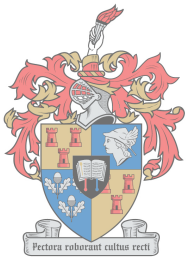
Creating 3D Models using Reconstruction Techniques

by

Javonne Jason Martin



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Computer Science) in the Faculty of Science at Stellenbosch University

Supervisors: Dr R. S. Kroon
Dr H. A. C. de Villiers

December 2018

The financial assistance of the National Research Foundation (NRF) and the MIH Media Lab towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF or MIH Media Lab.

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ..... December 2018 .....

Abstract

Creating 3D Models using Reconstruction Techniques

J J Martin

Thesis: M.Sc (Computer Science)

December 2018

Virtual reality models of real world environments have a number of compelling applications, such as preserving the architecture and designs of older buildings. This process can be achieved by using 3D artists to reconstruct the environment, however this is a long and expensive process. Thus, this thesis investigates various techniques and approaches used in 3D reconstruction of environments using a single RGB-D camera and aims to reconstruct the 3D environment to generate a 3D model. This would allow non-technical users to reconstruct environments and use these models in business and simulations, such as selling real-estate, modifying pre-existing structures for renovation and planning. With the recent improvements in virtual reality technology such as the Oculus Rift and HTC Vive, a user can be immersed into virtual reality environments created from real world structures. A system based on Kinect Fusion is implemented to reconstruct an environment and track the motion of the camera within the environment. The system is designed as a series of self-contained subsystems that allows for each of the subsystems to be modified, expanded upon or easily replaced by alternative methods. The system is made available as an open source C++ project using Nvidia's CUDA framework to aid reproducibility and provides a platform for future research. The system makes use of the Kinect sensor to capture information about the environment. A coarse-to-fine least squares approach is used to estimate the motion of the camera. In addition, the system employs a frame-to-model approach that uses a view of the estimated reconstruction of the model as the reference frame and the incoming scene data as the target. This minimises the drift with respect to the true trajectory of the camera. The model is built using a volumetric approach, with volumetric information implicitly stored as a truncated signed distance function. The system filters out noise in the raw sensor data by using a bilateral filter. A point cloud is extracted from the volume using an

orthogonal ray caster which enables an improved hole-filling approach. This allows the system to extract both the explicit and implicit structure from the volume. The 3D reconstruction is followed by mesh generation based on the point cloud. This is achieved by using an approach related to Delaunay triangulation, the ball-pivot algorithm. The resulting system processes frames at 30Hz, enabling real-time point cloud generation, while the mesh generation occurs offline. This system is initially tested using Blender to generate synthetic data, followed by a series of real world tests. The synthetic data is used to test the presented system's motion tracking against the ground truth. While the presented system suffers from the effects of drift over long frame sequences, it is shown to be capable of tracking the motion of the camera. This thesis finds that the ball pivot algorithm can generate the edges and faces for synthetic point clouds, however it performs poorly when using the noisy synthetic and real world data sets. Based on the results obtained it is recommended that the obtained point cloud be preprocessed to remove noise before it is provided to the mesh generation algorithm and an alternative mesh generation technique should be employed that is more robust to noise.

Uittreksel

3D-modelle met behulp van Rekonstruksie Tegnieke

J J Martin

Tesis: M.Sc (Rekenaar Wetenskap)

Desember 2018

Modelle in virtuele realiteit van werklike omgewings het 'n aantal belangrike toepassings, soos byvoorbeeld die behoud van die argitektuur en ontwerpe van geskiedkundig belangrike geboue. 3D kunstenaars kan ingespan word om omgewings te modelleer, maar dit is 'n lang en duur proses. Hierdie proefskrif ondersoek verskillende tegnieke en benaderings wat gebruik word in die 3D rekonstruksie van omgewings deur gebruik van 'n enkele RGB-D kamera en beoog om die 3D rekonstruksie van die omgewing te omskep in 'n 3D-model. Hierdie sal nie-tegniese gebruikers toelaat om self modelle te skep van omgewings, en om hierdie modelle te gebruik in besigheid toepassings en simulasies, soos byvoorbeeld die verkoop van vaste eiendom, die wysiging van bestaande strukture vir beplanning en opknapping. Met die onlangse tegnologiese verbeteringe in die veld van virtuele realiteit soos, byvoorbeeld, die Oculus Rift en HTC Vive, kan 'n gebruiker geplaas word in 'n virtuele omgewing wat geskep was vanaf strukture in die werklike wêreld. 'n Stelsel gebaseer op Kinect Fusion word geïmplementeer om 'n omgewing te rekonstrueer en die beweging van die kamera binne die omgewing te volg. Die stelsel is ontwerp as 'n reeks selfstandige modules wat die afsonderlike aanpassing, uitbreiding of vervanging van die modules vergemaklik. Die stelsel word beskikbaar gestel as 'n open source C++ projek met behulp van Nvidia se CUDA raamwerk om reproduseerbaarheid te bevorder en bied ook 'n platform vir toekomstige navorsing. Die stelsel maak gebruik van die Kinect-sensor om inligting oor die omgewing vas te vang. 'n Grof-tot-fyn kleinste kwadraat benadering word gebruik om die beweging van die kamera te skat. Daarbenewens gebruik die stelsel 'n beeld-tot-model benadering wat gebruik maak van die beraamde rekonstruksie van die model as die verwysingsraamwerk en die inkomende toneel data as die teiken. Dit verminder die drywing ten opsigte van die ware trajek van die kamera. Die model word gebou met behulp van 'n volumetriese benadering,

