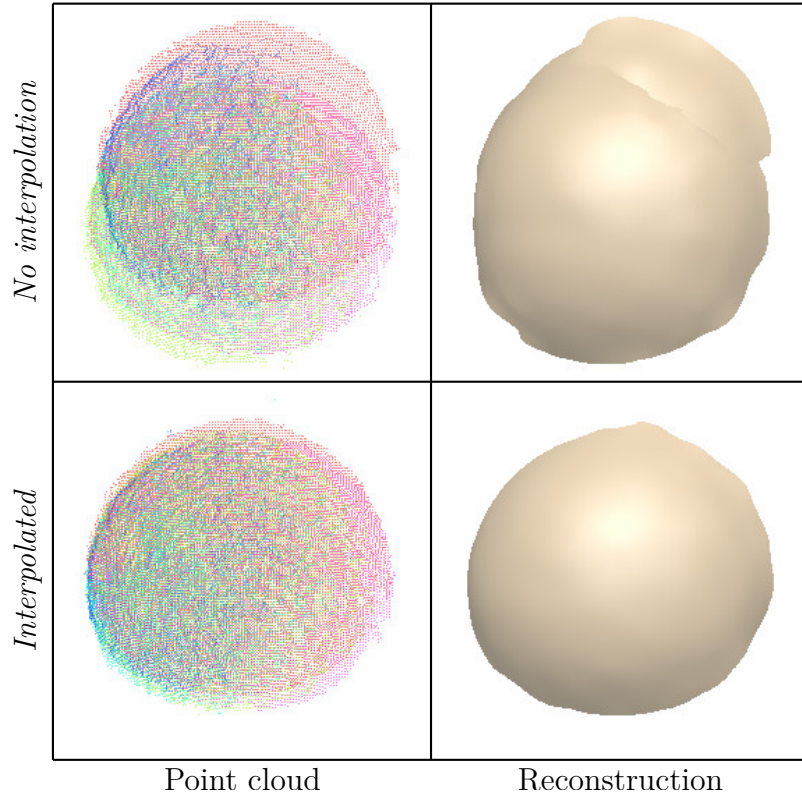


Figure 4.1: Demonstration of frame interpolation using a fast-moving spherical object. Without interpolation (top row), the point clouds from the different cameras do not align. Using interpolation (lower row), the point clouds align and the object is correctly reconstructed.

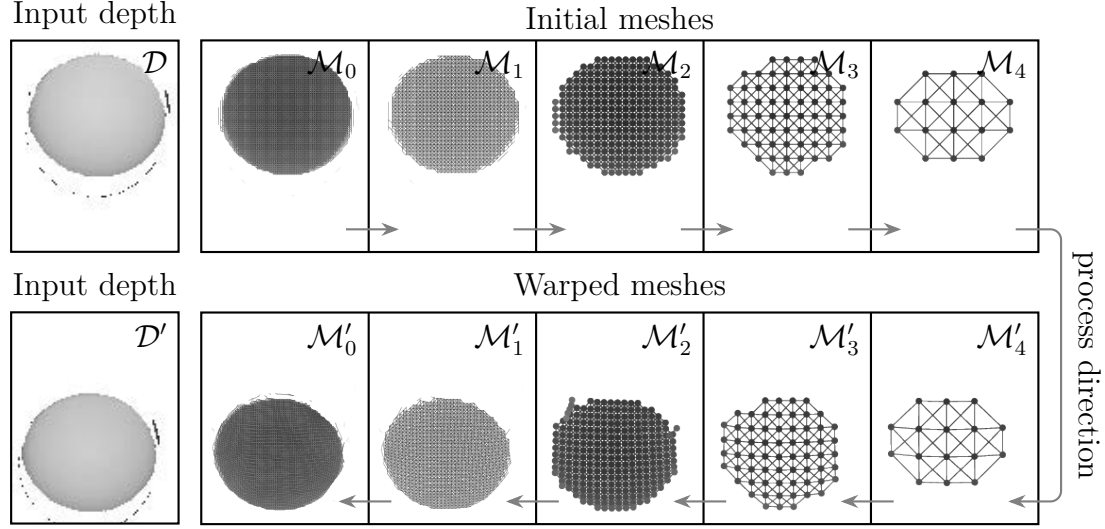


Figure 4.2: Scene flow estimation steps demonstrated on quickly moving spherical object. Mesh \mathcal{M}_0 is generated from input \mathcal{D} and is transformed through a series of steps, resulting in mesh \mathcal{M}'_0 , which closely matches the second input depth map \mathcal{D}' .

some local rigidity constraints on the mesh.

4.1 Scene flow estimation

The scene flow estimation takes as input two consecutive depth maps \mathcal{D} and \mathcal{D}' from one RGB-D camera. For the first depth map, \mathcal{D} , a dense mesh \mathcal{M} is generated from the depth pixels. Essentially, all neighboring depth points with distance less than a user-set threshold m_t are connected with an edge. The aim here is to warp this mesh so that it matches depth map \mathcal{D}' . The warped mesh is denoted \mathcal{M}' . The scene flow vectors for depth points then equal the displacement of vertices between meshes \mathcal{M} and \mathcal{M}' .

To reduce the computational cost of finding correspondences between \mathcal{M} and \mathcal{D}' , multiple mesh and depth scales, respectively $\{\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_n\}$ and $\{\mathcal{D}'_0, \mathcal{D}'_1, \dots, \mathcal{D}'_n\}$, are generated. A mesh scale \mathcal{M}_i is found by downscaling \mathcal{M}_{i-1} to half size. The downscaling works by removing every second vertex in horizontal and vertical direction. Vertices in \mathcal{M}_i are joined by edges only if there exists a path of edges with length two or less connecting corresponding vertices in mesh \mathcal{M}_{i-1} . The mesh warping strategy begins at the highest scale mesh \mathcal{M}_n , which is iteratively matched to target depth map

\mathcal{D}'_n . The warp parameters are then propagated down to the next mesh scale \mathcal{M}_{n-1} . The correspondences between n and $n - 1$ level vertices are stored when downsampling. Hence it is possible to simply copy flow data for vertices which exist in both levels. The data for vertices that only exist in level $n - 1$ can be generated by averaging flow vectors of neighboring edge connected vertices with already copied data. The warp estimation and propagation procedure is repeated on each scale until reaching mesh \mathcal{M}_0 .

The mesh warping procedure works in two steps. First, for all vertices in \mathcal{M}_i closest points in \mathcal{D}'_i are found using a grid search and the new vertex positions stored in \mathcal{M}'_i . Since the warping is carried out in multiple scales, it was found that a fairly small search window of 5×5 is sufficient for finding good correspondences. Secondly the found correspondences need to be regularized to get more accurate flow vectors. This works by imposing some local rigidity constraints on the meshes. Given corresponding vertices $v \in \mathcal{M}_i$ and $v' \in \mathcal{M}'_i$, the energy function

$$E_i = \sum_{a,b \in \mathcal{M}_i} \left\| (v_a - v_b) - (v'_a - v'_b) \right\|^2 \quad (4.1)$$

is minimized. Here the sum is taken over all pairs of vertices connected with edges in the mesh. The energy optimization is carried out using gradient descent due to its simplicity. In practice, each gradient descent iteration makes the mesh more rigid. Hence, it is possible to tune the mesh rigidity via the number of iterations.

A single warping pass on each mesh level is typically sufficient for slowly moving objects. In the case of rapidly moving objects, the depth has to be warped long distance from original location. Since the point correspondence search just selects closest points, lot of the initial matches can be inaccurate. Regularization cannot completely fix bad initial correspondences. Hence it is best to repeat warping few times using last warping result to increase quality. The major steps of the flow estimation are shown in Fig. 4.2.

4.2 Depth frame warping

The final step of motion estimation is to interpolate the depth frames based on the depth map meshes calculated in the previous subsection.

The proposed system runs with constant frame rate determined by user. That is,

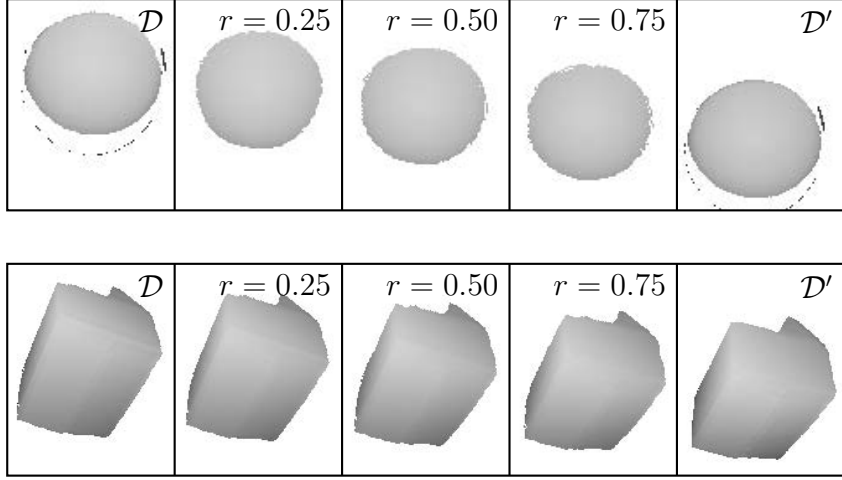


Figure 4.3: Depth frame interpolation examples. Three interpolated depth maps with interpolation ratios of 0.25, 0.50, and 0.75 are generated from input depth maps \mathcal{D} and \mathcal{D}' for a falling spherical object and a falling carton box.

the reconstruction time is not influenced by RGB-D camera frame timestamps. Let the reconstruction time be t . In that case, for each camera separately, the consecutive depth frames \mathcal{D} and \mathcal{D}' are found from the buffers at times t_1 and t_2 , respectively, such that $t_1 \leq t < t_2$. The interpolation ratio between those depth frames is then $r = (t_2 - t) / (t_2 - t_1)$.

Next, a new mesh \mathcal{M}'' is generated by interpolating the vertex positions of meshes \mathcal{M}_0 and \mathcal{M}'_0 . Given the corresponding vertices $v \in \mathcal{M}_0$ and $v' \in \mathcal{M}'_0$, the new vertex position for mesh \mathcal{M}'' is $(1 - r)v + rv'$. The resulting mesh \mathcal{M}'' can effectively be rendered using standard computer graphics tools to produce a new interpolated depth map. Figure 4.3 shows an example of interpolation at three different time points using real data.

Chapter 5

3D reconstruction

This chapter presents the proposed 3D reconstruction methods. First some necessary preprocessing steps such as initial normal estimation is described. Then the basics of moving least squares based 3D reconstruction are explained. Next, two actual methods, ZipperMLS and FusionMLS, are presented that are different in their approach, but share the same underlying MLS mathematics. Finally, the chapter is concluded with results and comparisons of the proposed methods.

5.1 Normal estimation

The proposed 3D reconstruction methods require rough initial surface normals to be available at each input point location. Normal methods were designed such that they are both suitable for use with MLS-based reconstruction methods and can be efficiently executed on GPUs.

It is preferred to calculate normals for each camera separately. The main reason is that point clouds of different cameras might not always perfectly align. This can distort normal calculations. Additionally, estimating normals for each camera separately allows very efficient GPU processing. In terms of accuracy, some inaccuracies are not a problem as normals from different depth maps are eventually combined in 3D reconstruction stage.

The normal estimation starts by calculating horizontal and vertical gradients as

$$g_x(x, y) = p(x + 1, y) - p(x - 1, y) \quad (5.1)$$

and

$$g_y(x, y) = p(x, y + 1) - p(x, y - 1) \quad (5.2)$$

at all depth pixel coordinates. Here $p(x, y)$ is a 3D point corresponding to depth map pixel at coordinates (x, y) . Next, temporary normals are calculated as

$$u(x, y) = g_x(x, y) \times g_y(x, y). \quad (5.3)$$

It is important to note that depth maps captured by RGB-D devices can contain invalid or missing pixels. Any gradients $g_x(x, y)$ or $g_y(x, y)$ that contain such pixels are marked as invalid. In turn, the invalid gradients can propagate to $u(x, y)$ which is in that case set to $[0, 0, 0]^T$.

The calculated normals $u(x, y)$ are very noisy and require further smoothing. In the next paragraphs two methods are provided for this purpose: integral images based estimation and spatially weighted estimation.

Integral images based normal estimation. The aim is to smooth $u(x, y)$ normal values over small rectangular window using simple averaging. For window size $N \times N$ it is necessary to fetch N^2 normal values from memory to calculate the average. This can quickly become an expensive operation as N increases. The computational cost can be reduced using integral images.

First step of using integral images is to calculate sum

$$U(x, y) = \sum_{i \leq x, j \leq y} u(i, j) \quad (5.4)$$

for each pixel location (x, y) . This operation can be implemented very efficiently on GPUs as two passes over images – first summing values horizontally and then vertically. Next, the unnormalized normal can be expressed as

$$v(x, y) = U(x_2, y_2) + U(x_1, y_1) - U(x_2, y_1) - U(x_1, y_2), \quad (5.5)$$

where

$$x_1 = x - \frac{N}{2} - 1, \quad (5.6)$$

$$x_2 = x + \frac{N}{2}, \quad (5.7)$$

$$y_1 = y - \frac{N}{2} - 1, \quad (5.8)$$

$$y_2 = y + \frac{N}{2}. \quad (5.9)$$

Final normal can be obtained by normalizing $v(x, y)$ as

$$n(x, y) = \frac{v(x, y)}{\|v(x, y)\|}. \quad (5.10)$$

Effectively, the integral images approach allows smoothing normals over any chosen window size without change in computational cost. The integral image calculations are cheap to compute and evaluating a single normal requires fetching only four image values from memory. The drawback is that normal values are averaged over rectangular window without regard to spatial distance. Next paragraphs describe an alternative weighted normal calculation method in case higher quality normals are required.

Spatially weighted normal estimation. In this method, the normals are calculated as a spatially weighted sum. For some depth map location (x, y) the unnormalized normal is found over small window as

$$v(x, y) = \sum_{i,j} u(i, j) w\left(\|p(x, y) - p(i, j)\|\right), \quad (5.11)$$

where $w(\cdot)$ represents spatial weighting. Following Guennebaud and Gross (2007), the weight function is defined as

$$w(r) = \begin{cases} \left(1 - \left(\frac{r}{h}\right)^2\right)^4 & r < h \\ 0 & \text{otherwise} \end{cases} \quad (5.12)$$

where h is a constant spatial smoothing factor. This weight is essentially a fast approximation of Gaussian function. Final normal can be obtained similarly to integral images approach by normalizing $v(x, y)$ as

$$n(x, y) = \frac{v(x, y)}{\|v(x, y)\|}. \quad (5.13)$$

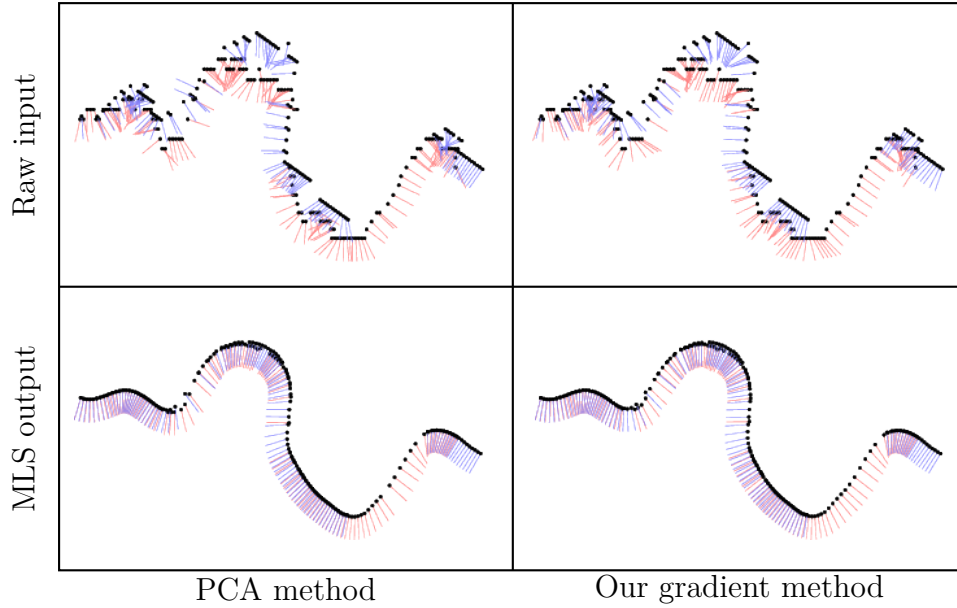


Figure 5.1: Comparison of normal estimation methods using real data. Top row shows initial normals and lower row shows normals after smoothing with MLS. Here integral images based estimation was used since in theory it has lower quality and better highlights how well the MLS smoothing works.

The window size of normal calculation can be derived from spatial smoothing factor h and RGB-D camera parameters dynamically for each point. However, in terms of GPU code optimization, it was best to use a hardcoded 7×7 pixel window in the test scenes.

Comparison. The difference between integral images and spatial weighting methods is essentially estimation quality and processing cost tradeoff. First method is faster and second slower. Conversely, second method has higher quality than first one. Which method is better suited depends on various parameters such as point cloud density, types of noise in depth map and available execution time. It is best to select the right method empirically.

Fig. 5.1 shows normal estimation results and compares them to well-known Principal Component Analysis (PCA) based method. Arguably, the normal calculation has rather low quality. However, this is not a problem due to the way the proposed 3D reconstruction methods work. Namely, MLS-based reconstruction takes rough initial normals as input and produces smoothed surface normals as output. It can be seen from the figure that

regardless of initial estimation quality, after applying MLS smoothing, the final normals are of high quality. In reverse, this implies that the initial normal quality is sufficient.

5.2 Moving least squares

MLS surface smoothing represents a family of methods that can define smooth surfaces for point clouds. For an overview of MLS methods please refer to Cheng et al. (2008). This section explains both the core ideas behind MLS and different ways of applying them.

Very generally, the 3D reconstruction problem can be defined as follows. The depth maps generated in Sec. 4 form a point cloud, more formally a point set, $\mathcal{P} = \{\mathbf{p}_i \in \mathbb{R}^3\}$ where $i \in \{1, 2, \dots, N\}$ for N points. Additionally, rough surface normals \mathbf{n}_i were estimated for each point \mathbf{p}_i in Sec. 5.1. Assume that the points \mathbf{p}_i and normals \mathbf{n}_i have been sampled from an unknown surface \mathcal{S} , but contain noise introduced by RGB-D sensors. The objective is to reconstruct surface by approximating \mathcal{S} from the sampled set \mathcal{P} .

The general concept of MLS is to approximate surface \mathcal{S} *locally*. It means that when estimating the surface geometry around some point \mathbf{x} then only points \mathbf{p}_i in the vicinity of \mathbf{x} are used in calculations. This contrasts to *global* reconstruction methods that combine information from the full set \mathcal{P} to find surface geometry. The main benefit of local methods is that the estimation is easier to partition or parallelize into independent calculations.

Various MLS formulations exist and they can be split into two categories: projection and implicit MLS methods. The projection methods produce new refined point clouds as output whereas implicit methods result in a scalar signed distance field. The proposed reconstruction methods use both techniques.

Projection MLS methods define surface as a point projection operator $g: \Omega \rightarrow \mathbb{R}^3$. Let $\mathbf{x} \in \Omega$ be a point in vicinity of a surface \mathcal{S} . In that case the function g will project the point \mathbf{x} on to the approximate surface \mathcal{S} . It implies that any point $\mathbf{s} \in \mathcal{S}$ already on the surface will yield $g(\mathbf{s}) = \mathbf{s}$.

Implicit MLS methods approximate surfaces around a point \mathbf{x} in space as an implicit function $f: \mathbb{R}^3 \rightarrow \mathbb{R}$ representing the algebraic distance from a 3D point to the surface \mathcal{S} . In other words, the implicit function f defines a continuous scalar signed distance field. The signed distance field has a sign depending on whether the sampled point lies inside

objects or outside.

This work follows MLS interpretation put forward by Alexa et al. (2003) who define an iterative MLS process suitable for fast execution that can also give high quality results. The method first defines an implicit MLS function and then proceeds to build projection methods on top of it. Next subsections detail both operations.

5.2.1 Signed distance field

The basis of implicit reconstruction methods is to find an implicit function f which approximates distance to surface \mathcal{S} . The strategy here is to locally approximate \mathcal{S} as a plane and then obtain f result by calculating the distance to this plane.

Given some point \mathbf{x} , an average point location a and normal n first needs to be computed as

$$a(\mathbf{x}) = \frac{\sum_i w(\|\mathbf{x} - \mathbf{p}_i\|) \mathbf{p}_i}{\sum_i w(\|\mathbf{x} - \mathbf{p}_i\|)} \quad (5.14)$$

and

$$n(\mathbf{x}) = \frac{\sum_i w(\|\mathbf{x} - \mathbf{p}_i\|) \mathbf{n}_i}{\sum_i w(\|\mathbf{x} - \mathbf{p}_i\|)}, \quad (5.15)$$

where $w(r)$ is a spatial weighting function and \mathbf{n}_i are the normals calculated in Sec. 5.1.

Numerous candidates for the weight function $w(r)$ have been put forward in the literature. This work uses the fast Gaussian function approximation previously defined in Eq. 5.12. Recall that this weight function has a single parameter h that determines its radius in 3D space.

The sums in Eq. 5.14-5.15 are taken over all points in vicinity of \mathbf{x} . Due to the cutoff range of the weighting function $w(r)$, considering points in the radius of h around \mathbf{x} is sufficient. Traditionally, points \mathbf{x}_i and normals \mathbf{n}_i are stored in spatial data structures such as k-d tree or octree. While fast, the spatial lookups still constitute the biggest impact on MLS performance. Kuster et al. (2014) propose storing points and normals data as two-dimensional arrays similar to range images. In that case, a lookup operation would consist of projecting search points to every camera and retrieving an $u \times u$ block of points around projected coordinates, where s is known as window size. This allows for very fast lookups and is cache friendly.

Finally, the position of surface is estimated as an implicit function

$$f(\mathbf{x}) = n(\mathbf{x})^T(\mathbf{x} - a(\mathbf{x})). \quad (5.16)$$

It is important to note that the implicit function accuracy greatly depends on the distance from the surface \mathcal{S} . Due to the action of the weight $w(r)$ the surface estimation becomes more local, and hence more accurate, when point \mathbf{x} is close to \mathcal{S} .

5.2.2 Point projection

Let \mathbf{x} be a point that needs to be projected to a surface \mathcal{S} . The core idea is to move \mathbf{x} closer to surface by projecting it to a locally estimated plane. As previously mentioned, the surface is estimated more accurately when \mathbf{x} gets closer to \mathcal{S} . Hence the plane estimation and point projection steps are repeated iteratively to move points closer to the true surface.

Alexa and Adamson (2004) present multiple ways of projecting points to the implicit surface. One core concept of this work is that the implicit function $f(\mathbf{x})$ can be understood as a distance from an approximate surface tangent frame defined by point $a(\mathbf{x})$ and normal $n(\mathbf{x})$. This means that a point \mathbf{x} can be projected to this tangent frame along the normal vector \mathbf{n} using

$$\mathbf{x}' = \mathbf{x} - f(\mathbf{x})n(\mathbf{x}). \quad (5.17)$$

This is called a *simple projection*. Since the tangent frame is only approximate, the procedure needs to be repeated. On each iteration, the surface tangent frame estimation becomes more accurate as a consequence of the spatial weighting function $w(r)$. Another option is to propagate points along the $f(\mathbf{x})$ gradient. This is called *orthogonal projection*.

5.3 ZipperMLS reconstruction method

ZipperMLS is a reconstruction method that combines MLS point cloud smoothing with mesh generation and merging methods. First the motivation for this approach is explained together with the main method steps. Then the method is described in detail, including novel MLS point cloud refinement scheme, mesh generation and mesh merging.

5.3.1 Overview

Direct triangle mesh generation (Hilton et al., 1996; Holz and Behnke, 2013; Kriegel et al., 2015; Orts-Escolano et al., 2015) from point clouds has been popular in 3D reconstruction systems (Maimone and Fuchs, 2011; Alexiadis et al., 2013). This method directly generates triangle meshes from RGB-D camera point clouds by connecting neighboring depth map points with edges to form triangle primitives. Its strengths are exceedingly fast operation and simplicity.

However, direct meshing can be problematic to use when the input point cloud is not smooth or when there are more than one RGB-D camera in use. ZipperMLS method adds a way to smooth surfaces and also consolidates meshes to form one single model when using two or more input devices.

MLS methods constitute an efficient way to smooth point clouds. Their memory requirements are dependent on the number of input points and not on a reconstruction volume. Additionally, the process is fully parallelizable, making it an excellent target for GPU acceleration (Guennebaud et al., 2008; Kuster et al., 2014). For this reason MLS smoothing is used to remove sensor noise.

Direct meshing of the input range images results in separate meshes for each range image. To reduce rendering costs and obtain a watertight surface, the meshes should be merged. Mesh zippering (Turk and Levoy, 1994; Marras et al., 2010) is a well-known method of doing that. However, this method is not particularly GPU-friendly. As such, a new approach was designed based on mesh zippering suitable for highly parallel execution.

Fig. 5.2 outlines the ZipperMLS reconstruction method processing steps. The depth maps generated by motion compensation in Sec. 4 are used as the input to the reconstruction pipeline. Normals are first estimated using points from depth maps. Then MLS reconstruction process takes both the raw points and initial normals and produces a set of refined points for mesh generation. The mesh generation is fairly complex process and contains multiple smaller steps. First initial meshes are generated from points of each RGB-D camera separately. Next, overlapping areas of multiple meshes are eroded so that only one layer of mesh is left. Next, boundaries of different meshes are welded in a merging process. Finally, the mesh is converted to a standard triangle mesh format and

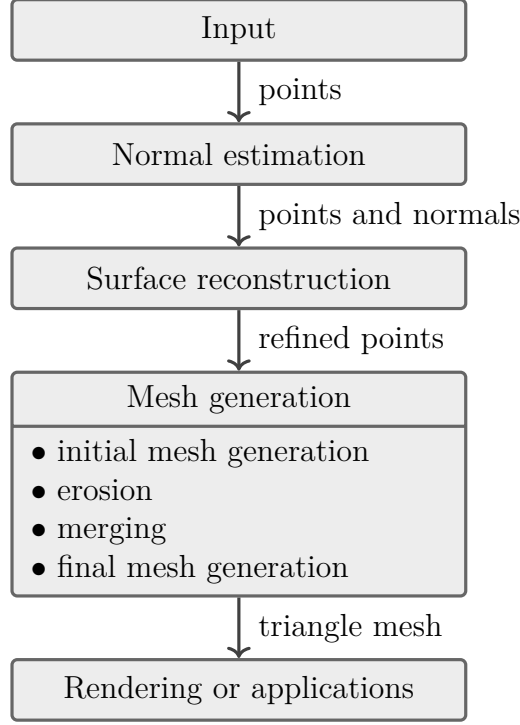


Figure 5.2: Overview of ZipperMLS reconstruction process steps.

in the process remaining issues with mesh connectivity are solved. The resulting triangle mesh can be rendered or used in applications.

5.3.2 MLS projection method

To refine point clouds the MLS projection methods presented in Sec. 5.2.2 can be used. The issue here is that direct meshing methods expect point clouds to have a regular grid-like structure when points are projected to camera. Unfortunately, the traditional MLS projection methods lose that structure. Therefore, it is preferable to develop a new MLS projection operator for which resulting point clouds conform to meshing requirements.

Instead of following normal vectors or an $f(\mathbf{x})$ gradient to the surface, the iterative optimization is confined to a line between the initial point location and the camera's viewpoint. Given a point \mathbf{x} to be projected to a surface and camera viewpoint \mathbf{v} , the projection will follow vector \mathbf{d} defined as

$$\mathbf{d} = \frac{\mathbf{v} - \mathbf{x}}{\|\mathbf{v} - \mathbf{x}\|}. \quad (5.18)$$

A novel *viewpoint projection* operator projects a point to the tangent frame in direction

\mathbf{d} instead of \mathbf{n} as in Eq. 5.17. With the use of some trigonometry, the projection operator becomes

$$\mathbf{x}' = \mathbf{x} - \frac{f\mathbf{d}}{\mathbf{n}^T\mathbf{d}}. \quad (5.19)$$

Algorithm 1 Point to surface projection

```

1:  $\mathbf{d} = (\mathbf{v} - \mathbf{x}) / \|\mathbf{v} - \mathbf{x}\|$ 
2:  $i = 0$ 
3: repeat
4:    $\mathbf{a} = a(\mathbf{x})$ 
5:    $\mathbf{n} = \mathbf{n}(\mathbf{x})$ 
6:    $f = \mathbf{n}^T(\mathbf{x} - \mathbf{a})$ 
7:   if  $f > 0$  then
8:      $f = \min(h, f / (\mathbf{n} \cdot \mathbf{d}))$ 
9:   else
10:     $f = \max(-h, f / (\mathbf{n} \cdot \mathbf{d}))$ 
11:  end if
12:   $\mathbf{x} = \mathbf{x} - f\mathbf{d}$ 
13:   $i = i + 1$ 
14: until  $f < \epsilon$  or  $i > i_{max}$ 

```

This operator works similarly to simple projection when \mathbf{d} is close to \mathbf{n} in value. However, this optimization cannot easily converge when \mathbf{n} and \mathbf{d} are close to a right angle. Conceptually, the closest surface is in a direction where the point to is not allowed to move. Dividing by $\mathbf{n}^T\mathbf{d}$ may propel the point extremely far, well beyond the local area captured by the implicit function f . Thus each projection step is limited to distance h (also used in the spatial weighting function 5.12). This results in a search through space to find the closest acceptable surface.

If a point does not converge to a surface after a fixed number of iterations i_{max} , the projection is considered to have failed and the point is discarded. This is a desired behavior and indicates that a particular point is not required. The pseudocode for the projection method is listed in Alg. 1.

Fig. 5.3 shows the visualization of different projection methods. Fig. 5.4 shows them in

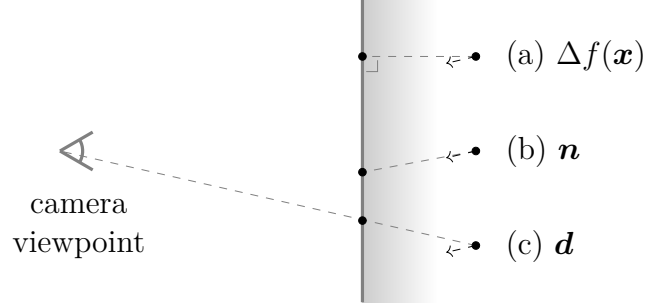


Figure 5.3: Visualization of different surface projection methods. (a) orthogonal projection, (b) simple projection, (c) viewpoint projection (this work).

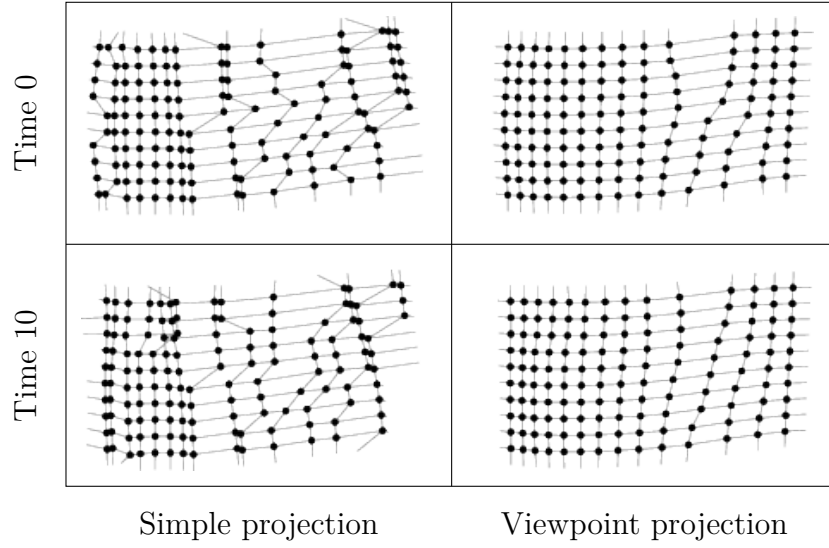


Figure 5.4: Comparison of simple projection (left) and the proposed viewpoint projection (right) for the same depth map patch. The latter shows excellent temporal stability.

action (except for orthogonal projection, which is computationally more expensive). The proposed method results in a more regular grid of points on a surface than the normals-based simple projection. This process is crucial in making the final mesh temporally stable. Moreover, the stability of the distances between points is a key condition to compute the mesh connectivity in the next section.

5.3.3 Mesh generation

The purpose of mesh generation is to take refined points produced by MLS and turn them into a single consistent mesh of triangles. The approach is to first generate initial triangle meshes for every RGB-D camera separately and then join those meshes to get a final result.

The proposed method is inspired by a mesh zippering method pioneered by Turk and Levoy (1994). This method was further developed by Marras et al. (2010) to enhance output quality and remove some edge-case meshing errors. Both zippering methods accept initial triangle meshes as input and produce a single consistent mesh as output. Conceptually, they work in three phases:

1. **Erosion** - Remove triangles from meshes so that overlapped mesh areas are minimized.
2. **Clipping** - In areas where two meshes meet and slightly overlap, clip triangles of one mesh against triangles of another mesh so that overlapping is completely eliminated.
3. **Cleaning** - Retriangulate areas where different meshes connect to increase mesh quality.

Prior zippering work did not consider the parallelization of these processes. As such, it is necessary to modify the approach to be suitable for GPU execution.

The mesh erosion process of zippering utilizes a global list of triangles. The main operation in this phase is deleting triangles. If parallelized, the triangle list would need to be locked during deletions to avoid data corruption. For this reason, a new data structure needs to be introduced where triangles are not deleted, only updated to reflect a new state.

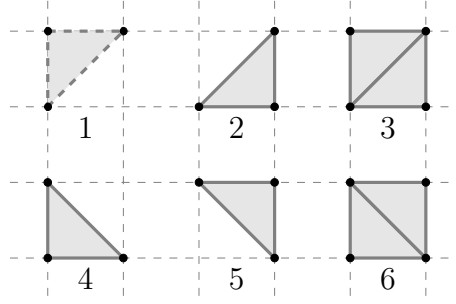


Figure 5.5: Forming triangles adaptively between vertices. Each number indicates the triangle formation type. Type 0, which represents an empty cell, is not shown.

This allows for completely lockless processing on GPUs. A similar issue arises with mesh clipping, as it would require locking access to multiple triangles to carry out clipping. To counter this, the mesh clipping is replaced with a process called mesh merging. It updates only one triangle at a time and thus does not requiring locking. The last step of the mesh generation process is to turn the custom data structures back into a traditional triangle list for rendering or other processing. It is called final mesh generation. This step also assumes the use of mesh cleaning as seen in previous works. In summary, the mesh generation consists of following steps:

1. **Initial mesh generation** - Create a separate triangle mesh for every RGB-D camera depth map.
2. **Erosion** - Detect areas where two or more meshes overlap (but do not delete triangles like in zippering).
3. **Merging** - Locate points where meshes are joined.
4. **Final mesh generation** - Extract a single merged mesh.

The next sections discuss each of these points in detail.

Initial mesh generation

The first step of meshing process is to generate a triangle mesh for each depth map separately. In practice, neighboring pixels are joined in the depth map to form the triangle

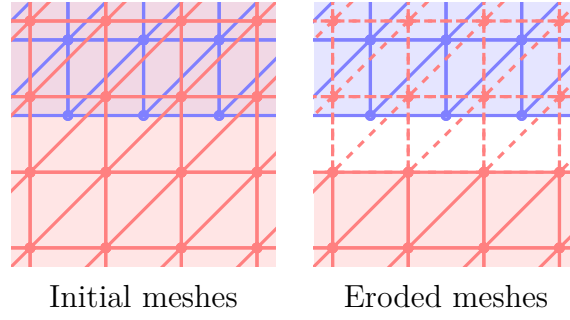


Figure 5.6: Illustration of mesh erosion. Initially, two meshes (one red and one blue) overlap. After erosion, the red mesh in the overlap area becomes a shadow mesh, denoted by dashed lines.

mesh. The idea was proposed in Hilton et al. (1996) and has widespread uses. Following Holz and Behnke (2013), the triangles are generated *adaptively*.

Triangles can be formed inside a *cell* which is made out of four neighboring depth map points (henceforth called vertices). A cell at depth map coordinates (x, y) consists of vertices v_{00} at (x, y) , v_{10} at $(x + 1, y)$, v_{01} at $(x, y + 1)$ and v_{11} at $(x + 1, y + 1)$. Edges are formed between vertices as follows: e_u between v_{00} and v_{10} , e_r between v_{10} and v_{11} , e_b between v_{01} and v_{11} , e_l between v_{00} and v_{01} , e_z between v_{10} and v_{01} , and e_x between v_{00} and v_{11} . An edge is valid only if both its vertices are valid and their distance is below a constant value d . The maximum edge length restriction acts as a simple mesh segmentation method, e.g. to ensure that two objects at different depths are not connected by a mesh.