met volumetriese inligting wat implisiet gestoor word as 'n verkorte getekende afstandfunksie. Die stelsel filter ruis in die ruwe sensor data uit deur om 'n bilaterale filter te gebruik. 'n Puntwolk word uit die volume onttrek deur 'n ortogonale straalvolger te gebruik wat die vul van gate in die model verbeter. Dit laat die stelsel toe om die eksplisiete en implisiete struktuur van die volume te onttrek. Die 3D-rekonstruksie word gevolg deur maasgenerasie gebaseer op die puntwolk. Dit word behaal deur 'n benadering wat verband hou met Delaunay triangulasie, die bal wentelings algoritme, te gebruik. Die resulterende stelsel verwerk beelde teen 30Hz, wat intydse-puntwolkgenerasie moontlik maak, terwyl die maasgenerering aflyn plassvind. Hierdie stelsel word aanvanklik getoets deur om met Blender sintetiese data te genereer, gevolg deur 'n reeks werklike wêreldtoetse. Die sintetiese data word gebruik om die stelsel se afgeskatte trajek teenoor die korrekte trajek te vergelyk. Terwyl die stelsel ly aan die effekte van wegdrywing oor langdurige intreevideos, word dit getoon dat die stelsel wel die lokale beweging van die kamera kan volg. Hierdie proefskrif bevind dat die bal wentelingsalgoritme die oppervlaktes en bygaande rande vir sintetiese puntwolke kan genereer, maar dit is sterk gevoelig vir ruis in sintetiese en werklike datastelle. Op grond van die resultate wat verkry word, word aanbeveel dat die verkrygde puntwolk vooraf verwerk word om ruis te verwyder voordat dit aan die maasgenereringsalgoritme verskaf word, en 'n alternatiewe maasgenereringstegniek moet gebruik word wat meer robuust is ten opsigte van ruis.

Contents

List of Figures

List of Tables

List of Algorithms

List of Common Symbols

d The distance represented by the TSDF volume.

d_j A destination point in the local coordinate system used in the ICP algorithm.

d_j^g A destination point in the global coordinate system used in the ICP algorithm.

f The focal length for the X and Y axis, the distance between the camera centre and the image plane.

f_x The focal length for the X axis.

f_y The focal length for the Y axis.

g_{max} The maximum value that the distance function can be.

g_{min} The minimum value that the distance function can be.

gs A vector that contains the dimensions of the volume containing the voxels.

$G(\mathbf{x})$ The combination of a series of signed distance functions with $\mathbf{x}$ representing a point in 3D space.

$g(\mathbf{x})$ The signed distance function with $\mathbf{x}$ representing a point in 3D space.

I The image plane (as illustrated in Figures 2.2, 2.3 and 2.4).

I_i An RGB-D image captured at time i.

P_i Point cloud generated from the RGB-D image at time i.

K The camera intrinsic matrix. Converts between 3D world coordinates to 2D camera coordinates.

$N_l^{d,g}$ The destination normal map constructed from the pyramid at level l in the global coordinate system.

n_j^d The normal vector at the destination point in the local coordinate system.

$n_j^{d,g}$ The normal vector at the destination point in the global coordinate system.

n_j^s The normal vector at the source point in the local coordinate system.

$n_j^{s,g}$ The normal vector at the source point in the global coordinate system.

N_l^s The source normal map constructed from the pyramid at level l.

$N_l^{s,g}$ The source normal map constructed from the pyramid at level l in the global coordinate system.

o The centre of the image plane.

o_x The displacement of the optical axis along the X axis.

o_y The displacement of the optical axis along the Y axis.

p_{camera} The position of the camera in 3D space.

$\mathbf{p}^H$ A homogeneous coordinate in 2D space.

p_{object} The position of a object in 3D space.

R The raw depth image obtained from the Kinect sensor.

`RayDir` The unit vector of `RayNext`. The direction of the ray being cast from the camera's position to the image plane.

`RayNext` This is a vector generated from a ray being cast from the camera's position to a pixel location on the image plane.

s_j A source point in the local coordinate system used in the ICP algorithm.

s_j^g A source point in the global coordinate system used in the ICP algorithm.

`StepSize` The size that the ray is incrementally increased in length by.

t_{i+1} The translation vector obtained from the transformation matrix T_{i+1}.

u A pixel location on the image plane.

$\hat{\mathbf{u}}$ A back projected pixel location.

u_x The X pixel coordinate of an image.

u_y The Y pixel coordinate of an image.

vc The location of the centre of the 3D volume that contains the voxels.

vcs A vector that contains the size in meters that each voxel represents for each axis.

$V_l^{d,g}$ The vertex map of the destination points created from a pyramid level l in the global coordinate system.

V_l^s The vertex map of the source points created from a pyramid level l.

$V_l^{s,g}$ The vertex map of the source points created from a pyramid level l in the global coordinate system.

$W(\mathbf{x})$ The combined weight function with $\mathbf{x}$ representing a point in 3D space.

Chapter 1

Introduction

Low cost 3D-sensing hardware such as the Microsoft Kinect have facilitated broad-based development of faster and higher quality 3D reconstruction. As a result, many advances have been made in the field. Research on structure from motion (SFM), multi-view stereo (MVS) and improving hardware have led to more efficient systems. SFM is a ranging technique for estimating 3D structures from 2D images. MVS is a group of techniques that uses stereo correspondence to reconstruct objects in 3D. With the emergence of low cost virtual reality equipment, such as the Oculus Rift and the HTC Vive, 3D reconstructions could be used in game engines to display environments in virtual reality. Currently there are many proposed solutions to the problems of 3D reconstruction and mesh reconstruction. This thesis aims to investigate the problem of 3D reconstruction and mesh reconstruction for the creation of virtual reality environments.

Most 3D reconstruction techniques are either used in research for self driving cars or commercial robotics systems that mainly focus on trajectory reconstruction. It is difficult for a non-technical user to make use of these 3D modelling techniques. The proposed system opens its usages to many applications such as creating assets from real world objects in gaming, viewing 3D models of properties online and simulations for virtual reality exposure therapy.

Virtual reality exposure therapy is a treatment method where patients are slowly exposed to traumatic stimuli with the assistance of virtual reality equipment. This allows patients to interact with their phobia virtually, without access to the real phobia itself. The use of this system could allow technicians to assist therapists in simulating real-world environments for patients.

Such a system could also be applied in the property industry. Instead of viewing buildings by visiting them, the model could be placed on a website to display the 3D mesh of the building for potential clients to view. This would allow measurements of the rooms and dimensions to be easily accessible for the potential clients.

1.1 Problem Statement

Virtual environments are used to create visualizations for simulations, virtual reality and games. These environments are time-consuming to create, requiring artists to model and texture the environment. Creating a digital version of a room automatically from photos could accelerate the process and reduce the cost of modelling.

This could be performed by 'scanning' the environment through taking photos and collecting the depth information. This would allow automatic reconstruction of the environment from this data. This approach would reduce the time taken to model the environment and would not require the skill of a CGI artist. The process of 'scanning' requires the use of a depth sensor such as the Cyberware 3030 (Cyberware Incorporated, 2018) or Microsoft Kinect (Microsoft, 2018). The ability to reconstruct a room gives a person the power to create a digital version of an environment by pre-arranging a room according to their desire.

One key problem is the scarcity of open-source integration of existing techniques.

The 3D reconstruction problem has several challenges that must be overcome. Two things are required to reconstruct the environment: the position of the camera and the view from that position. The main challenge in 3D reconstruction is computing the position of the view.

Mesh reconstruction has the problem of ambiguity when generating meshes from point clouds. Since information about the surface is not known, mesh reconstruction algorithms need to determine the surface given a set of arbitrary points.

1.1.1 Aims and Objectives

The aim of this work is to create a system that allows a non-technical user to build a 3D model relatively quickly and export it to a common format that can be easily incorporated into 3D modelling tools and virtual reality simulations. In order to do this, the following objectives were set out for the system proposed in this thesis.

- Collect depth and colour data from the environment.
- Estimate the motion of the depth sensor.
- Achieve online motion estimation speeds.
- Pre-process the sensor data to reduce noise.
- Fuse successive sensory data sets together.

- Enable generation of large scale environments.
- Extract the implicit and explicit structure of the environment as a point cloud.
- Generate a mesh from the final point cloud data.
- Integrate colour data into the mesh.
- Export the meshed point cloud with the texture map in a universal format.

1.2 Contributions

This thesis makes the following contributions:

- A system was developed that allows for 3D reconstruction of environments. A scan-matching algorithm was implemented to estimate the motion of the camera. This works in conjunction with a volumetric model to reduce drift using a frame-to-model scan-matching technique. The volumetric model was assisted by a ray caster that allows the system to extract the implicit structure of the environment.
- A plug-in was developed that allowed synthetic data sets to be created along with ground truth information. This was achieved using Blender's Python interface.[1]
- Provision was made for large scale environments by using a volumetric approach to 3D reconstruction in combination with a wrap-around indexing system that allows the system to continuously map new areas (Section 5.5.2).
- It is demonstrated in Section 7.4 that the ball-pivoting algorithm (Bernardini, Mittleman, Rushmeier, Silva and Taubin, 1999) is more suited for object reconstruction, as opposed to being used for environment reconstruction.
- To facilitate reproducibility and future work by other researchers, the entire system is provided as an open-source implementation under the MIT license (Martin, 2017).

[1]Scripts are provided in an open-source repository to create synthetic data sets and record the ground truth trajectory (`https://gitlab.com/pleased/3d-reconstruction`).

1.3 Outline

This thesis is divided into 8 chapters. Chapter 2 (Background) introduces the concepts and the mathematical methods that are required to understand the rest of the thesis. Chapter 3 (Related Work) reviews existing techniques that are used for 3D reconstruction and mesh generation. Chapter 4 (Methodology) specifies the techniques that will be used in the system to achieve the aims and objectives of this thesis. Chapter 5 (Kinect Fusion) details the implementation used to reconstruct the point cloud as mentioned in the methodology. Chapter 6 (Mesh Generation) discusses generating a mesh from the point cloud obtained in the Kinect Fusion chapter. Chapter 7 (Experiments) describes the experiments performed to test the system's performance and analyses the results. This is followed by a summary of the system's performance. Chapter 8 (Conclusion) summarizes the thesis results and considers the proposed system's advantages and disadvantages. Finally, the chapter discusses additional work that could be considered for improving on reported results.