Connected loops made out of edges form triangle faces. A cell can have six different triangle formulations as illustrated by Fig. 5.5. For example, the type 1 form is made out of edges e_u, e_z, e_l . However, ambiguity can arise when all possible cell edges are valid. In this situation, either type 3 or type 6 is selected depending on whether edge e_x or e_z is shorter. Since the triangles for a single cell can be stored in just one byte, this representation is highly compact.

Erosion

The initial generated meshes often cover the same surface area twice or more due to the overlap of RGB-D camera views. Mesh erosion detects redundant triangles in those areas;

more specifically, erosion labels all initial meshes to *visible* and *shadow* mesh parts. This labeling is based on the principle that overlapping areas should only contain one mesh that is marked visible. The remaining meshes are categorized as shadow meshes. In the previous mesh zippering methods, redundant triangles were simply deleted or clipped. In the proposed method, those triangles are kept in shadow meshes for later use in the mesh merging step.

Segmenting mesh into visible and shaded parts starts from the basic building block of a mesh: a vertex. All vertices are categorized as visible or as a shadow by projecting them onto other meshes. Next, if an initial mesh edge consists of at least one shadow vertex, the edge is considered a shadow edge. Finally, if a triangle face has a shadow edge, it is a shadow triangle.

Note that if each vertex was projected onto every other mesh, the result would only consist of shadow meshes and no visible meshes at overlap areas. Therefore, one mesh should remain visible. For this purpose, vertices are only projected to meshes with lower indices. For example, a vertex in mesh i will only be projected to mesh j if $i > j$.

Algorithm 2 Erosion process

```

1: for every vertex  $\mathbf{v}$  in meshes  $i \in \{1, 2, \dots, n\}$  do ▷ parallelized
2:   Initially label  $\mathbf{v}$  as visible vertex
3:   for every mesh  $j \in \{1, 2, \dots, i - 1\}$  do
4:      $p = \text{PROJECTVERTEXTOMESHSURFACE}(\mathbf{v}, j)$ 
5:     if  $\text{ISPOINTINSIDETRIANGLE}(p, j)$  then
6:       Label  $\mathbf{v}$  as shadow vertex
7:     end if
8:   end for
9: end for

```

Alg. 2 sums up the erosion algorithm. The $\text{PROJECTVERTEXTOMESHSURFACE}(\mathbf{v}, j)$ function projects a vertex \mathbf{v} to mesh j surface. This is possible because initial meshes are stored as 2D arrays in camera image coordinates. As such, the function simply projects the vertex to a corresponding camera image plane. $\text{ISPOINTINSIDETRIANGLE}(p, j)$ checks if coordinate p falls inside any triangle of mesh j . Fig. 5.6 shows an example of labeling a

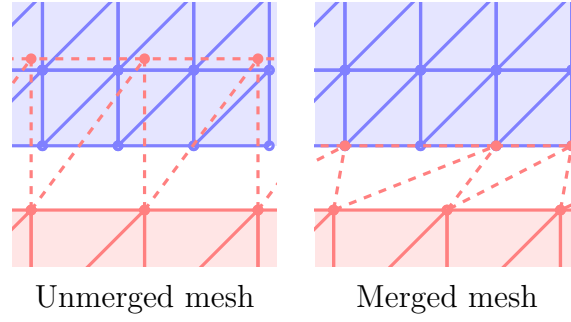


Figure 5.7: Illustration of mesh merging. Shadow mesh points are projected to other mesh (left) and then traced to the closest triangles for merging (right).

mesh into visible and shaded parts. There are visible gaps between the two meshes, but this can be rectified by following the meshing stages.

Mesh merging

The task of mesh merging is to find transitions from one mesh to another. For simplicity, consider that meshes A and B were to merge. If mesh A has a shadow mesh that extends over mesh B , then a transition from A to B exists. Such a situation can be seen on the left side of Fig. 5.7, where A would be the red mesh and B the blue mesh. In terms of notation, visible vertices are marked with V and shadow vertices are marked with S , e.g. v_S^A denotes the A mesh's shadow vertex.

Merging begins by going through all shadow vertices v_S^A . If a vertex is found that is joined by an edge to a visible vertex v_V^A , then this edge covers a transition area between two meshes. Such edges are depicted as dashed lines on the left side of Fig. 5.7.

Having located the correct shadow vertices v_S^A , the next task is to merge them with the v_V^B vertices so that the two meshes are connected. The end result of this is illustrated on the right side of Fig. 5.7. A more primitive approach of locating the nearest v_V^B to v_S^A would not work well, since the closest vertices v_V^B are not necessarily on the mesh boundary. Instead, an edge is traced from v_V^A to v_S^A until hitting the first B mesh triangle. The closest triangle vertex, v_V^B to v_V^A , will be selected as a match. Since meshes are stored as two-dimensional arrays, a simple drawing algorithm, such as a digital differential analyzer, can be used to trace from v_V^A to v_S^A .

The result of mesh merging procedure are edges connecting the two meshes. However,

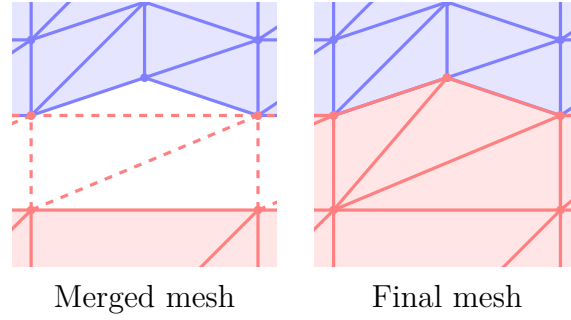


Figure 5.8: Illustration of final mesh generation. Merged mesh (left) might not have watertight connection between different camera meshes. This is rectified in final mesh (right), where extra triangles have been inserted to create watertight connections.

triangles have yet to be generated. This is addressed in the next and final mesh generation section.

Final mesh generation

The last part of the proposed meshing method collects all data from previous stages and outputs a single properly connected mesh. Handling triangles made out of visible vertices is simple, since they can simply be copied to output. However, transitions from one mesh to another require an extra processing step.

For simplicity’s sake, once again two mesh scenario is examined using notation introduced in Sec. 5.3.3. All the triangles in transition areas consist of one or two shadow vertices v_S^A , with the rest being visible vertices v_V^A . Triangles with just one shadow vertex can be copied to the final mesh without modifications. Triangles with two shadow vertices, however, are a special case. The problem lies in connecting the two consecutive v_S^A vertices with an edge. This situation is illustrated on the left side of Fig. 5.8. Specifically, the red mesh A ’s top shadow edge does not coincide with mesh B ’s edges. Therefore, a polygon is created that traces through B ’s mesh vertices v_V^B . To reiterate, the vertices of the polygon will be starting point v_V^A , the first shadow vertex v_S^A , mesh B ’s vertices v_V^B , the second shadow vertex v_S^A and the starting point v_V^A . This polygon is broken up into triangles, as illustrated on the right side of Fig. 5.8. Note that the polygon is not necessarily convex, but in practice, non-convex polygons tend to be rare and may be ignored for performance gains if the application permits small meshing errors.

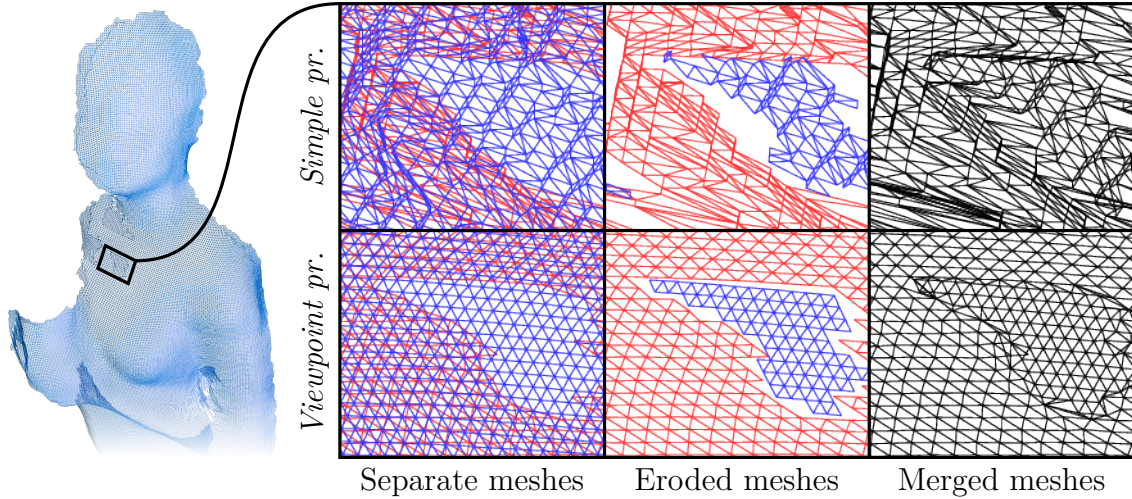


Figure 5.9: A mesh merging example. The proposed viewpoint projection method (lower row) can produce much higher quality meshes than simple projection (upper row). The columns from left to right show the merging process stages for two meshes (red and blue).

This concludes the stages of mesh generation. The results can be used in rendering or for any other application.

5.3.4 Performance

Table 5.1 gives an overview of the time spent on different system processes. The measurements were obtained with OpenGL query timers to get precise GPU time info. The experiment used four RGB-D cameras, all producing 512×424 resolution depth maps, resulting in up to 868k points per frame. The test was run with parameters shown in Table 5.2.

An overwhelming majority of processing time is spent on surface reconstruction. This is due to fetching a large number of points and normals from GPU memory. Nevertheless, as the data is retrieved in $s \times s$ square blocks from textures, the GPU cache is well utilized. The surface reconstruction was also implemented on the CPU for comparison reasons. The execution has been parallelized across 6 processor cores using OpenMP. The average runtime was 1.6 seconds per frame on the test dataset. This means that using a GPU gives roughly 10x the performance benefit over a CPU.

The proposed mesh generation method boasts better performance than competing

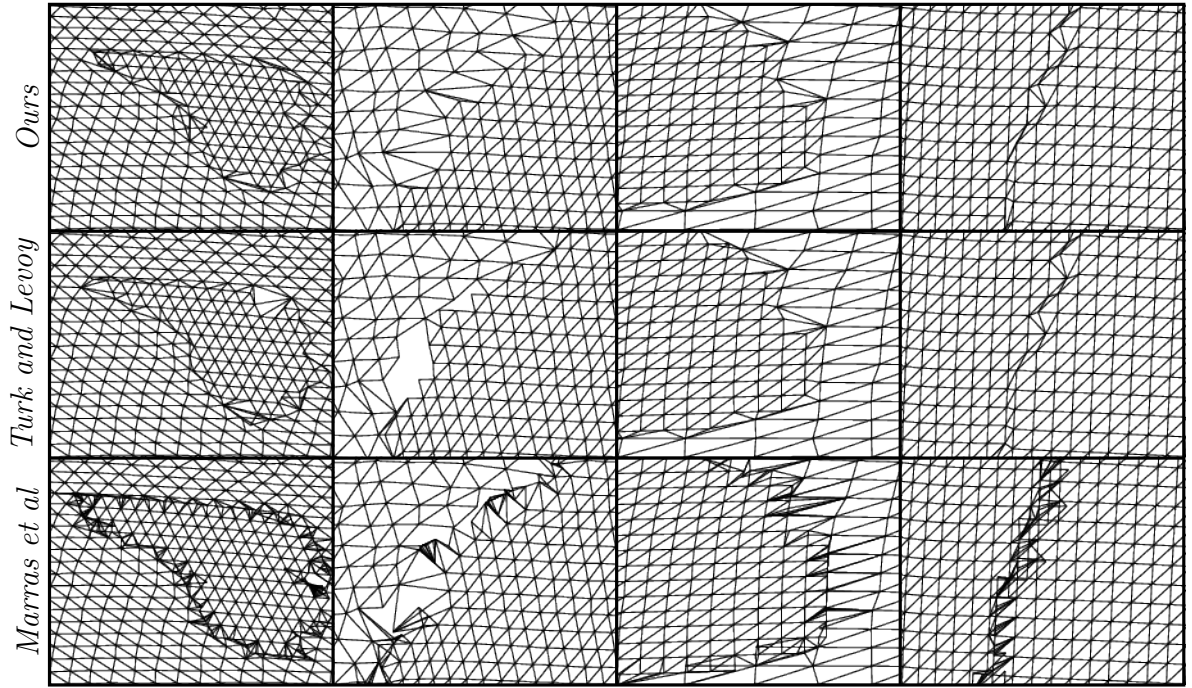


Figure 5.10: A comparison of mesh zippering methods at various randomly-chosen locations in a real scene. Turk and Levoy (1994) shows comparable meshing quality to the proposed method, but is CPU-based method and has limited speed. Marras et al. (2010) implementation produces excessive amounts of triangles in merger regions.

Table 5.1: ZipperMLS performance measurements

| Process | Avg. time | Max. time |
|-------------------------------|----------------|----------------|
| Motion estimation | 12.7 ms | 14.5 ms |
| Scene flow | 11.1 ms | 12.5 ms |
| Depth warping | 1.6 ms | 2.0 ms |
| Normal estimation | 2.4 ms | 2.7 ms |
| Surface reconstruction | 9.7 ms | 11.2 ms |
| Mesh generation | 0.43 ms | 0.49 ms |
| Initial mesh | 0.07 ms | 0.08 ms |
| Erosion | 0.12 ms | 0.13 ms |
| Merging | 0.04 ms | 0.05 ms |
| Final mesh | 0.20 ms | 0.23 ms |
| Total | 25.2 ms | 28.9 ms |

Table 5.2: ZipperMLS recommended parameter values

| Parameter | Value |
|--------------------------------------|------------------|
| Reconstruction | |
| MLS spatial smoothing factor | $h = 3\text{cm}$ |
| MLS window size | $u = 9$ |
| MLS maximum number of iterations | $i_{max} = 3$ |
| MLS number of camera frames used | $f_{num} = 4$ |
| Mesh generation | |
| Maximum allowed triangle edge length | $d = 3\text{cm}$ |

mesh zippering-based methods (Marras et al., 2010; Turk and Levoy, 1994). For the test dataset two initial meshes are generated as outlined in Sec. 5.3.3 and then the processes in Sec. 5.3.3-5.3.3 are compared with two previous methods. The Turk and Levoy (1994) implementation takes 48 seconds while the Marras et al. (2010) implementation takes over 9 minutes of execution time. These methods were originally designed for offline use on static scenes and thus focused on mesh quality rather than execution speed. These implementations are single-threaded CPU processes and cannot be easily parallelized due to algorithmic constraints outlined in Sec. 5.3.3.

Another major reason for the timing differences in zippering (Marras et al., 2010; Turk and Levoy, 1994) is the speed of point lookups. Previous methods are more general and can accept arbitrary meshes as input; they use tree structures such as k -d tree for indexing mesh vertices. The proposed method arranges meshes similarly to RGB-D camera depth maps. This allows for spatial point lookups by projecting a point to the camera image plane. This is much faster than traversing a tree.

Fig. 5.9 shows a comparison of meshes using different MLS projection methods. The simple viewpoint projection method produces very noisy meshes and no grid-like regularity is observed. The proposed viewpoint projection method, however, can organize points in a grid-like structure, making the result much higher quality.

Fig. 5.10 shows comparisons with previous mesh zippering research. Due to differences in the erosion process, the merger areas of meshes may end up in radically different

places depending on method used. Therefore, the developed erosion method was applied to force mesh mergers to appear in the same places for comparison. Turk and Levoy (1994) produce meshes with similar quality to this work. Marras et al. (2010) reference implementation tends to produce a high number of triangles in merger areas regardless of configuration parameters. An issue with this method is that its intended use is to fill holes in a mesh using another mesh. However, many test scenes are open and do not satisfy this requirement.

5.4 FusionMLS reconstruction method

FusionMLS is a volumetric 3D reconstruction method based on MLS. It can handle highly dynamic scene content and supports reconstructing large scenes. Next subsections give a detailed overview of the method.

5.4.1 Overview

Most volumetric reconstruction methods, especially ones developed using truncated signed distance fields (TSDF), work by accumulating data about scene to a reconstruction volume over time. This comes with several restrictions. First issue is that reconstruction volumes have to be stored in memory which can consume a lot of memory. Second and bigger issue is that dynamic content is not naturally supported in volumes. When objects move in scenes the respective volume voxel values have to be also relocated. Any mistake in tracking motions can result in wrong voxel value relocation which in turn causes corruption of the model. Errors can quickly accumulate and result in completely incorrect model. The solution presented by Dou et al. (2016) has been to reset the reconstruction volume after every few frames. A more extreme version of that idea is proposed by resetting the reconstruction volume on every frame. This comes with both problems and benefits.

From the problem side, TSDF method of accumulating voxel data over multiple frames is no longer possible. Results of TSDF reconstruction with only single frame of data show holes in models due to lack of depth map density. Also the surfaces are noisy as TSDF achieves smoothing by averaging incoming data over time. This can be counteracted by employing MLS smoothing techniques. MLS can handle much lower sampling density and

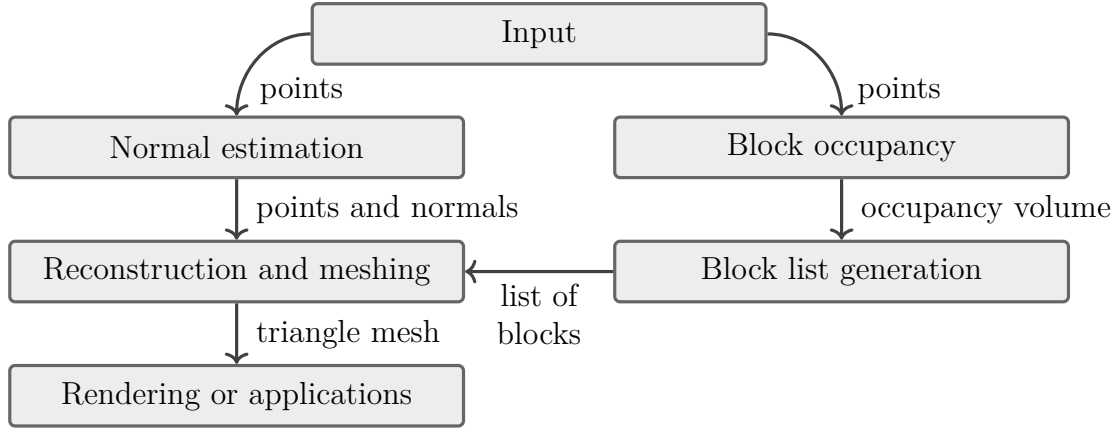


Figure 5.11: Overview of FusionMLS reconstruction process steps.

also fill smaller holes by interpolating point cloud data. Additionally, it has very good smoothing capabilities.