Chapter 2

Background

This chapter provides essential background information about various techniques and mathematical methods used in this thesis. This chapter covers the pinhole camera model (Section 2.1), which facilitates conversions between camera coordinates and real-world coordinates. 3D transformation matrices (Section 2.2) and quaternions (Section 2.2.4) are also discussed, providing a means for representing the location and orientation of the camera. Next, linear and non-linear least squares optimisation is presented to provide a method to compute motion estimates for the 3D transformation (Section 2.3). This is followed by the foundations of 3D modelling, demonstrating various techniques used during the 3D modelling process (Section 2.4). Finally, some background information about the Kinect sensor is provided (Section 2.5).

2.1 Pinhole Camera Model

The pinhole camera model describes a method for projecting the three-dimensional world onto a two-dimensional image plane. Imagine a closed box with a very small aperture that allows light in. Rays of light from an object in front of the box pass through the aperture and fall on the interior surface at the rear of the box. These light rays that pass through the aperture create an inverted image on this surface. This is illustrated in Figure 2.1.

2.1.1 Converting to Camera Space

The pinhole camera projection process is described mathematically as follows: Given a 3D Cartesian coordinate system, a point p_{camera} (representing the pinhole) located at the origin, a 2D plane I representing the image plane (i.e. the back of the box) with I intersecting the Z axis at p_{camera}, the point $\mathbf{o}$ representing the centre of the image plane, a 3D point p_{object}, and f the focal length (the perpendicular distance between $\mathbf{o}$ and p_{camera}). In this situation

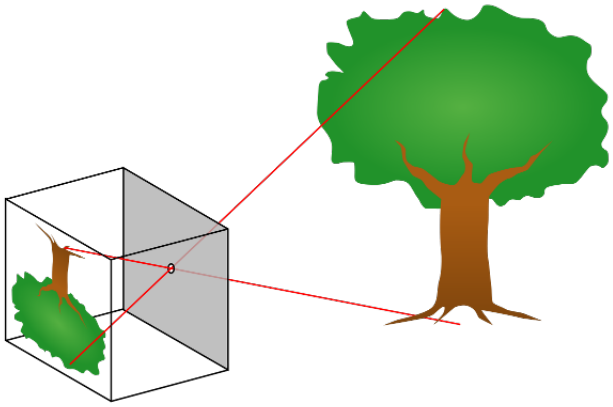


Figure 2.1: The basic pinhole camera model illustrates the reflected light from a tree entering the box through the aperture and falling on the interior surface at the rear of the box.

the Z axis is the optical axis, the direction the camera is facing. The goal is to project the 3D point p_{object} onto the 2D plane I. Projecting all the objects in the scene in this manner produces an image.

The projection is performed by drawing a line from p_{object} passing through p_{camera} to the plane I. This produces two similar triangles that can be used to compute the projected point on the image plane as shown in Figure 2.2. This process is repeated for the YZ plane in Figure 2.3 producing Equation 2.1 that computes the location on the image plane.

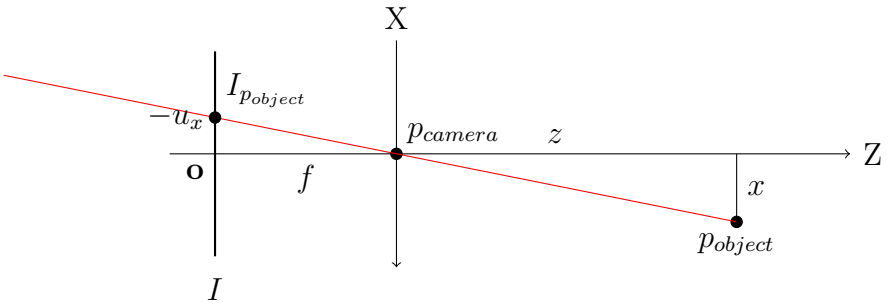


Figure 2.2: This figure demonstrates a ray of light from p_{object} passing through p_{camera} and intercepting the plane I at location $-u_x$. This is illustrated for the XZ axis.

$$(u_x, u_y) = \left(\frac{-fx}{z}, \frac{-fy}{z}\right) \tag{2.1}$$

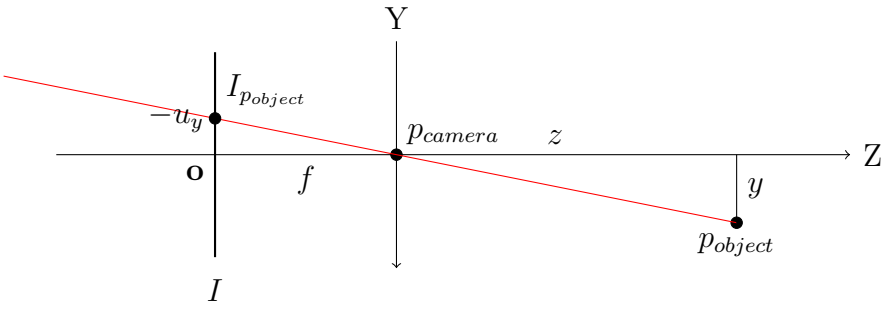


Figure 2.3: This figure demonstrates a ray of light from p_{object} passing through p_{camera} and intercepting the plane I at location $-u_y$. This is illustrated for the YZ axis.

The image plane currently has its centre located at the origin **o** on the X and Y axes. However, the image centre region does not always coincide with a rectangle centred at the origin in I due to inaccuracies in the camera mount created during the camera's construction (i.e due to inaccuracies when placing the lens relative to the image sensor), thus o_x and o_y are introduced to model the displacement of the optical axis. Typically, o_x and o_y are half the resolution of the image and is added to Equation 2.1 to ensure that the image plane is only defined in the positive XY axis as shown in Figure 2.4. This allows the image coordinates to start at position (0, 0), which is convenient for implementation. This updates Equation 2.1 to produce Equation 2.2.