A good side of resetting reconstruction volume on every frame is that it is no longer necessary to keep reconstruction volume in memory for accumulating data. In fact it is possible to forgo storing volume at all by combining surface reconstruction and mesh generation processes. This results in both lower total memory usage and also considerably lower memory bandwidth.

An overview of the FusionMLS reconstruction process can be seen in Fig. 5.11. The input to the reconstruction is depth maps produced from motion compensation method described in Sec. 4. The points of depth maps are fed into normal estimation method and also block occupancy evaluator. The block occupancy system counts how many points occupy parts of the reconstruction volume called blocks. The result is stored as a low-resolution volume which is fed into block list generation. That process generates a list of blocks that contain one or more points and need to be reconstructed. The reconstruction takes both point cloud and initial normal data and uses the list of blocks to reconstruct the 3D model as a triangle mesh. That mesh is then used in simple rendering of the results or in some application.

5.4.2 Volume hierarchy

The proposed method belongs to volumetric reconstruction category, and the reconstructed scene area is defined as the reconstruction volume. This volume can be specified by the

user or calculated from the camera positions and their parameters. The smallest elements of volume are voxels, arranged in a grid-like fashion. A sub-volume of voxels of fixed size is called a *block*. It has a uniform number of voxels in all dimensions, for example, $8 \times 8 \times 8$.

There are two major reasons for dividing the total volume into blocks. The first reason is that a spatial hierarchy allows to determine the occupied volume regions, which need to be reconstructed. In other words, the need for expensive voxel value calculations is eliminated in areas where there are no depth map points and hence no surfaces. This method is simpler than octrees or kd-trees and can be computed quickly. The second reason concerns storing voxel values. It is preferred not to store voxel data in the GPU main memory as it is expensive and there is no use for this data when reconstructing the next frame. This method is more light-weight than other proposed volume memory reduction schemes (Chen et al., 2013; Steinbrücker et al., 2014). However, it is important to note that voxel values are still necessary to generate a mesh. The size of a block of voxels is reasonable to be stored temporarily. Furthermore, modern GPU features can be utilized in this scenario to speed up the processing.

Modern GPUs have multiple types of memory with different characteristics. Global memory has significant capacity and is persistent from allocation to deallocation but has slow access times. Shared memory, on the other hand, has a capacity of just tens of kilobytes and the data is not persistent over program execution, but it has much faster access times. According to GPU manufacturer documentation, recent GPUs such as those in the Nvidia GeForce range have roughly 100 times lower shared memory latency than uncached global memory. This can be leveraged to greatly accelerate 3D reconstruction. Typical volumetric reconstruction methods store per-voxel data in GPU global memory. This data needs to be read and written when estimating surfaces. Furthermore mesh generation also requires multiple lookups of the voxel values. In contrast, the proposed method is designed to use only shared memory to store voxel data.

Using the shared memory comes with certain restrictions. Most importantly, the data is stored only for the duration of GPU thread group execution. This means that after any voxel value calculation, the mesh generation must be immediately run on the same voxels. Additionally, the amount of shared memory is very limited, which in turn limits

maximum block size.

The mesh generation method, which uses the marching cubes algorithm, can only generate triangles between neighboring voxels. Essentially, every possible $2 \times 2 \times 2$ voxel sub-volume is processed to yield zero or more triangles. The marching cubes algorithm can be run inside a block of voxels but not at block edges. However, using the shared memory for block voxel storage means that it is not possible to look up the voxel values of neighboring blocks. This problem is solved similarly to method proposed by Steinbrücker et al. (2014) by making all blocks overlap each other by one voxel in each direction. As an example, if a block consists of $8 \times 8 \times 8$ voxels, then all blocks are laid out with spacing of seven voxels on all axes.

Because the voxel blocks overlap, some voxels are calculated several times as part of different blocks. The theoretical worst-case overhead can be calculated as $s^3/(s-1)^3$, where s is the size of the block in voxels. In the case of an $8 \times 8 \times 8$ voxel block, a 49% processing overhead is obtained. While the extra processing might seem considerable, the savings from not having to store and load voxels in the global memory is greater. However, block size s should be chosen with the balance of shared memory usage and processing overhead in mind.

5.4.3 Block occupancy

To determine which blocks of voxels are likely to contain surfaces, the number of points found in each block is counted. A three-dimensional array of integers is allocated such that there is one entry per block in the reconstruction volume. Assuming that a GPU with atomic operation support is used, all depth map points can be projected to the reconstruction volume. Atomic addition, `ATOMICADD(m, n)`, which adds the value n to some array location m , is used to sum the number of points in each block in parallel. Note that because the blocks overlap by one voxel on each side, it is necessary to detect when points are on edges and add them to other block counts as well. The procedure is summarized in Alg. 3. An example of finding the number of points inside the blocks is depicted in Fig. 5.12.

Next a list of non-empty blocks needs to be found so that they can be reconstructed.

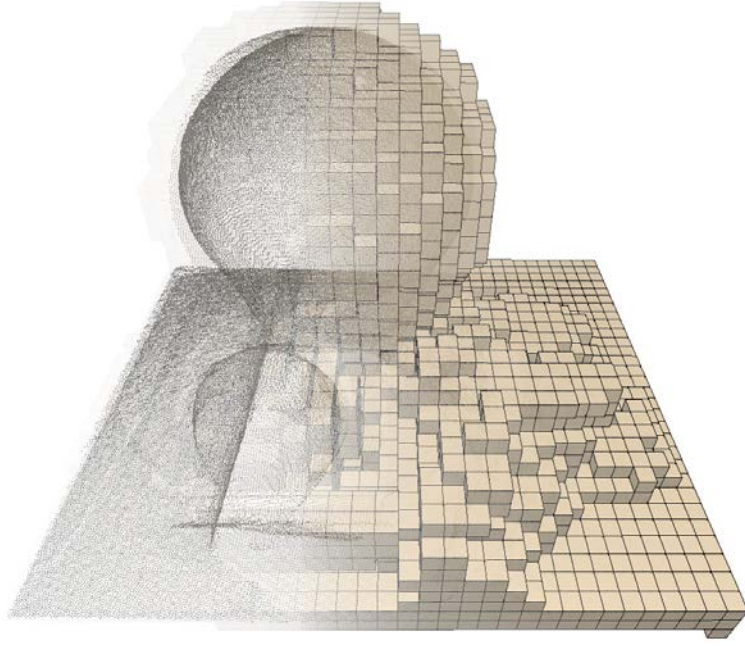


Figure 5.12: Visualization of blocks to be reconstructed. Left side shows the input point cloud and the right side shows the corresponding non-empty blocks.

For this each block is checked to determine whether the point count exceeds a constant threshold b_t . This threshold acts as a coarse point cloud filter. In practice, however, blocks tend to have relevant surfaces even at quite low point counts and hence setting $b_t = 1$ is recommended. Again atomic add operation is used to create a list of occupied block coordinates. Details are given in Alg. 4.

5.4.4 Reconstruction process

The main part of the scene reconstruction consists of estimating the surface geometry and generating the respective triangle meshes. The surfaces are defined implicitly using a signed distance function (SDF). The meshing algorithm can then find a zero-level set of SDF and output triangles.

SDF is estimated at each voxel center by sampling nearby points from all RGB-D cameras. State-of-the-art dynamic scene reconstructions using signed distance functions (e.g., Newcombe et al. (2015); Innmann et al. (2016)) typically use only one depth map point per camera to update single voxel values. This method works well only if the SDF is updated over multiple frames. Because the volume is not stored between reconstructions,

Algorithm 3 Block occupancy calculation

Require: camera points \mathbf{p} **Ensure:** occupancy volume V

```
1: reset occupancy volume  $V$  to zeros
2: for every point  $\mathbf{p}$  from cameras do ▷ parallelized
3:    $\mathbf{p} \leftarrow R\mathbf{p} + t$  ▷ transform point to volume
4:    $\mathbf{b} \leftarrow \mathbf{p} / (s - 1)$  ▷ block coordinates
5:   ATOMICADD( $V(\mathbf{b}), 1$ )
6:   if  $p_x \bmod (s - 1) = 0$  and  $b_x > 0$  then
7:     ATOMICADD( $V(\mathbf{b} - [1, 0, 0]^T), 1$ )
8:   end if
9:   if  $p_y \bmod (s - 1) = 0$  and  $b_y > 0$  then
10:    ATOMICADD( $V(\mathbf{b} - [0, 1, 0]^T), 1$ )
11:  end if
12:  if  $p_z \bmod (s - 1) = 0$  and  $b_z > 0$  then
13:    ATOMICADD( $V(\mathbf{b} - [0, 0, 1]^T), 1$ )
14:  end if
15: end for
```

this approach does not suit the needs of this work. It is preferred to estimate the local surfaces using MLS, which samples many points in the neighborhood of the voxel.

To estimate the local surface, depth points \mathbf{p}_i first need to be retrieved together with corresponding initial surface normals \mathbf{n}_i from the neighborhood of a given voxel center \mathbf{x} . Similarly to Kuster et al. (2014), point \mathbf{x} is projected to each RGB-D depth map to retrieve a $u \times u$ square block of depth points \mathbf{p}_i and normals \mathbf{n}_i around the projected point.

For the actual surface estimation, the moving least squares formulation put forward in Sec. 5.2.1 is used with SDF estimated using Eq. 5.16. In addition a voxel *confidence* value is defined simply as

$$c(\mathbf{x}) = \sum_i w(\|\mathbf{x} - \mathbf{p}_i\|), \quad (5.20)$$

where the sum is taken over points in the local neighborhood in the same way as calculating

Algorithm 4 Block list generation

Require: occupancy volume V **Ensure:** block index B

```
1:  $i \leftarrow 0$ 
2: for every block coordinate  $\mathbf{b}$  do ▷ parallelized
3:   if  $V(\mathbf{b}) \geq b_t$  then
4:      $j \leftarrow \text{ATOMICADD}(i, 1)$ 
5:      $B(j) \leftarrow \mathbf{b}$ 
6:   end if
7: end for
```

implicit function f .

In cases where the confidence value $c(\mathbf{x})$ is below a constant user-specified threshold c_t , the value $f(\mathbf{x})$ is marked as invalid. This effectively removes the surfaces that do not have enough points for an accurate estimation. The SDF value $f(\mathbf{x})$, the normal $n(\mathbf{x})$, and the confidence value $c(\mathbf{x})$ are stored for each voxel. The normal is stored such that there is no need to estimate the surface normal again during meshing. Also the confidence value can be used to generate smooth object edges during rendering.

To generate the mesh, the marching cubes triangulation Lorensen and Cline (1987) is used. Every possible $2 \times 2 \times 2$ voxel sub-volume of the block is passed to triangulation. The marching cubes method decides what triangles to create between each voxel based on SDF values. If any values $f(\mathbf{x})$ are found marked as invalid, then no triangles are outputted. All valid triangles are written to a global buffer and include a point location \mathbf{x} , a normal $n(\mathbf{x})$, and a confidence $c(\mathbf{x})$ attribute for each vertex.

Both surface estimation and mesh generation are summarized in Alg. 5. The major steps of the reconstruction are also visualized in Fig. 5.13.

5.4.5 Rendering

The 3D mesh can contain discontinuities at the edges of thin objects or due to limited depth map coverage of the scene. A naive rendering of such areas will result in jagged edges. This is because marching cubes has no native way of handling discontinuities.

Algorithm 5 3D reconstruction and mesh generation

Require: block index B **Ensure:** triangle mesh M

```
1:  $i \leftarrow 0$ 
2: for every block  $\mathbf{b}$  in index  $B$  do ▷ parallelized
3:   ▷ MLS reconstruction
4:   for voxels  $\mathbf{c} \in [0, 1, \dots, s]^3$  do ▷ parallelized
5:      $\mathbf{p} \leftarrow R(\mathbf{c} + s\mathbf{b}) + T$  ▷ global coordinates
6:      $F(\mathbf{c}), N(\mathbf{c}) \leftarrow \text{MOVINGLEASTSQUARES}(\mathbf{p})$ 
7:   end for
8:   ▷ Marching cubes triangulation
9:   for voxels  $\mathbf{c} \in [0, 1, \dots, s-1]^3$  do ▷ parallelized
10:     $m \leftarrow \text{MARCHINGCUBES}(F(\mathbf{c}), N(\mathbf{c}))$ 
11:    for all triangles  $t \in m$  do
12:       $j \leftarrow \text{ATOMICADD}(i, 1)$ 
13:       $M(j) \leftarrow t$ 
14:    end for
15:  end for
16: end for
```

However, the confidence value $c(\mathbf{x})$ of the vertices can be used to smoothly cut off triangles at a user-defined threshold c_r . Optionally, the edges can be smoothly made transparent using alpha blending together with an order-independent transparency technique such as depth peeling Everitt (2001). Different ways of handling edge rendering are visualized in Figure 5.14.

5.4.6 Performance

All of the experiments were conducted on a server with Intel i7-5930K CPU, 32 GB of RAM, and an Nvidia GeForce GTX 1080 Ti graphics card. The client computers are Intel NUC7i3 machines with Intel i3-7100U CPU and 16 GB of RAM. As for cameras, Microsoft Kinect 2 RGB-D devices were used.

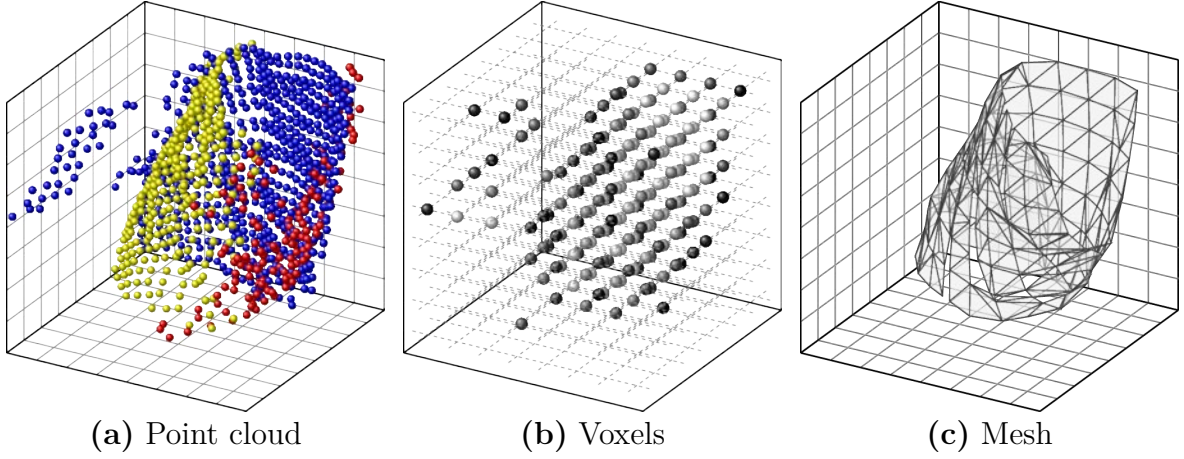


Figure 5.13: Visualization of reconstruction of an $8 \times 8 \times 8$ voxel block using real-world data. (a) Input point cloud data; colors mark different RGB-D cameras; (b) MLS voxel values calculated at center of voxels with only negative voxel values visualized; (c) meshing result after marching cubes triangulation. Note that the outlier points on the left side of the block are successfully excluded from the final result.

Table 5.3: FusionMLS performance measurements

| Process | Avg. time | Max. time |
|-------------------------------|----------------|----------------|
| Motion estimation | 12.7 ms | 14.6 ms |
| Scene flow | 11.1 ms | 12.9 ms |
| Depth warping | 1.6 ms | 1.7 ms |
| Normal estimation | 2.3 ms | 2.7 ms |
| Surface reconstruction | 16.8 ms | 17.7 ms |
| Block occupancy | 0.07 ms | 0.06 ms |
| Reconstruction | 16.7 ms | 17.6 ms |
| Total | 31.8 ms | 35.0 ms |

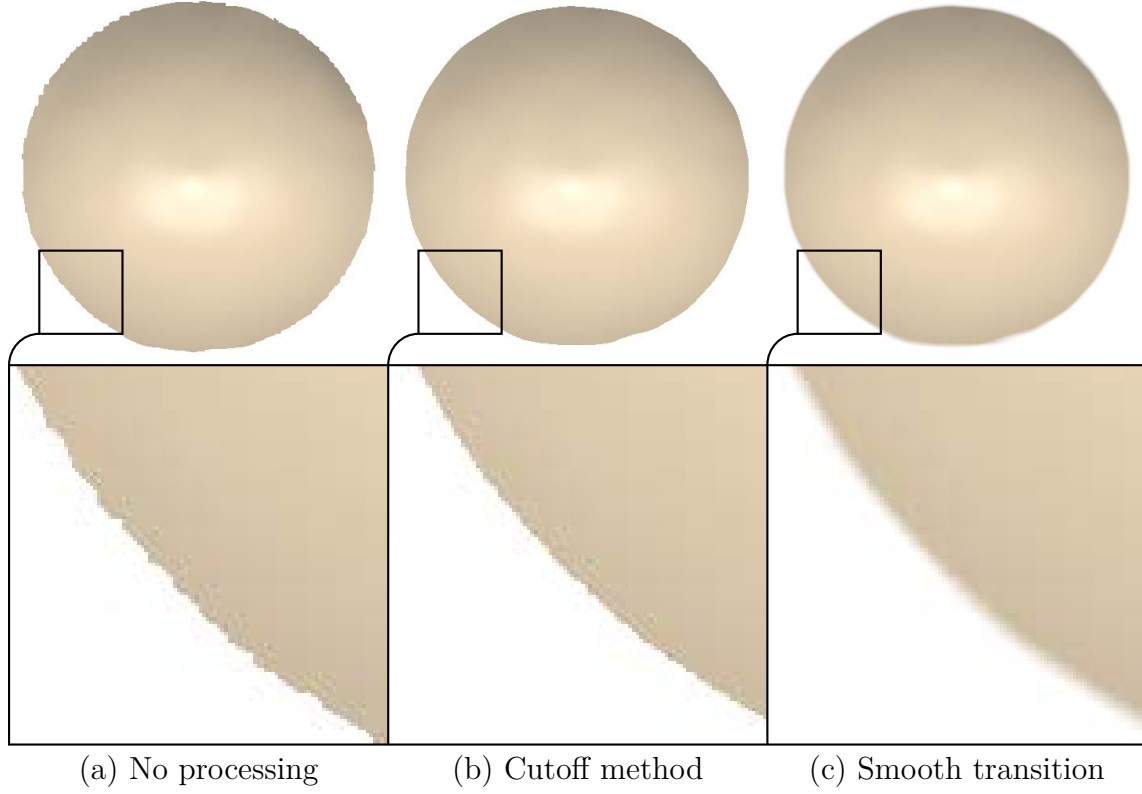


Figure 5.14: Visualization of edge rendering methods. (a) The mesh is rendered as is; (b) mesh triangles are cut off at a user-defined confidence value; and (c) mesh triangles are smoothly transitioned from opaque to transparent based on confidence value.