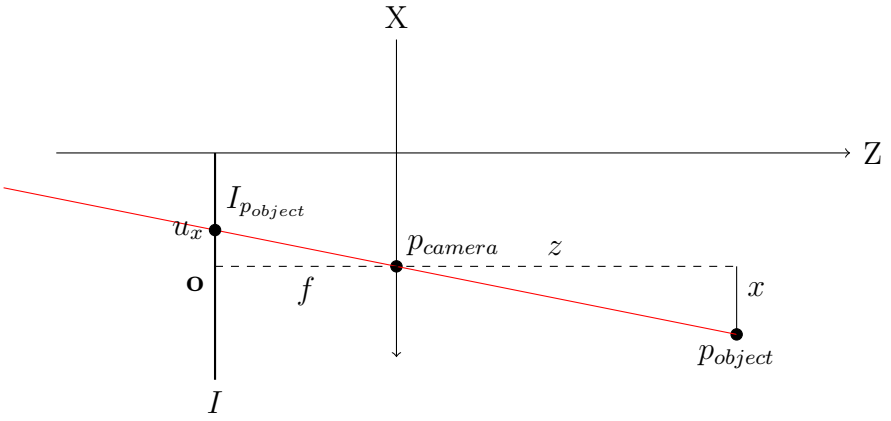


Figure 2.4: This demonstrates a ray of light from p_{object} passing through p_{camera} and intercepting the plane I with the plane shifted such that all intercepts are defined in the positive XY plane.

$$(u_x, u_y) = \left(\frac{fx}{z} + o_x, \frac{fy}{z} + o_y\right) \tag{2.2}$$

This produces the coordinates on the image plane I for the object p_{object}. For simplicity a matrix K, the camera intrinsic matrix, is constructed that allows this transformation to be applied through matrix multiplication as shown in Equation 2.3. In reality, the focal length f is two values f_x and f_y on a low-cost image sensor because each individual pixel (light sensor) on the image plane is rectangular rather than square. The actual value of the focal length, f_x, is the product of the physical focal length and the width of the light sensor. Similarly, f_y is the product of the physical focal length and height of the light sensor. The values for the physical focal length, the width of the light sensor and the height of the light sensor can generally not be obtained without physically opening the camera and directly measuring these values. The parameters f_x, f_y, o_x and o_y can be obtained through camera calibration.

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix} = \frac{K \begin{bmatrix} x \\ y \\ z \end{bmatrix}}{z} \tag{2.3}$$

2.1.2 Converting to World Space

Given the camera space projection (a 2D image) of the three-dimensional world, it is possible to compute the world space coordinates. The world space coordinates are the 3D positions relative to the origin on the camera. This is done by applying the inverse operations starting with the image space coordinates u_x and u_y to obtain two world coordinates. Since many points in the three-dimensional world can project to the same image location, z is used as the last world coordinate. This is achieved by performing the inverse transformation from the previous section.

$$K^{-1} \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix} z_{world} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.4}$$

2.2 Transformation Matrix

A transformation matrix is a combination of a rotation matrix and a translation vector. An n-dimensional rotation matrix R is an $n \times n$ matrix that comprises the rotational information of the local coordinate system. The elements contained in the rotation matrix are further described in Section 2.2.2. The size of this matrix depends on whether it is a 2D or 3D problem, using

a 2×2 or 3×3 rotation matrix. The translation vector T is a n-dimensional vector, depending on the dimensionality of the problem, containing the x, y and z translation. The 3D transformation matrix is constructed as follows:

$$\begin{bmatrix} R & T \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.5}$$

The transformation matrix is used to translate and rotate points in space. It is also used to convert between different coordinate systems.

2.2.1 Homogeneous Coordinates

Homogeneous coordinates are a way of representing points that allow for affine and projective transformations to be easily represented in the form of a matrix. A point $\mathbf{p}$ in $\mathbf{R}^2$ is represented by two values such as the points in two coordinates. This is extended by adding an extra dimension to the vector $\mathbf{p}$. The new representation is defined as the vector $\mathbf{p}^H$ with an additional dimension containing the value 1:

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \mathbf{p}^H = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.6}$$

To represent an arbitrary point in $\mathbf{R}^2$ as a homogeneous coordinate, $\mathbf{p}^H$ is constructed as shown in Equation 2.7. The variable s is a non-zero scaling factor for the vector. Scaling a homogeneous vector by a non-zero factor does not change the point.

$$\mathbf{p}^H = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \tag{2.7}$$

To compute the x and y values of a $\mathbf{R}^2$ homogeneous vector, one simply divides the vector by the scaling factor as shown

$$\mathbf{p}^H = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} sx/s \\ sy/s \\ s/s \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.8}$$

Vectors $\mathbf{p}^{H1}$ and $\mathbf{p}^{H2}$ illustrate two vectors that represent the same point.

$$\mathbf{p}^{H1} = \begin{bmatrix} 8 \\ 12 \\ 4 \end{bmatrix} \qquad \mathbf{p}^{H2} = \begin{bmatrix} 4 \\ 6 \\ 2 \end{bmatrix} = \frac{1}{2}\mathbf{p}^{H1} \tag{2.9}$$

Homogeneous coordinates can be extended to $\mathbf{R}^3$, which therefore require the use of a vector in $\mathbf{R}^4$.

$$\mathbf{p}^{H3} = \begin{bmatrix} sx \\ sy \\ sz \\ s \end{bmatrix} \tag{2.10}$$

For more information about homogeneous coordinates, the reader is referred to Hartley and Zisserman (2004).

2.2.2 Rotation Matrix

A rotation matrix is a way of rotating points in 2D or 3D Euclidean space. Given a 2D coordinate system, a point $\mathbf{p}$ can be rotated around the origin by an angle of ϕ radians in a counterclockwise direction to produce a new point $\mathbf{q}$ as shown:

$$\mathbf{q} = R\mathbf{p}$$

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{2.11}$$

This concept can be extended to 3D, producing three rotation matrices, one for each axis. Using the parameters θ, ϕ and α which represent the rotation in radians in the X axis, Y axis and Z axis respectively the following matrices are obtained.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.12}$$

$$R_y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{2.13}$$

$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

These rotation matrices can be consolidated into a single rotation matrix to produce the desired effect of the combined rotations.

$$R_{zyx} = R_z(\alpha)R_y(\phi)R_x(\theta) \tag{2.15}$$

Any rotation matrix R is an orthogonal matrix, therefore the following property holds: the inverse matrix can be computed as follows.

$$R^{-1} = R^T$$

It follows that the determinant of the rotation matrix is

$$\det(R) = \pm 1.$$

We can now use the combined rotation matrix R_{zyx} to rotate a point $\mathbf{p}$ around the origin as follows.

$$\mathbf{p}' = R_{zyx}\mathbf{p} \tag{2.16}$$

Similarly the point can be rotated in the opposite direction by computing the inverse rotation matrix or transpose $R_{zyx}^{-1} = R_{zyx}^T$. This is called the inverse rotation.

$$R_{zyx}^T\mathbf{p}' = \mathbf{p} \tag{2.17}$$

Equation 2.16 allows one to transform points around the origin. This will not work for rotating points around a fixed point.

To rotate the point $\mathbf{p}_1$ around the fixed point $\mathbf{p}_F$, $\mathbf{p}_1$ needs to be adjusted by subtracting the fixed point $\mathbf{p}_F$ from the point $\mathbf{p}_1$ as follows

$$\mathbf{p}_1^{Origin} = \mathbf{p}_1 - \mathbf{p}_F. \tag{2.18}$$

The point $\mathbf{p}_1^{Origin}$ can now be rotated using the rotation matrix R to obtain the new point $\mathbf{p}_{1'}^{Origin}$

$$\mathbf{p}_{1'}^{Origin} = R\mathbf{p}_1^{Origin}. \tag{2.19}$$

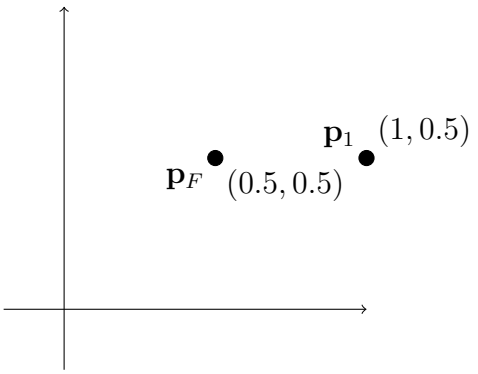
When the point has been rotated, it can be adjusted by adding the point $\mathbf{p}_F$ to produce the final point $\mathbf{p}_{1'}$. The rotation around a fixed point can be rewritten as shown in Equation 2.20. This is demonstrated in Figure 2.5.

$$\mathbf{p}_{1'} = R(\mathbf{p}_1 - \mathbf{p}_F) + \mathbf{p}_F \tag{2.20}$$

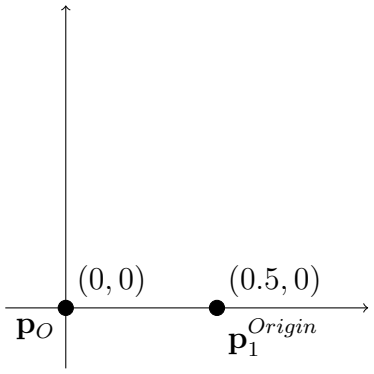
2.2.3 Building the Transformation Matrix

In order to rotate a point p about the origin and then translate it, a rotation matrix R and translation vector T can be applied as follows:

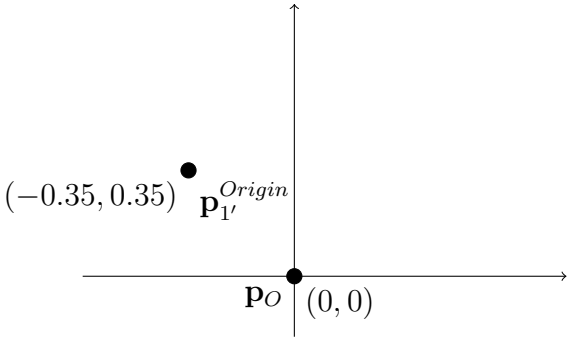
$$\mathbf{p}' = R\mathbf{p} + \mathbf{T}. \tag{2.21}$$



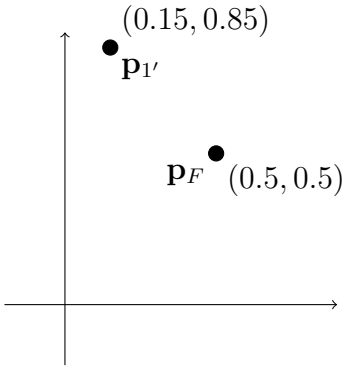
(a) Original points



(b) Adjusted point $\mathbf{p}_1$ to obtain $\mathbf{p}_1^{Origin}$



(c) The rotated point $\mathbf{p}_{1'}^{Origin}$ after a rotation by $\frac{3\pi}{4}$.