Table 5.4: FusionMLS recommended parameter values

| Parameter | Value |
|------------------------------|----------------------|
| Motion estimation | |
| Mesh segmentation threshold | $m_t = 1.5\text{cm}$ |
| Mesh layers | $n = 4$ |
| Reconstruction | |
| Number of voxels in volume | 2×10^7 |
| Block size in voxels | $s^3 = 8^3$ |
| Minimum points in block | $b_t = 1$ |
| MLS spatial smoothing factor | $h = 4\text{cm}$ |
| MLS window size | $u = 11$ |
| Confidence value threshold | $c_t = 30$ |

The system performance characteristics for a typical scene recorded with four RGB-D cameras can be seen in Tab. 5.3. Because the proposed pipeline is completely executed on the GPU, precise statistics for each process step can be obtained through OpenGL timer queries. The system parameters given in Tab. 5.4 were tuned to obtain maximum reconstruction quality while retaining a real-time frame rate.

5.5 Results

While there are plenty of RGB-D camera datasets available (Firman, 2016), almost none of them use multiple sensors simultaneously in a dynamic scene. In experiments following datasets were used:

- Public dataset by Alexiadis et al. (2018). All scenes contain four Microsoft Kinect 2 devices based on time-of-flight technology. Scenes contain human actors recorded from all sides performing various movements. Time calibration accuracy is roughly 15ms which can result in some inaccuracies when reconstructing fast movements.
- Private dataset by Kuster et al. (2014). All scenes use two Asus Xtion cameras based on structured light technology. Scenes contain human actors recorded only from one side. Time calibration accuracy is unknown, but scenes only contain slow movements which do not impact reconstruction quality.
- An original dataset captured using multiple Microsoft Kinect 2 devices. It contains recorded scenes of both human actors and various objects interacting.

Multiple datasets were used to demonstrate the general nature of the reconstruction methods. Small and large scale scenes with abstract objects, highly-deformable objects like cloth and human actors are represented. Both structured light and time-of-flight technology devices are used to show resilience to various types of noise.

An important aspect of the proposed reconstruction systems is the ability to handle scenes with significant dynamic content. This includes fast-moving objects as well as changing scene topology. Fig. 4.1 shows the effectiveness of the approach. A selection of challenging situations are shown in Fig. 5.15. Rapid movements of objects are correctly reconstructed thanks to the depth frame interpolation method.

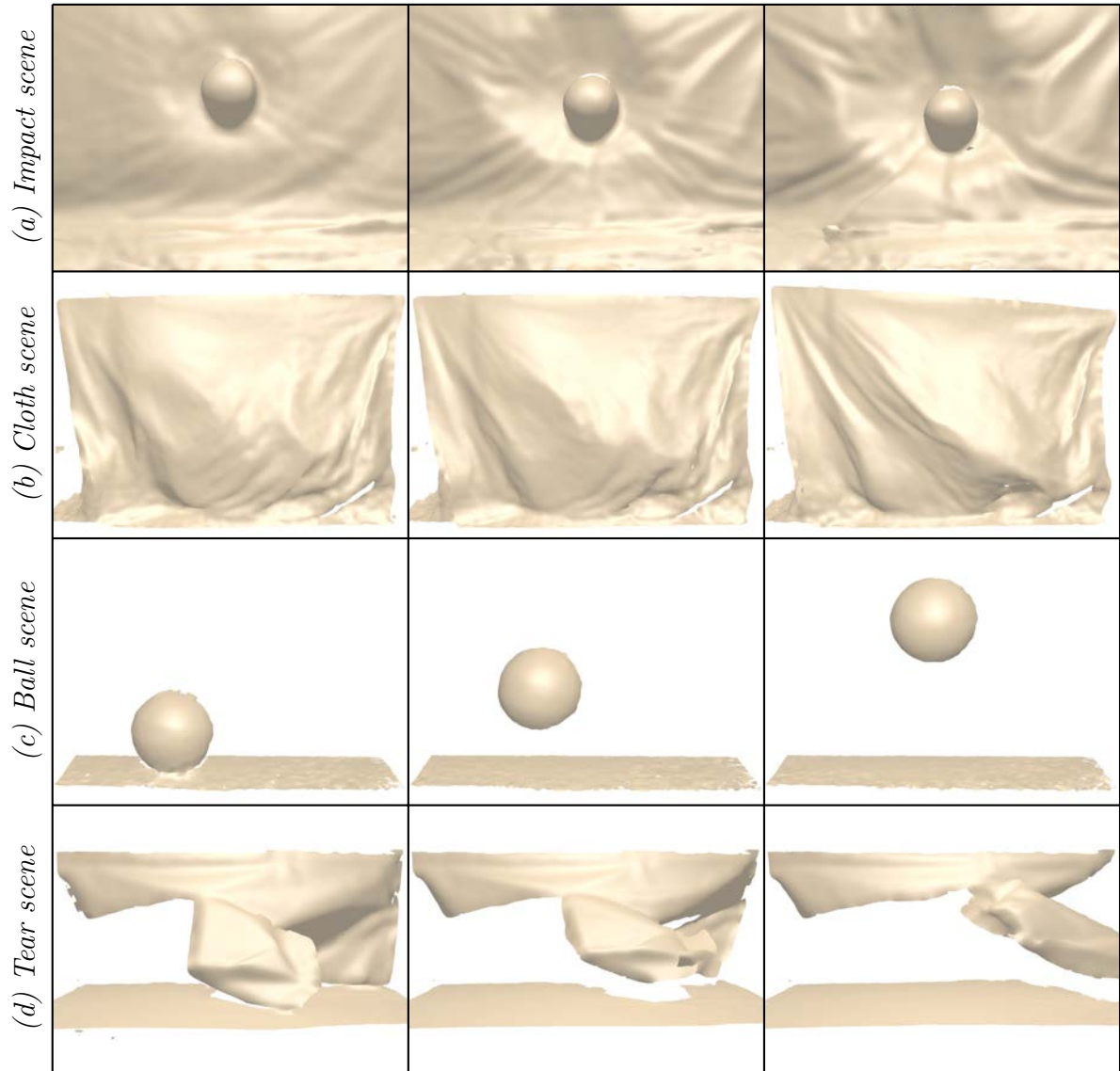


Figure 5.15: Reconstructing highly dynamic scenes with FusionMLS: (a) A ball is thrown at a cloth curtain; (b) a cloth is shaken; (c) a ball bounces off the ground; and (d) a large piece of paper is torn apart.

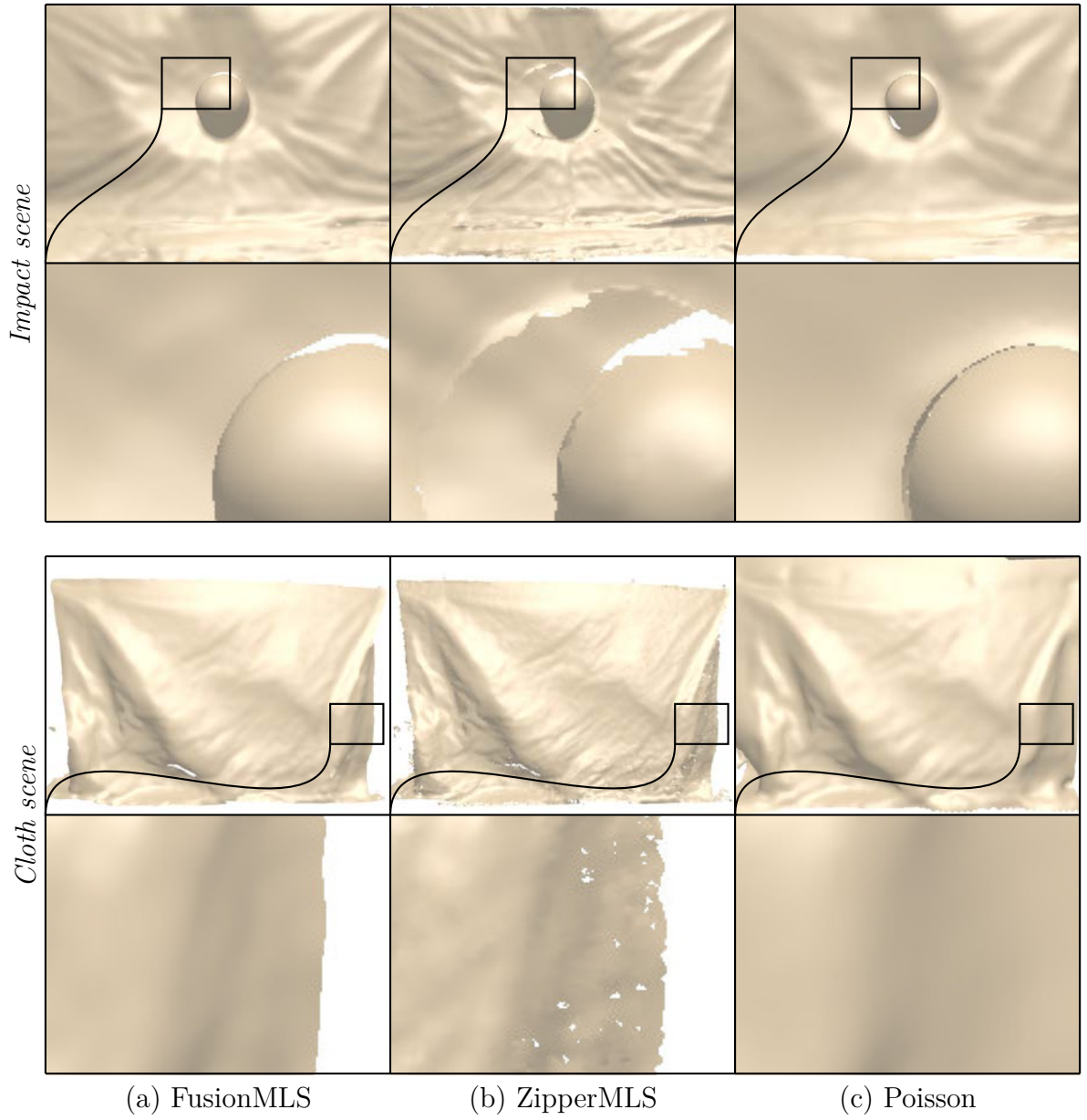
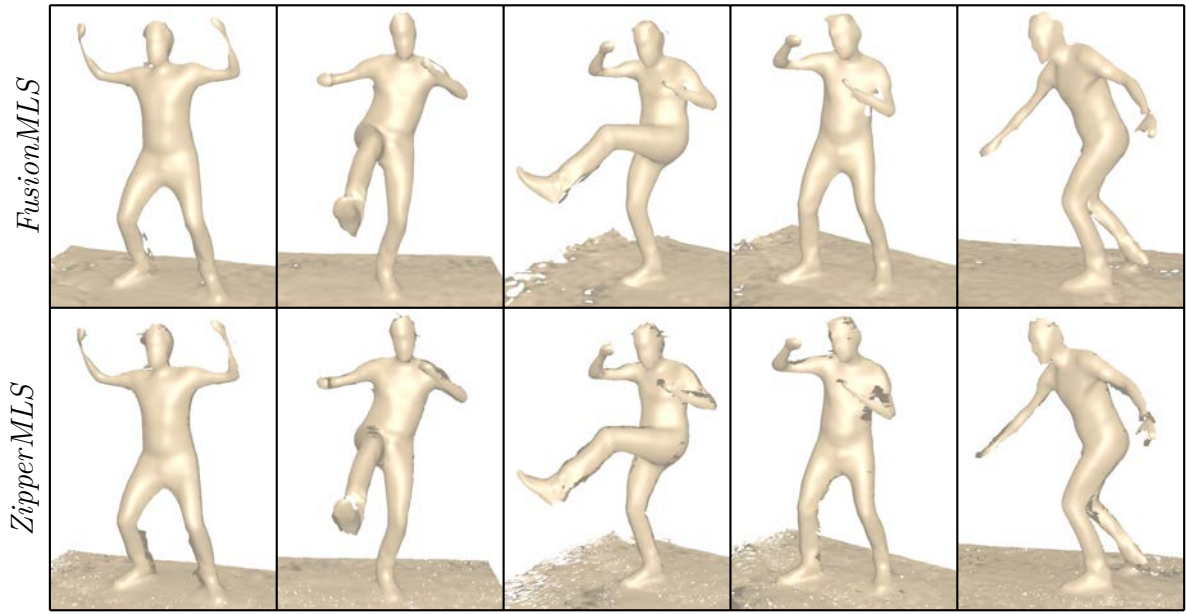
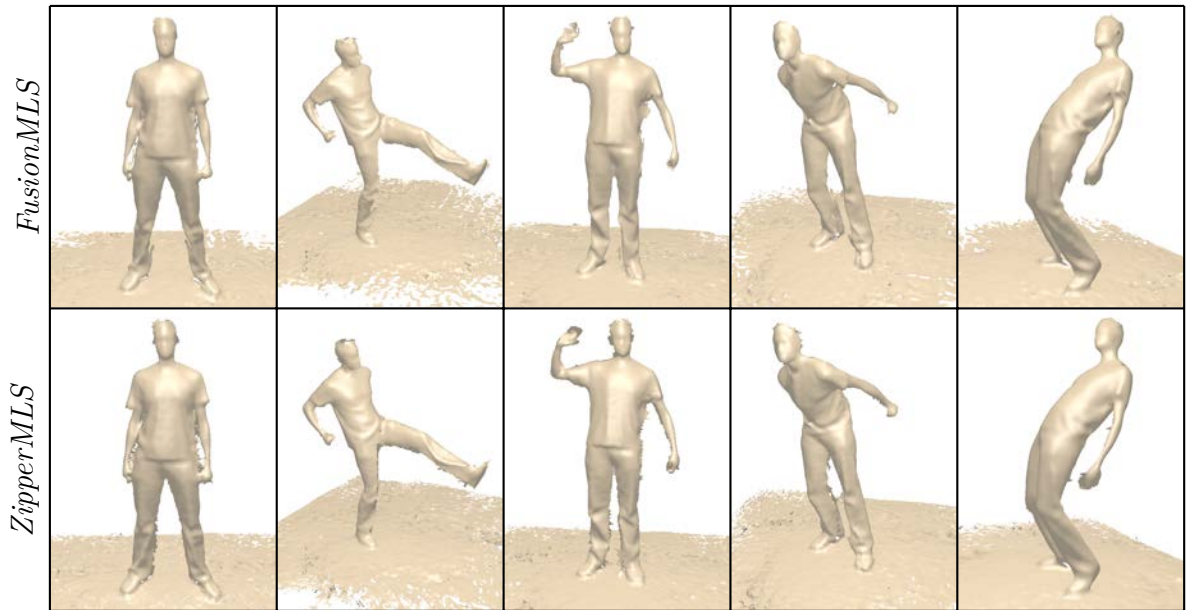


Figure 5.16: Comparison of 3D reconstruction methods: (a) shows proposed FusionMLS method, (b) uses proposed ZipperMLS method, and (c) shows the Poisson reconstruction method Kazhdan et al. (2006). Zippering shows bad edge quality and occasionally has incorrect triangles protruding from surfaces. The Poisson method tends to over-smooth areas and incorrectly handles open scenes.

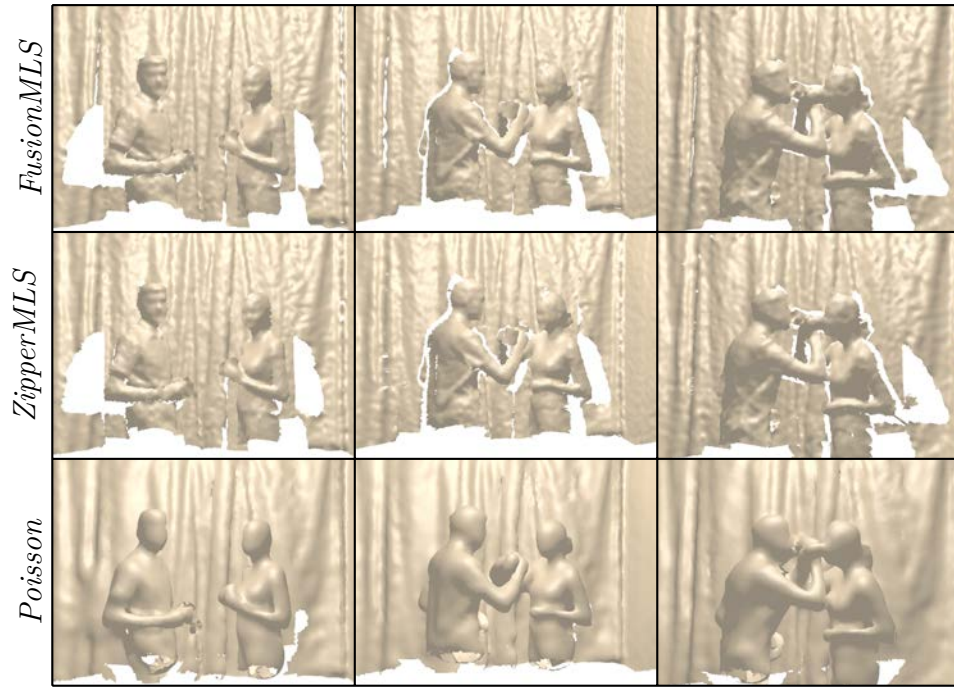


‘Alexandros’ scene

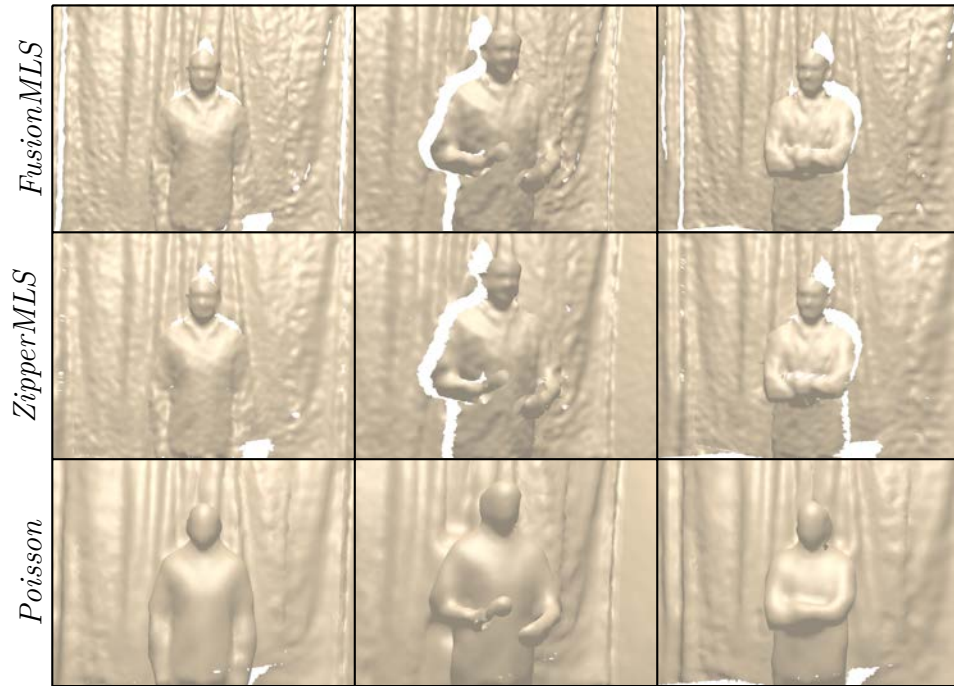


‘Apostolakis1’ scene

Figure 5.17: FusionMLS and ZipperMLS reconstruction results of human actors in various poses. Two different sequences captured with four RGB-D cameras are shown. The captured data is courtesy of Alexiadis et al. (2018). In general, the models are correctly reconstructed. However, some smaller details are missing due to lack of visibility from cameras.