(d) $\mathbf{p}_{1'}$ The final position of $\mathbf{p}_1$ after a rotation by $\frac{3\pi}{4}$.

Figure 2.5: This figure illustrates how a rotation around a fixed point is performed. Subfigure (a) illustrates the scenario, the point $\mathbf{p}_1$ is the point that needs to be rotated around the fixed point $\mathbf{p}_F$. Subfigure (b) illustrates that each point has been moved to the origin by subtracting $\mathbf{p}_F$. This results in a new point $\mathbf{p}_1^{Origin}$. Subfigure (c) illustrates the rotation applied to $\mathbf{p}_1^{Origin}$ resulting in the point $\mathbf{p}_{1'}^{Origin}$. Subfigure (d) illustrates the points shifted by $\mathbf{p}_F$ resulting in a rotation of the point $\mathbf{p}_1$ around the fixed point $\mathbf{p}_F$ producing the point $\mathbf{p}_{1'}$.

This is called a rigid body transformation. The inverse of this transformation can be computed by applying the inverted operations on a point p' to obtain

$$R^T(\mathbf{p}' - \mathbf{T}) = R^T\mathbf{p}' - R^T\mathbf{T} = \mathbf{p}. \tag{2.22}$$

Equation 2.21 can be condensed by constructing a 4×4 transformation matrix M that will apply the rotation matrix R followed by the addition of the translation vector $\mathbf{T}$

$$M = \begin{bmatrix} & R & & \mathbf{T} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.23}$$

Given this transformation matrix M and a point $\mathbf{p}$ in homogeneous coordinates, the point can be transformed as follows

$$\mathbf{p}' = M\mathbf{p}. \tag{2.24}$$

Similarly, the original point $\mathbf{p}$ can be computed given the point $\mathbf{p}'$ and an associated transformation matrix M by multiplying by M^{-1} to obtain

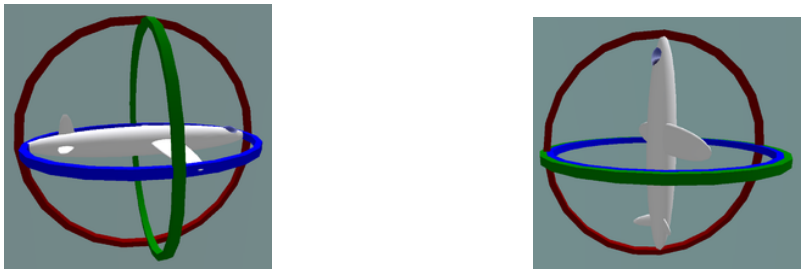
$$M^{-1}\mathbf{p}' = \mathbf{p}. \tag{2.25}$$

The inverse transformation matrix M^{-1} exists, and can be obtained from the components of the transformation matrix as

$$M^{-1} = \begin{bmatrix} & R^T & & -R^T\mathbf{T} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.26}$$

2.2.4 Quaternions

Another method of representing rotations is using quaternions (Hamilton, 1844). Quaternions are a way of representing rotations using only four values. Quaternions were created to get around the problem of gimbal lock when using Euler angles. Gimbal lock occurs when a series of rotations cause a loss of one degree of freedom in three-dimensions. This happens when two of the three axes are in a parallel configuration which cause a rotation around two of the axes to have the same effect, as shown in Figure 2.6. Quaternions approach the rotation from a different perspective. Quaternions use a vector to represent the rotation and can be easily converted to a rotation matrix and back. In computer graphics and the gaming industry, OpenGL uses rotation and transformation matrices during rendering. However, when computing rotations,



(a) (b)

Figure 2.6: Subfigure (a) illustrates a plane with its associated rotation axes. Subfigure (b) shows a configuration such that two gimbals are in alignment, causing gimbal lock.

developers tend to use quaternions when constructing the rotations that need to be applied to the game world. Once this quaternion is constructed, it is converted to its rotation matrix representation and applied to the objects in the scene. Quaternions are an extension of the complex numbers by defining two more values j and k; then the quaternions are

$$q = w + xi + yj + zk.$$

with the following properties

$$i^2 = j^2 = k^2 = -1 \tag{2.27}$$

$$ij = k \qquad jk = i \qquad ik = j. \tag{2.28}$$

To understand how quaternions can be used to represent rotation, q is rewritten as follows

$$q = (w, \mathbf{v}) \tag{2.29}$$

$$\mathbf{v} = (x, y, z). \tag{2.30}$$

Represented in this way, quaternions encapsulate the idea that there is a vector $\mathbf{v}$ that represents an axis in space around which the object will be rotated. Given an axis around which the rotation will be performed, the angle of rotation is required. The quaternions are always represented as a unit quaternion, i.e. the length of the quaternion is 1. The vector q is further adjusted to encode the rotational angle about the axis as follows

$$q = (\cos\theta, (\sin\theta)\mathbf{v}). \tag{2.31}$$

This allows us to store the angle of rotation in the quaternion representation, which can easily be extracted by computing $\cos^{-1} w$. The advantage of using

quaternions is that quaternion multiplication can combine a series of rotations to produce a single axis around which the rotation occurs. This is the combination of all the quaternion rotations. Quaternions also have the advantage of never suffering from gimbal lock, which is a problem when employing Euler angles in rotation. The quaternion is represented as a vector, which allows us to check the similarity of two quaternions by computing their dot product.