Scene 99



Scene 102

Figure 5.18: Reconstruction results of human actors with scene backgrounds. The capture data is courtesy of Kuster et al. (2014). Both proposed reconstruction methods, FusionMLS and ZipperMLS, are shown together with Poisson reconstruction (Kazhdan et al., 2006) for reference.

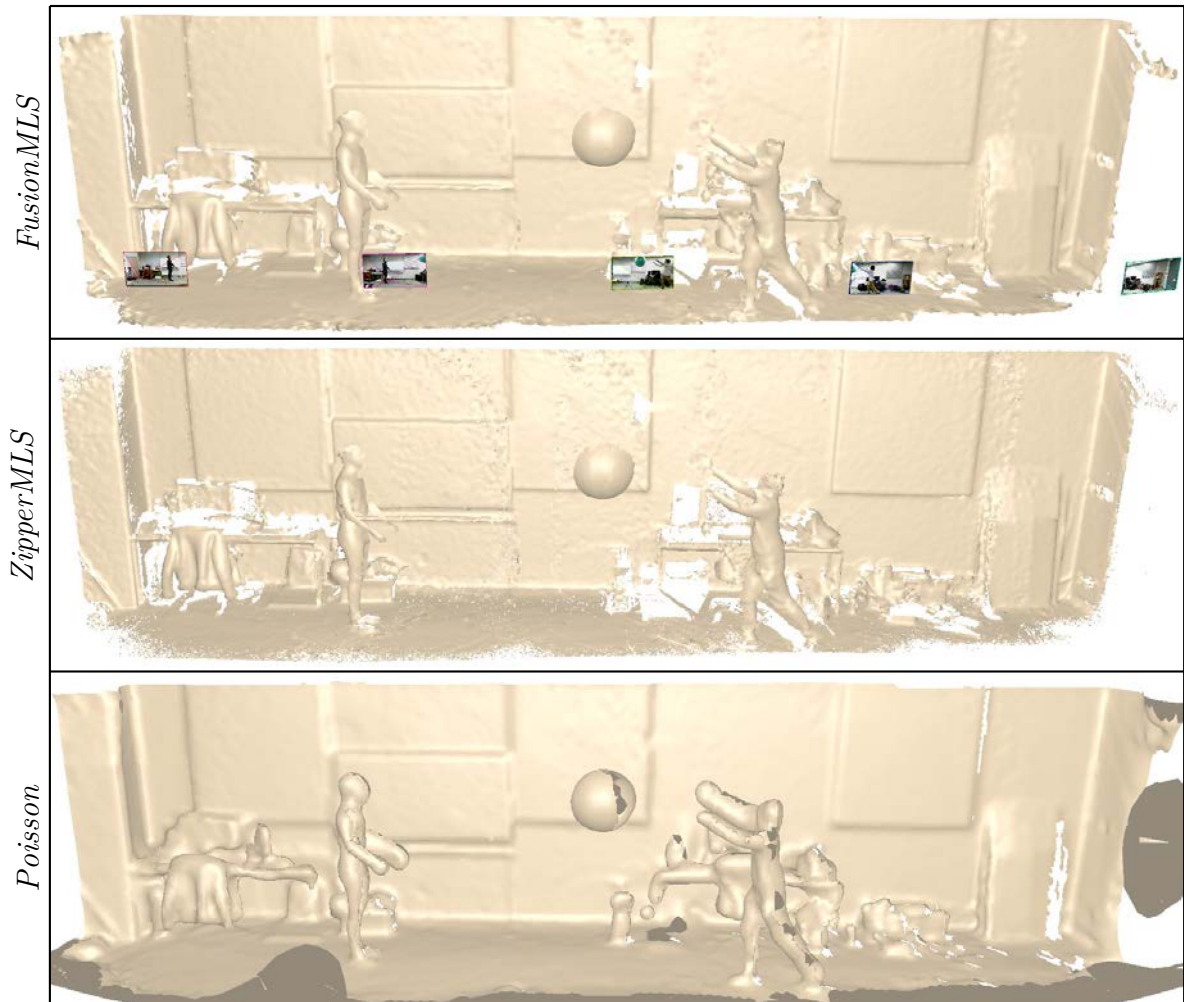


Figure 5.19: Reconstructing fast ball throw in a large scene with five RGB-D cameras. Both proposed reconstruction methods, FusionMLS and ZipperMLS, are shown together with Poisson reconstruction (Kazhdan et al., 2006) for reference. The FusionMLS results also contain images depicting camera positions.

Human actor reconstruction can be seen in Fig. 5.17 and Fig. 5.18. The source data for the first figure is courtesy of Alexiadis et al. (2018) and data for the second figure was provided by Kuster et al. (2014). The scene in Fig. 5.17 is set up so that cameras capture human activity from four viewpoints. The resulting human model has good quality where visibility from cameras is good. However, as the reconstruction method is designed to be very general, human templates are not used to fill missing surface areas. The scene shown in Fig. 5.18 uses only two RGB-D cameras which also have higher level of noise. Understandably, this results in somewhat lower reconstruction quality than shown in Fig. 5.17.

Large scenes can also be reconstructed with the proposed methods in real time. Fig. 5.19 shows a room with a size of $10 \times 3 \times 3$ m. The scene can be reconstructed in real time mostly due to the block occupancy test, which avoids the need to reconstruct empty spaces in the scene. Moreover, because the reconstruction volume is not stored, the memory cost is low. For a volume made out of 2×10^7 voxels, only block occupancy, taking 153kB, and block index data, taking up to 457kB of memory, needs to be stored. Rest of the memory usage is related to input depth map and normals, and output triangle mesh.

The number of systems that can be used for comparison is limited. Recent dynamic scene reconstruction methods, especially ones that utilize truncated signed distance volumes, require fusing depth data over multiple frames and are not designed to handle cameras with unsynchronized shutters. In addition, the selection of comparison methods is restricted as the recorded scenes contain backgrounds and highly non-rigid objects, such as cloth, for which template generation is very difficult.

The proposed 3D reconstruction methods were compared with Poisson surface reconstruction Kazhdan et al. (2006) in Fig. 5.16, 5.18 and 5.19. The Poisson method is CPU-based and takes around 5 to 10 seconds to execute depending on scene (with reconstruction depth parameter set to 8). It was chosen for its ability to reconstruct entire scenes from only one depth frame per RGB-D camera. Results have signs of oversmoothing whereas the MLS-based methods manage to preserve small details. An upside of Poisson reconstruction is its ability to complete a surface even when point cloud data is missing from some scene area. While this feature can be beneficial in repairing some areas of the

generated mesh, it also has drawbacks. First, real-world scenes tend to be topologically open and have many boundaries. These areas are incorrectly reconstructed by the Poisson method. Second, repairing surfaces is an under-determined problem and holes in geometry can be filled in various ways. This results in temporally inconsistent surfaces.

In summary, both ZipperMLS and FusionMLS can reconstruct various real-life scenes containing dynamic content in both small and large scales. In terms of quality, ZipperMLS may occasionally have rougher edges at surface discontinuities compared to FusionMLS. This is due to ZipperMLS lacking smooth edge rendering feature, introduced in Sec. 5.4.5, which is present in FusionMLS. Additionally, joining meshes can fail in some complex geometry areas, resulting in small holes in the meshes or incorrectly generated triangles protruding from surfaces. FusionMLS is more robust to complex geometry areas, but can result in some loss of details. This is due to ZipperMLS generating meshes at the same density as input point clouds whereas FusionMLS defines voxel grid that is generally more coarse to achieve better execution performance.

5.6 Discussion

Both proposed ZipperMLS and FusionMLS can be used to reconstruct various scenes. However, since the methods are different, there are some tradeoffs in their performance. Recommendation of which method to use is based on the camera setup. There are two basic scenarios for selection of the method:

- **Camera views have lot of overlap such as capturing some object from multiple sides.**

In this scenario FusionMLS is likely a better choice. The main reason is that in this scenario the point cloud can typically have very wildly varying density. In FusionMLS the density of reconstruction and meshes can be controlled by selecting the resolution of the voxel grid. Therefore FusionMLS scales better when dealing with very dense point clouds.

- **Camera views have little overlap such as capturing a large room.**

In this scenario ZipperMLS is likely a better choice. Since there is little overlap

between camera views, most scene areas contain single mesh. There ZipperMLS point refinement-based reconstruction has lower memory cost and higher speed than volumetric FusionMLS method. Therefore ZipperMLS scales better in sparser scenes.

It is also possible to characterize differences in the methods in terms of input point cloud properties. In general, both ZipperMLS and FusionMLS use same underlying moving least squares based reconstruction method. Hence both methods have many similarities. Differences, however, appear when also considering how triangle meshes are generated. Berger et al. (2017) compared 3D reconstruction methods in terms of input point cloud properties. The same list of comparison terms can be used to compare ZipperMLS and FusionMLS methods:

- **Ability to handle nonuniform sampling of points.**

Both methods can deal with nonuniform point sampling, but handle it differently. ZipperMLS has the ability to adapt the density of the output mesh to the density of input point cloud whereas FusionMLS has constant mesh density based on voxel grid resolution. Which result is preferred can depend on the usage scenario.

- **Robustness to noise.**

The handling of noise is practically identical between the methods due to the use of same MLS smoothing.

- **Robustness to outliers.**

Generally, both methods filter outliers using same MLS machinery. However, when MLS fails to correctly exclude outliers, ZipperMLS tends to have better quality. This is because FusionMLS can slightly interpolate and extrapolate surfaces, making outlier surfaces larger.

- **Ability to handle multiple misaligned point clouds.**

Both methods can handle slightly misaligned point clouds again using the same underlying MLS smoothing. Large misalignments are not detected and appear as distinct surfaces.

- **Ability to complete shapes when point cloud data is missing.**

FusionMLS can interpolate and extrapolate surfaces up to the MLS smoothing weight radius distance. ZipperMLS does not have this capability and strictly generates surfaces between existing points.

Chapter 6

Applications

3D reconstruction is only meaningful if it can be applied in real-world problems. Here some research areas are explored that directly make use of reconstruction methods and demonstrate a diminished reality application in depth. In all of those systems the 3D reconstruction method plays a crucial role and typically its performance has the biggest impact on the quality of results.

6.1 Overview

Telepresence has been a popular choice when showcasing the capabilities of reconstruction methods. Initially, the reconstruction results were shown using 3D displays (Plüss et al., 2016; Maimone and Fuchs, 2011). Recently, the move has been to completely immerse users in the environment using head-mounted displays (HMD) (Orts-Escolano et al., 2016).

Mixed reality, particularly augmented reality (AR) and diminished reality (DR) sub-fields, have also made use of 3D reconstruction. Traditionally, AR systems added virtual objects to scenes with the use of see-through displays which meant that full 3D reconstruction of the surroundings was not necessary. Recently, however, Lindlbauer and Wilson (2018) demonstrate an augmented reality solution that completely reconstructs scenes. This allows much more flexible manipulation, i.e. augmentation, of the scenes.

The objective of diminished reality is to make some objects in scenes invisible to human user so that it would be possible to see behind them. For a recent overview of diminished reality methods, please refer to Mori et al. (2017).

Some examples of DR use cases are as follows. In sports applications Hashimoto et al. (2010) diminish catcher in baseball game to obtain a good view of the pitcher and Sakai et al. (2018) create a sports game using head mounted displays and use diminished reality to make the environment look as a video game. Another popular use case is operating machinery. For example, Rameau et al. (2016); Yoshida et al. (2008) hide a part of car to show what is in front of it for safety reasons and Sugimoto et al. (2014) diminish robot arms to enhance visibility of objects to be manipulated. DR can also be used when creating teaching tools. Yokoi and Fujiyoshi (2006) remove lecturer in classroom from user view to expose whiteboard and Ienaga et al. (2016) show human anatomy learning application using diminished reality. Another interesting direction of DR research is to use it in home remodeling. Siltanen (2017) show hiding and replacing furniture in a room to help visualize results of home remodeling.

Various approaches to diminished reality exist, but to get accurate DR results in dynamic scenes, multiple cameras need to be used which capture scene in real time. In practice this requires reconstructing scenes so that objects to be diminished could be overlaid with rendered background images. The developed 3D reconstruction methods are perfect for this application as they work in real time, can handle dynamic content with any topology changes and do not require object shapes to be known a priori. Hence, in the next section, the effectiveness of the proposed 3D reconstruction methods are demonstrated in a diminished reality application.

6.2 Diminished reality application

A diminished reality system is developed on top of the proposed 3D reconstruction methods. The input to the diminished reality method is a reconstructed triangle mesh and an image of the target view where some objects need to be removed. The first step is to estimate pose of the object to be diminished. After that, it is possible to cull the diminished object from the mesh and render it at target image viewpoint. Finally, reconstruction and target view images are composed into a single result. This requires both adjusting colors of images to blend them seamlessly and filling in small missing details of reconstruction.

6.2.1 Object pose estimation

The positions of the cameras in test scenes have been calibrated as described in Sec. 3.3. In DR applications there exists an extra pose estimation problem. Namely, the position of objects to be diminished should be determined in real time. The simplest method of diminishing objects would be to define a 3D space region where all objects are diminished. In reality, however, it might be necessary to diminish objects that are moving around. Hence, methods to track their movement need to be developed.

Two object tracking techniques are presented in this work. First is tracking objects with AR markers (Garrido-Jurado et al., 2014). It is assumed that the shape of an object is known a priori and it includes a marker with known position. The benefit of this method is that it is possible to diminish objects very reliably with low computational cost and it is easy to change objects that are diminished. A second method is designed to track objects that have certain shape and appearance. In particular, diminishing a ball that does not have any other features than circular shape and color is demonstrated.

6.2.2 Rendering

The reconstructed scene needs to be rendered so that objects to be hidden can be covered with background images. The issue here is that naive rendering of the whole scene would also include the objects that need to be diminished. Hence a way to remove objects from reconstruction is developed as well. This can be done in three different ways.

First option is to remove depth data from all RGB-D cameras around the area of object to be diminished. The primary issue here is that it would cause issues with texturing in rendering phase. The color images from RGB-D cameras cannot directly be mapped to 3D models – occlusions need to be checked first. By removing some parts of the models, the occlusion checks would be disrupted.

Second possibility is to delete parts of the reconstructed mesh to hide objects. This can become somewhat expensive operation if the object to be diminished has complex shape. Namely, it is necessary to iterate over all mesh vertices and check whether they fall into the hidden region or not.

Third method is to create a depth mask for rendering to exclude objects from final

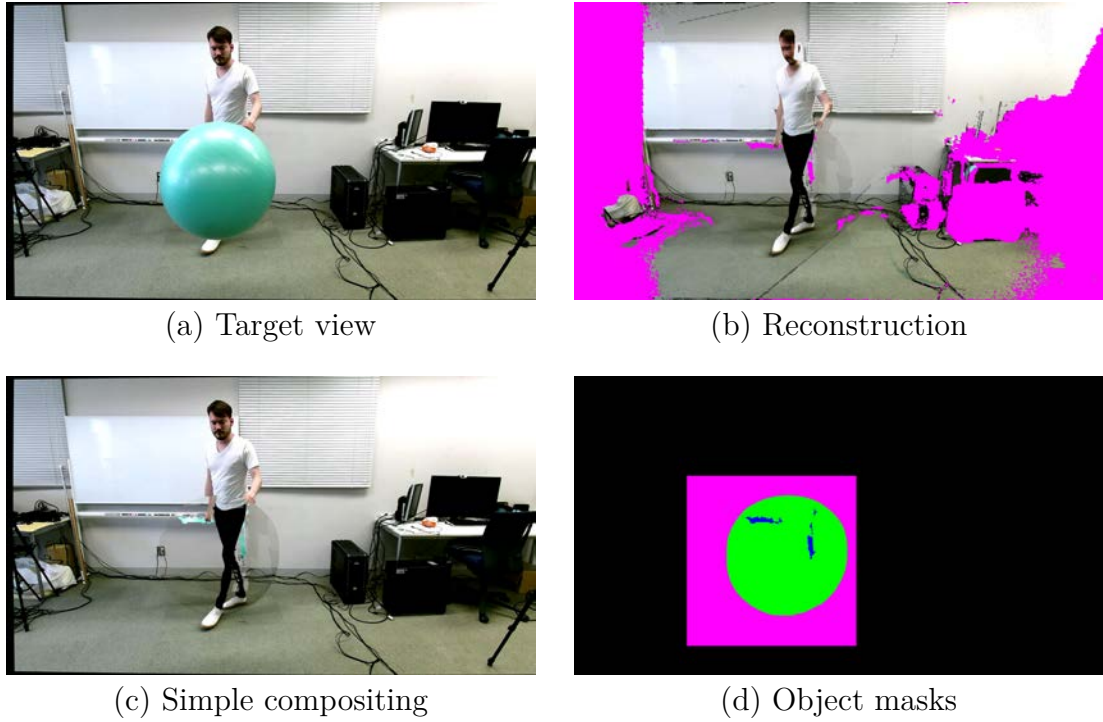


Figure 6.1: Visualization of the composition input images and region masks: (a) shows target view where a spherical object needs to be diminished, (b) shows reconstructed scene background with missing areas in magenta color, (c) shows simple compositing of reconstruction and target views and (d) shows scene mask where diminished object region is given in magenta, the detected object to be diminished is given in green and finally missing background reconstruction areas are marked with blue.

result. This approach works by first rendering the hidden region as a 3D model to a so called z-buffer. Typically z-buffer contains depth values of rendered objects. A small modification is made here to record not the nearest depth value of an object but the farthest value. What it means is that when the 3D scene model is rendered it can be easily checked if any parts of the scene falls in the diminished object region. it is preferred to use this method as it is computationally cheap and does not depend on the complexity of the diminished object shape.

6.2.3 Compositing

The objective of compositing is to cover regions of objects to be diminished with background images. Fig. 6.1 shows input images for this process. The scene background has been rendered at the viewpoint of the target view. The object to be diminished, in this case a spherical object, has been highlighted in the object masks with green color.

To compose the input images, two masks are first generated. The first mask, called valid pixel mask, contains pixels that can be copied from reconstruction to target view image. It is crafted by finding the intersection of the area to be diminished and the region of successfully rendered background pixels. The second mask, called invalid pixel mask, contains pixels that are missing from reconstruction, but fall into the area to be diminished. This can happen for multiple reasons. Most likely, the RGB-D cameras lack full coverage of the scene background. It is also possible that depth sensor noise has caused reconstruction to fail.

Armed with two input images and a pixel mask, it is possible to carry out the actual compositing. Fig. 6.2 shows steps of the compositing stages. Naive compositing by just copying the reconstruction image over target view using valid pixel mask will not work well and produce visually pleasing results. First issue is that the colors of reconstruction image and target view are quite different. This is due to differences in both sensors and lighting conditions. It would be possible to calibrate the colors of different cameras, but trying to estimate scene lighting can be very complex problem. Therefore a different approach using only existing source images is preferred. Pérez et al. (2003) introduce the concept of Poisson Image Editing that allows seamless compositing of two images. In this process, the features of images are preserved, but colors are modified so that there are no noticeable discontinuities at transitions from one image to another. After taking care of colors, only the final problem of filling image areas that have missing pixels in reconstruction is left. Typically the missing areas are small. This makes it possible to employ image inpainting techniques to guess the image content. Even though the guesses are not fully accurate in terms of what the real background is they still improve the visual appeal of the results. A method by Telea (2004) was chosen as it has good performance filling smaller areas.

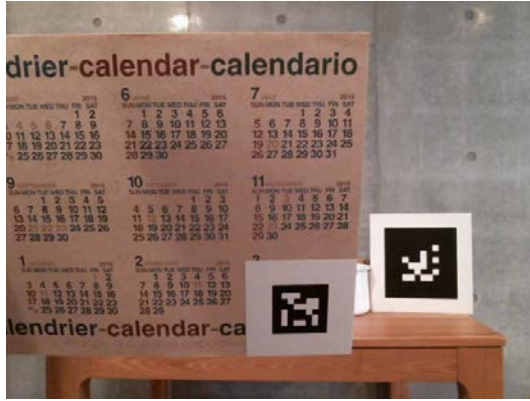
6.2.4 Results

Fig. 6.3 and Fig. 6.4 show scenes where objects are diminished from user view. In both scenes four cameras were used. One camera acts as a target view where some objects are diminished. Remaining three RGB-D cameras captured scene geometry. Following paragraphs discuss various aspects of the results.

In Fig. 6.3 a human is holding an AR marker object. This marker defines a pose of a bounding box with dimensions $0.8 \times 0.8 \times 1.9\text{m}$. Any object inside this bounding box area is diminished. This setup serves to demonstrate that diminishable objects can move freely in the scene as long as the marker is seen from cameras. A ball was thrown behind the human actor to show that even fast moving objects can be correctly reconstructed.

In Fig. 6.4 a ball object was diminished to show both scene background and human actor behind it. The actor is interacting with the ball and bouncing it off the ground. Even though the object is moving quickly and is quite close to the actor, background is correctly reconstructed.

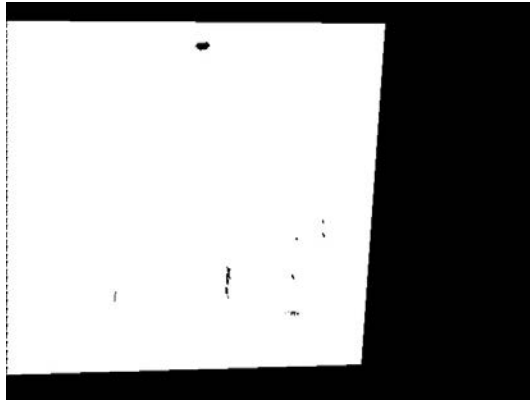
The geometry of both scenes is fairly complex and contains rich textures. The quality of the results is highly dependent of the scene background reconstruction results. In turn the reconstruction quality is mostly dependent on how much scene coverage RGB-D cameras have. When the object to be diminished gets too close to background, it might become impossible to reconstruct backgrounds due to lack of visibility.



(a) Target view



(b) Simple compositing



(c) Valid pixels mask



(d) Color correction



(e) Invalid pixels mask



(f) Inpainted result

Figure 6.2: Visualization of composition post-processing steps: (a) shows camera view where a calendar object should be diminished, (b) shows compositing without any post-processing, (c) shows reconstructed pixels that need color correction, (d) shows effect of color-correction filter, (e) shows pixels that are missing from reconstruction and need inpainting and (f) shows result after inpainting missing areas of the scene.

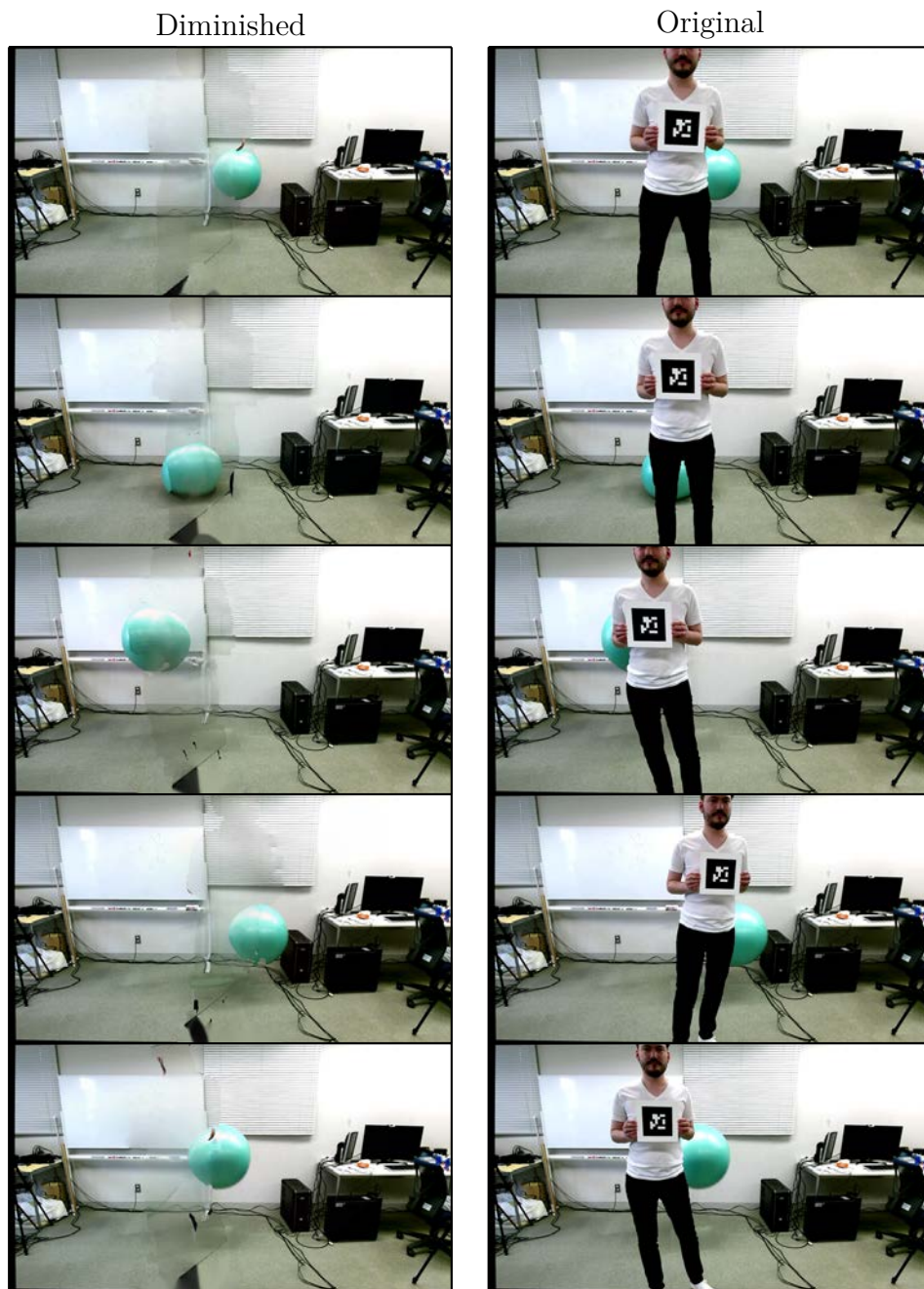


Figure 6.3: Results of diminishing human actor carrying an AR marker from a scene.

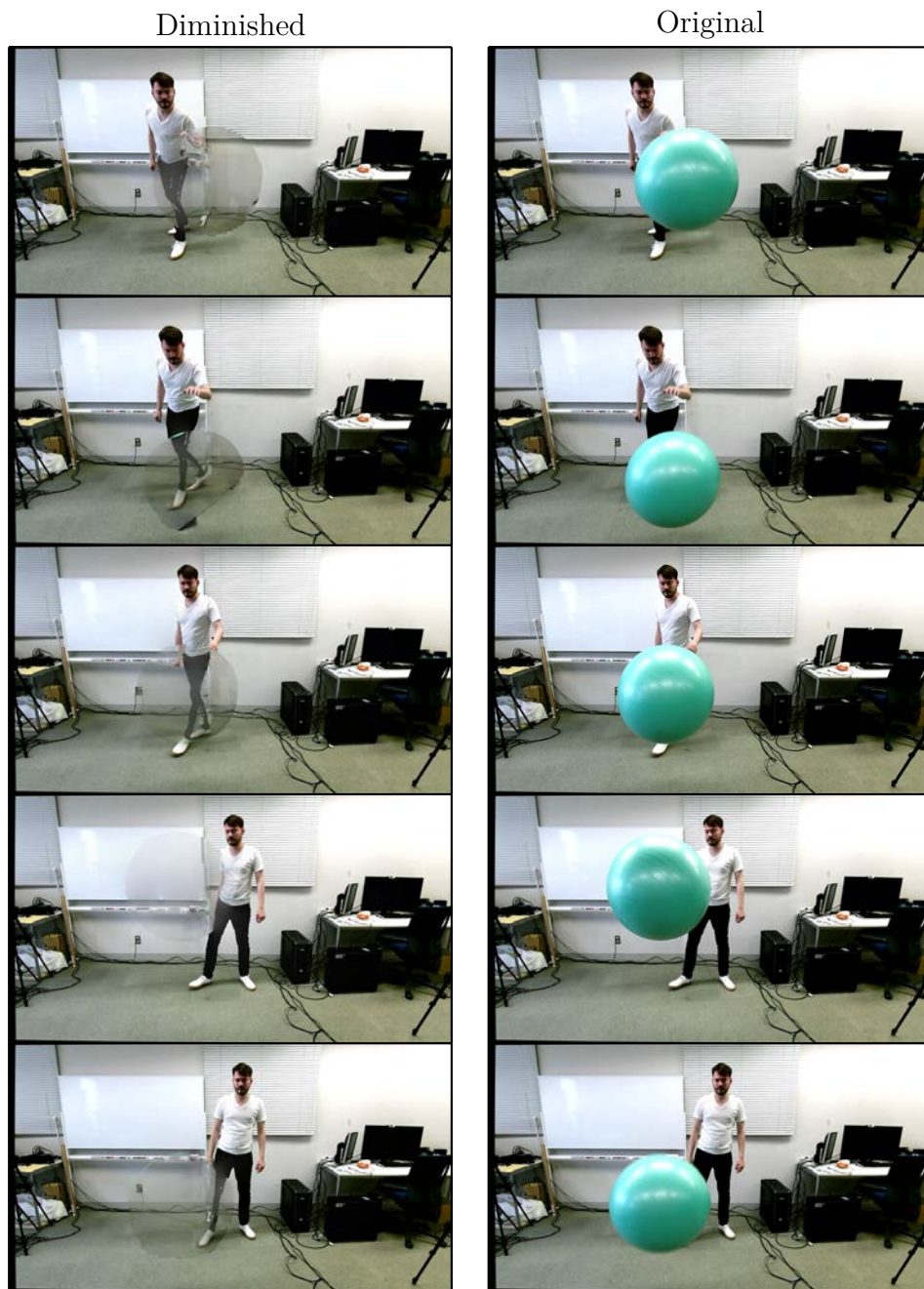


Figure 6.4: Results of diminishing colored spherical object from a scene.

Chapter 7

Conclusions

This thesis has presented two novel 3D reconstruction methods, ZipperMLS and FusionMLS, that utilize multiple RGB-D cameras. In addition, a diminished reality application utilizing the developed reconstruction system was presented. In this chapter a summary of the dissertation contributions is provided. Some insights about 3D reconstruction are also discussed.

Contributions. The key to using multiple consumer-grade RGB-D cameras for reconstruction was a combination of good temporal calibration of devices and a novel depth interpolation method. This approach considerably lowers the hardware requirements for simultaneous multi-camera capturing. Essentially, the method can be implemented in the application layer and does not require fine-grained hardware control over shutters. Unless widespread hardware support materializes soon, the methods proposed in this thesis might find use in other future 3D reconstruction systems.

Both of proposed reconstruction methods contain multiple smaller contributions. However, those smaller contributions make most sense when considered together as a single method. From the start of the design the methods were required to be able to handle dynamic content, large topology changes and not rely on any a priori known models. The primary goals of the methods were to achieve fast execution and low memory footprint.

ZipperMLS reconstruction can be seen as a modernized or re-imagined version of the classic mesh zippering (Turk and Levoy, 1994) method. Smoothing of surfaces was made

possible by using moving least squares resampling of point clouds. A new projection operator was suggested for MLS to obtain a regular point cloud structure suitable for meshing. The original mesh zippering method was not suited for execution on GPUs and while the proposed method in this work achieves same goal its fully-parallelized approach is quite different.

FusionMLS is a volumetric reconstruction method that uses very similar surface smoothing mechanism to ZipperMLS, but has very different way of extracting geometry from point clouds. It belongs to volumetric reconstruction methods family which has many desirable aspects evident of the widespread use of volumetric reconstruction in state-of-the-art works. The main contribution here is a joint surface estimation and mesh generation method that allows storing reconstruction volumes in memory. The low memory footprint allows reconstructing large scenes. It should be noted that traditional volumetric reconstruction concept of accumulating point cloud data over many frames cannot be used as there is no stored volume. This is where MLS surface estimation comes in and allows the reconstruction to succeed.

The proposed reconstruction methods can be developed further in multiple directions. The strength and weakness of current methods is that they reconstruct full scene in a single frame. On one hand it allows them to correctly reconstruct complex object movements but on the other hand any object parts that are not visible from cameras cannot be reconstructed. The way forward would be to develop a tracking mechanism that allows retaining scene areas which might be temporarily occluded from camera view.

Presented reconstruction methods are flexible and suitable for a variety of applications. Recently, mixed reality systems including augmented and diminished reality, have started to use 3D reconstruction more. The benefits of that direction are shown using a diminished reality application. The diminished reality methods diminish, i.e. hide, user selected objects from view. As the 3D reconstruction methods can handle arbitrary scene geometry, the application works well in various situations.

Insights. Research is inherently an iterative process. Many ideas need to be tried in the exploration of the unknown to find success. Insight from previous work can considerably lessen the burden of exploring full solution space of research problems. In

the end, however, many ideas need to be tried through implementation. The turnaround time for implementing methods needs to be minimized to maximize research output. In terms of 3D reconstruction, much of the time is spent on developing tooling instead of building reconstruction algorithms. Number of frameworks have appeared for static scene reconstruction and visualization, but none are well suited for real-time dynamic scenes. While many computer vision libraries have added support for GPU accelerated methods, the data structures are not interchangeable between frameworks. This has led to considerable amount of reimplementing of methods to avoid prohibitively expensive copying and reordering of data. The takeaway here is that current 3D reconstruction and visualization toolkits could use modernization and GPU interface unification. Another point is that much of 3D reconstruction work is about understanding data and the effect of reconstruction methods on that data. For this reason tooling might be more important than the actual reconstruction algorithm development in terms of research process.

In recent years, the pace of 3D reconstruction research has accelerated. This is likely driven due to GPU devices becoming faster and RGB-D capture device improvements. Furthermore, the computer vision field, including 3D reconstruction, is being increasingly used in consumer and industrial applications. This has resulted in increase of funding to industry backed research laboratories. In terms of 3D reconstruction, there is a need to build capture systems that require multi-disciplinary knowledge base and considerable investment. Industry related laboratories have proven to be adept in building such systems. This has also resulted in very high level research publications. The takeaway here is that academic institutions with limited manpower and funding need to be fast and flexible to be able to compete in current research environment. Within the bounds of this dissertation it meant using only consumer-grade off-the-shelf hardware and compensating for hardware shortcomings algorithmically. Additionally, open source software should be extensively exploited with most work directed towards integrating existing solutions for anything out of the core competence of 3D reconstruction.

Future developments. Generally, 3D reconstruction methods are evolving to work in real time, be able to handle any dynamic content, and have the ability to reconstruct large scenes. Here some possible future developments are discussed in scene capture and

3D reconstruction fields.

The need for good capture devices is well understood in the industry. Both new time-of-flight and stereo-based methods are actively developed. Long-range sensing (over 10 meters) is being driven forward by self-driving car development and almost exclusively make use of time-of-flight sensing. At the same time short range sensing is finding more use in consumer settings, being integrated into mobile phones, laptops and gaming devices for augmented reality applications. These sensors are likely to become more dominated by stereo-based methods in combination with some structured light techniques that can improve accuracy. This is due to the increased effort to produce application specific integrated circuits (ASICs) that can do stereo matching. Stereo methods can work with traditional image sensors as opposed to hard-to-manufacture time-of-flight sensors that remain at VGA resolutions. Additionally, time-of-flight requires power hungry light sources that cannot be integrated into small devices and has serious unsolved interference issues when using multiple devices. In conclusion it is likely that best new RGB-D devices for indoors 3D reconstruction are mostly stereo based. The noise models of reconstruction methods should be changed appropriately. For example the edges of depth maps will become more unreliable while number of outlier points are likely to be reduced compared to current time-of-flight cameras.

Almost all recent state-of-the-art 3D reconstruction methods use TSDF volumes. The TSDF volume can be interpreted as a probability distribution of surface positions. By accumulating data over multiple frames, that probability distribution becomes more accurate, hence the increased quality of reconstruction. One of the main benefits of TSDF is that integrating new measurements is computationally very cheap. As a drawback using volumes comes with high memory costs in terms of both size and bandwidth. This suggests that the probabilistic concept of TSDF should be retained in future, but the volume representation is best discarded. Some sparse spatial TSDF structures have been proposed, but they still retain the basic volume and voxels concept. A more radical departure from volumes can be envisioned. For example, a reconstruction method creates rough mesh of a scene as a first step, perhaps using methods brought forward in this thesis. Then, a TSDF probability distribution is embedded to the mesh vertices or faces to describe geometry around surface. This would allow very compact memory usage while

retaining the power of TSDF methods to enhance reconstruction quality over multiple frames.

Point-based 3D reconstruction methods have shown some advantages over volumetric reconstruction. For example, outliers can easily be detected by categorizing points as unstable or stable, waiting for confirmation of their validity from follow-up frames. This concept cannot be easily used in TSDF volumes as info about individual points is lost after accumulating data to volumes. However, this idea could be used with triangle meshes. Initial rough meshes could be marked as unstable and then be gradually moved to stable category as their existence is confirmed from other frames. The implementation might be more complicated than for points, but well-known issues about point-based methods such as difficulties in rendering, would be immediately solved.

The methods proposed in this thesis show that reconstructing a scene from single RGB-D camera frame can have fairly good results. However, static reconstruction methods have previously shown that high-quality results can be obtained when accumulating data over multiple frames. Using the same concept in dynamic scenes requires additionally movement tracking mechanisms to be developed. In TSDF volume-based reconstruction methods voxels lack explicit connectivity information and hence it is difficult to assert which model parts should be moved together under deformations. Occasionally, a graph or mesh is built to represent that connectivity. At this point, again, it might be reasonable to work purely with triangle meshes. Meshes are sparse and save memory, connectivity information can be made available for optimizing deformations. This would likely also solve an issue known as voxel-collision where separate surfaces are welded together when they touch. Meshes, however, can be kept separated even when surfaces touch.

The central idea of the last paragraphs is that triangle meshes might take a central role in 3D reconstruction instead of volumes or point clouds. The main issue with this approach is that meshes need to be first generated by some method. The most viable method so far has been to use TSDF volumes. In that scenario, it makes sense to make volumes central to reconstruction as there is no way around them. The methods developed in this thesis might present an alternative way of generating triangle meshes with low memory cost and fast execution. Essentially, ZipperMLS or FusionMLS could act as a first stage of a larger tracking-based 3D reconstruction system.

Bibliography

- Alexa, M. and Adamson, A. (2004). On normals and projection operators for surfaces defined by point sets. In *Proceedings of the First Eurographics Conference on Point-Based Graphics*, SPBG'04, pages 149–155, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.
- Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. (2001). Point set surfaces. In *Proceedings of the Conference on Visualization '01*, VIS '01, pages 21–28, Washington, DC, USA. IEEE Computer Society.
- Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. (2003). Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15.
- Alexiadis, D. S., Chatzitofis, A., Zioulis, N., Zoidi, O., Louizis, G., Zarpalas, D., and Daras, P. (2017). An integrated platform for live 3D human reconstruction and motion capturing. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):798–813.
- Alexiadis, D. S., Zarpalas, D., and Daras, P. (2013). Real-time, full 3-D reconstruction of moving foreground objects from multiple consumer depth cameras. *IEEE Transactions on Multimedia*, 15(2):339–358.
- Alexiadis, D. S., Zioulis, N., Zarpalas, D., and Daras, P. (2018). Fast deformable model-based human performance capture and FVV using consumer-grade RGB-D sensors. *Pattern Recognition*. In Press.

- Amenta, N. and Bern, M. (1999). Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504.
- Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266. ACM.
- Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A., and Silva, C. T. (2017). A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 36(1):301–329.
- Bradski, G. (2000). The OpenCV library. *Dr. Dobb’s Journal of Software Tools*.
- Cazals, F. and Giesen, J. (2006). Delaunay triangulation based surface reconstruction: Ideas and algorithms. In *EFFECTIVE COMPUTATIONAL GEOMETRY FOR CURVES AND SURFACES*, pages 231–273. Springer.
- Chen, J., Bautembach, D., and Izadi, S. (2013). Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4):113:1–113:16.
- Cheng, Z.-Q., Wang, Y.-Z., Li, B., Xu, K., Dang, G., and Jin, S.-Y. (2008). A survey of methods for moving least squares surfaces. In *Volume graphics*, pages 9–23.
- Chien, C., Sim, Y., and Aggarwal, J. (1988). Generation of volume/surface octree from range data. In *Computer Vision and Pattern Recognition, 1988. Proceedings CVPR’88., Computer Society Conference on*, pages 254–260. IEEE.
- Collet, A., Chuang, M., Sweeney, P., Gillett, D., Evseev, D., Calabrese, D., Hoppe, H., Kirk, A., and Sullivan, S. (2015). High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34(4):69:1–69:13.
- Connolly, C. (1984). Cumulative generation of octree models from range data. In *Robotics and Automation. Proceedings. 1984 IEEE International Conference on*, volume 1, pages 25–32. IEEE.
- Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH ’96*, pages 303–312, New York, NY, USA. ACM.

- Digne, J., Cohen-Steiner, D., Alliez, P., de Goes, F., and Desbrun, M. (2014). Feature-preserving surface reconstruction and simplification from defect-laden point sets. *Journal of Mathematical Imaging and Vision*, 48(2):369–382.
- Doi, A. and Koide, A. (1991). An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems*, 74(1):214–224.
- Dou, M., Davidson, P., Fanello, S. R., Khamis, S., Kowdle, A., Rhemann, C., Tankovich, V., and Izadi, S. (2017). Motion2Fusion: Real-time volumetric performance capture. *ACM Trans. Graph.*, 36(6):246:1–246:16.
- Dou, M. and Fuchs, H. (2014). Temporally enhanced 3D capture of room-sized dynamic scenes with commodity depth cameras. In *2014 IEEE Virtual Reality (VR)*, pages 39–44.
- Dou, M., Khamis, S., Degtyarev, Y., Davidson, P., Fanello, S. R., Kowdle, A., Escolano, S. O., Rhemann, C., Kim, D., Taylor, J., Kohli, P., Tankovich, V., and Izadi, S. (2016). Fusion4D: Real-time performance capture of challenging scenes. *ACM Trans. Graph.*, 35(4):114:1–114:13.
- Duckworth, T. and Roberts, D. J. (2014). Parallel processing for real-time 3D reconstruction from video streams. *Journal of Real-Time Image Processing*, 9(3):427–445.
- Everitt, C. (2001). Interactive order-independent transparency. *White paper, nVIDIA*, 2(6):7.
- Firman, M. (2016). RGBD datasets: past, present and future. In *CVPR Workshop on Large Scale 3D Data: Acquisition, Modelling and Analysis*.
- Fleishman, S., Cohen-Or, D., and Silva, C. T. (2005). Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552.
- Franco, J.-S. and Boyer, E. (2003). Exact polyhedral visual hulls. In *British Machine Vision Conference (BMVC’03)*, volume 1, pages 329–338, Norwich, United Kingdom.

- Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292.
- Guennebaud, G., Germann, M., and Gross, M. (2008). Dynamic sampling and rendering of algebraic point set surfaces. *Computer Graphics Forum*, 27(2):653–662.
- Guennebaud, G. and Gross, M. (2007). Algebraic point set surfaces. *ACM Trans. Graph.*, 26(3).
- Guo, K., Xu, F., Yu, T., Liu, X., Dai, Q., and Liu, Y. (2017). Real-time geometry, albedo, and motion reconstruction using a single RGB-D camera. *ACM Trans. Graph.*, 36(3):32:1–32:13.
- Hashimoto, T., Uematsu, Y., and Saito, H. (2010). Generation of see-through baseball movie from multi-camera views. In *2010 IEEE International Workshop on Multimedia Signal Processing*, pages 432–437.
- Hilton, A., Stoddart, A. J., Illingworth, J., and Windeatt, T. (1996). *Reliable surface reconstruction from multiple range images*, pages 117–126. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Holz, D. and Behnke, S. (2013). Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. In *Intelligent autonomous systems 12*, pages 61–73. Springer.
- Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78.
- Ienaga, N., Bork, F., Meerits, S., Mori, S., Fallavollita, P., Navab, N., and Saito, H. (2016). First deployment of diminished reality for anatomy education. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*, pages 294–296.
- Innmann, M., Zollhöfer, M., Nießner, M., Theobalt, C., and Stamminger, M. (2016). VolumeDeform: Real-time volumetric non-rigid reconstruction. In Leibe, B., Matas,

- J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, pages 362–379, Cham. Springer International Publishing.
- Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., et al. (2011). KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM.
- Kazhdan, M. (2005). Reconstruction of solid models from oriented point sets. In *Proceedings of the Third Eurographics Symposium on Geometry Processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.
- Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In Sheffer, A. and Polthier, K., editors, *Symposium on Geometry Processing*. The Eurographics Association.
- Kazhdan, M. and Hoppe, H. (2013). Screened Poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13.
- Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., and Kolb, A. (2013). Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *2013 International Conference on 3D Vision - 3DV 2013*, pages 1–8.
- Kostavelis, I. and Gasteratos, A. (2015). Semantic mapping for mobile robotics tasks. *Robot. Auton. Syst.*, 66(C):86–103.
- Kriegel, S., Rink, C., Bodenmüller, T., and Suppa, M. (2015). Efficient next-best-scan planning for autonomous 3D surface reconstruction of unknown objects. *Journal of Real-Time Image Processing*, 10(4):611–631.
- Kuster, C., Bazin, J.-C., Öztireli, C., Deng, T., Martin, T., Popa, T., and Gross, M. (2014). Spatio-temporal geometry fusion for multiple hybrid cameras using moving least squares surfaces. *Computer Graphics Forum*, 33(2):1–10.

- Laurentini, A. (1994). The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(2):150–162.
- Levin, D. (2004). Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization*, pages 37–49. Springer.
- Li, M., Schirmacher, H., Magnor, M., and Siedel, H. P. (2002). Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *2002 IEEE Workshop on Multimedia Signal Processing.*, pages 9–12.
- Li, Z., Ji, Y., Yang, W., Ye, J., and Yu, J. (2018). Robust 3D human motion reconstruction via dynamic template construction. *ArXiv e-prints*.
- Lindlbauer, D. and Wilson, A. D. (2018). Remixed reality: Manipulating space and time in augmented reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI ’18, pages 129:1–129:13, New York, NY, USA. ACM.
- Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169.
- Maimone, A. and Fuchs, H. (2011). Encumbrance-free telepresence system with real-time 3D capture and display using commodity depth cameras. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 137–146. IEEE.
- Marras, S., Ganovelli, F., Cignoni, P., Scateni, R., and Scopigno, R. (2010). Controlled and adaptive mesh zippering. In *GRAPP*, pages 104–109.
- Marton, Z. C., Rusu, R. B., and Beetz, M. (2009). On fast surface reconstruction methods for large and noisy datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3218–3223, Kobe, Japan.
- Matusik, W., Buehler, C., and McMillan, L. (2001). *Polyhedral Visual Hulls for Real-Time Rendering*, pages 115–125. Springer Vienna, Vienna.
- Mori, S., Ikeda, S., and Saito, H. (2017). A survey of diminished reality: Techniques for visually concealing, eliminating, and seeing through real objects. *IPSJ Transactions on Computer Vision and Applications*, 9(1):17.

- Narayanan, P. J., Rander, P. W., and Kanade, T. (1998). Constructing virtual worlds using dense stereo. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 3–10.
- Newcombe, R. A., Fox, D., and Seitz, S. M. (2015). DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352.
- Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6):169:1–169:11.
- Oliveira, A., Oliveira, J. F., Pereira, J. M., de Araújo, B. R., and Boavida, J. (2014). 3D modelling of laser scanned and photogrammetric data for digital documentation: the Mosteiro da Batalha case study. *Journal of Real-Time Image Processing*, 9(4):673–688.
- Orts-Escalano, S., Morell, V., Garcia-Rodriguez, J., Cazorla, M., and Fisher, R. B. (2015). Real-time 3D semi-local surface patch extraction using GPGPU. *Journal of Real-Time Image Processing*, 10(4):647–666.
- Orts-Escalano, S., Rhemann, C., Fanello, S., Chang, W., Kowdle, A., Degtyarev, Y., Kim, D., Davidson, P. L., Khamis, S., Dou, M., Tankovich, V., Loop, C., Cai, Q., Chou, P. A., Mennicken, S., Valentin, J., Pradeep, V., Wang, S., Kang, S. B., Kohli, P., Lutchyn, Y., Keskin, C., and Izadi, S. (2016). Holoportation: Virtual 3D teleportation in real-time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST ’16, pages 741–754, New York, NY, USA. ACM.
- Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318.
- Pfister, H., Zwicker, M., van Baar, J., and Gross, M. (2000). Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’00, pages 335–342, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.
- Plüss, (Kuster), C., Ranieri, N., Bazin, J.-C., Martin, T., Laffont, P.-Y., Popa, T., and Gross, M. (2016). An immersive bidirectional system for life-size 3D communication. In

- Proceedings of the 29th International Conference on Computer Animation and Social Agents, CASA '16*, pages 89–96, New York, NY, USA. ACM.
- Rameau, F., Ha, H., Joo, K., Choi, J., Park, K., and Kweon, I. S. (2016). A real-time augmented reality system to see-through cars. *IEEE Transactions on Visualization and Computer Graphics*, 22(11):2395–2404.
- Ronfard, R. and Taubin, G. (2010). *Image and Geometry Processing for 3-D Cinematography*, volume 5 of *Geometry and Computing*. Springer.
- Rünz, M. and Agapito, L. (2017). Co-Fusion: Real-time segmentation, tracking and fusion of multiple objects. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4471–4478.
- Saito, H., Baba, S., and Kanade, T. (2003). Appearance-based virtual view generation from multicamera videos captured in the 3-D room. *IEEE Transactions on Multimedia*, 5(3):303–316.
- Sakai, S., Yanase, Y., Matayoshi, Y., and Inami, M. (2018). D-ball: virtualized sports in diminished reality. In *Proceedings of the First Superhuman Sports Design Challenge: First International Symposium on Amplifying Capabilities and Competing in Mixed Realities, SHS '18*, pages 6:1–6:6, New York, NY, USA. ACM.
- Scheidegger, C. E., Fleishman, S., and Silva, C. T. (2005). Triangulating point set surfaces with bounded error. In *Proceedings of the Third Eurographics Symposium on Geometry Processing, SGP '05*, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.
- Schreiner, J., Scheidegger, C. E., Fleishman, S., and Silva, C. T. (2006). Direct (re)meshing for efficient surface processing. *Computer Graphics Forum*.
- Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1, CVPR '06*, pages 519–528, Washington, DC, USA. IEEE Computer Society.

- Siltanen, S. (2017). Diminished reality for augmented reality interior design. *The Visual Computer*, 33(2):193–208.
- Steinbrücker, F., Sturm, J., and Cremers, D. (2014). Volumetric 3D mapping in real-time on a CPU. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2021–2028.
- Sugimoto, K., Fujii, H., Yamashita, A., and Asama, H. (2014). Half-diminished reality image using three RGB-D sensors for remote control robots. In *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, pages 1–6.
- Telea, A. (2004). An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34.
- Turk, G. and Levoy, M. (1994). Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318. ACM.
- Wang, J., Yu, Z., Zhu, W., and Cao, J. (2013). Feature-preserving surface reconstruction from unoriented, noisy point data. *Computer Graphics Forum*, 32(1):164–176.
- Wang, K., Zhang, G., and Xia, S. (2017). Templateless non-rigid reconstruction and motion tracking with a single RGB-D camera. *IEEE Transactions on Image Processing*, 26(12):5966–5979.
- Wang, R., Wei, L., Vouga, E., Huang, Q., Ceylan, D., Medioni, G., and Li, H. (2016). Capturing dynamic textured surfaces of moving targets. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 271–288, Cham. Springer International Publishing.
- Whelan, T., F Salas-Moreno, R., Glocker, B., J Davison, A., and Leutenegger, S. (2016). ElasticFusion: Real-time dense SLAM and light source estimation. *The International Journal of Robotics Research*, 35:1697–1716.
- Whelan, T., McDonald, J. B., Kaess, M., Fallon, M. F., Johannsson, H., and Leonard, J. J.

- (2012). Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia.
- Yaguchi, S. and Saito, H. (2004). Arbitrary viewpoint video synthesis from multiple uncalibrated cameras. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):430–439.
- Yan, Z. and Xiang, X. (2016). Scene flow estimation: A survey. *CoRR*, abs/1612.02590.
- Yokoi, T. and Fujiyoshi, H. (2006). Generating a time shrunk lecture video by event detection. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 641–644. IEEE.
- Yoshida, T., Jo, K., Minamizawa, K., Nii, H., Kawakami, N., and Tachi, S. (2008). Transparent cockpit: Visual assistance system for vehicle using retro-reflective projection technology. In *2008 IEEE Virtual Reality Conference*, pages 185–188.
- Yu, T., Guo, K., Xu, F., Dong, Y., Su, Z., Zhao, J., Li, J., Dai, Q., and Liu, Y. (2017). BodyFusion: Real-time capture of human motion and surface geometry using a single depth camera. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 910–919.
- Zhang, H. and Xu, F. (2018). MixedFusion: Real-time reconstruction of an indoor scene with dynamic objects. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1.
- Zollhöfer, M., Nießner, M., Izadi, S., Rehmann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A., Loop, C., Theobalt, C., and Stamminger, M. (2014). Real-time non-rigid reconstruction using an RGB-D camera. *ACM Trans. Graph.*, 33(4):156:1–156:12.
- Zollhöfer, M., Stotko, P., Görlitz, A., Theobalt, C., Nießner, M., Klein, R., and Kolb, A. (2018). State of the art on 3D reconstruction with RGB-D cameras. *Computer Graphics Forum (Eurographics State of the Art Reports 2018)*, 37(2).
- Zwicker, M., Pfister, H., van Baar, J., and Gross, M. (2001). Surface splatting. In

Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, pages 371–378, New York, NY, USA. ACM.