2.3 Least Squares

The method of least squares (Stewart, 2007) is a technique used to obtain an approximate solution to an overdetermined system of equations that has no exact solution. It is based on observation values and the values predicted by a candidate solution (expected/ estimated value) minimising the sum of the squared differences between observed value and the estimated value for each of the equations in the system. A residual is such a difference between an actual value and an estimated value. One common purpose of using the least squares method is to fit a data model to a set of observations. An example of this is to fit a collection of (x, y) points to a straight line, $y = Dx + C$, by determining the coefficients D and C that will minimise the sum of the squared residuals. For example given three points (0, 6), (1, 0) and (2, 0), a straight line needs to be fit to these points. Using the straight line equation the following equations are generated

$$0 \cdot D + C = 6 \tag{2.32}$$
$$1 \cdot D + C = 0 \tag{2.33}$$
$$2 \cdot D + C = 0. \tag{2.34}$$

This system has no solution. This is rewritten in the form of

$$A\mathbf{x} = b \tag{2.35}$$
$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} D \\ C \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} \tag{2.36}$$

The aim is now to find some D and C that best satisfies this system of equations. The basic idea of solving this is minimising S, the sum of the squared residual functions. The residual function r is the difference between the observed values y_i and the estimated values of the approximation $f(x_i, \mathbf{x})$, where

$$f(x_i, \mathbf{x}) = x_i D + C$$

in the example above.

$$r_i = y_i - f(x_i, \mathbf{x}) \tag{2.37}$$

$$S = \sum_{i=0}^{n} r_i^2 \tag{2.38}$$

Least squares problems are often divided into two categories: linear and non-linear least squares problems. The difference between linear and non-linear depends on the parameters that need to be determined. Linear least squares problems use models that are linear with respect to the model parameters. For example solving v and w in

$$y = vx^2 + wx + 2.$$

However, for example, the following equations are non-linear with respect to the parameters (v, w, σ, μ):

$$y = \sin v \cos x,$$

$$y = \frac{v}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) + w,$$

$$\text{or } y = w^2 x + vx.$$

These equations cannot be solved using linear least squares. An approach to tackling these types of equations will be briefly explained in Section 2.3.2.

2.3.1 Linear Least Squares

Minimising Equation 2.38 can be performed by setting the gradient of S to 0 as follows

$$\nabla S = 0\,. \tag{2.39}$$

In the linear case, solving Equation 2.39 generates a set of normal equations

$$A^T A\mathbf{x} = A^T b, \tag{2.40}$$

with corresponding solution

$$\mathbf{x} = (A^T A)^{-1} A^T b.$$

Solving Equation 2.40 is equivalent to solving the Equation 2.39. For more information refer to Strang (2016) on least squares. The solutions to computing a linear least squares problem and a non-linear least squares problem diverge before Equation 2.40.

2.3.2 Non-Linear Least Squares

Solving non-linear least squares problems requires more steps. This is due to the derivative of S in Equation 2.38. The gradient produces a function that does not generally have a closed-form solution. Since the derivative is non-linear, the solution has to be calculated iteratively with an initial estimate x_0 for $\mathbf{x}$ chosen. At each iteration, an incremental difference that the estimate needs to be updated by will be computed. Given the estimate of the solution $\mathbf{x}_j$ at iteration $j+1$ and the newly calculated incremental change $\Delta\mathbf{x}_{j+1}$, the estimate at iteration $j+1$ is calculated as

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta\mathbf{x}_{j+1}\,. \tag{2.41}$$

Since the derivative is non-linear at each iteration, the model is linearised using a Taylor series expansion around $\mathbf{x}_j$. In terms of the linearised model, the gradient is now a list of gradient equations $\frac{\partial r_i}{\partial \mathbf{x}} = -J_i^j$. At every iteration $\mathbf{x}_j$ the Jacobian $\mathbf{J}^j$ has to be recalculated due to the changing linearisation point used. Using the Jacobian, each iteration of the non-linear least squares solution is approximated as follows

$$\left(J^T J\right)\Delta\mathbf{x}^{j+1} = J^T \Delta b\,. \tag{2.42}$$

Solving non-linear least squares problems requires multiple iterations and depending on the initial value used, the technique can produce a value that is only locally optimal. In the linear case, Equation 2.42 has a constant Jacobian so just one step is needed.

2.4 3D Modelling

3D modelling is the process of developing surface representations of 3D objects. A surface consists of a group of faces. In 3D modelling, there are 3 concepts that are used to create a model: a vertex, an edge and a face. A vertex is a point in $\mathbf{R}^3$ defined by an x, y and z coordinate. Two connected vertices form an edge. Three or more coplanar edges in a cycle form a face. Creating these vertices, edges and faces produces a 3D model that can be used in a variety of ways. Examples of these concepts are shown in 2D in Figure 2.8. These 3D models are often used as designs in manufacturing or used as visual representations in the gaming industry. One fundamental approach to representing such models is a *point cloud*. A point cloud consists of a set of points in $\mathbf{R}^3$. Such a point cloud is generally obtained from a surface that has been sampled in $\mathbf{R}^3$. Figure 2.7 illustrates a cube generated in $\mathbf{R}^3$ with vertices, edges, faces and illustrates the normal to each face in teal. The modelling process consists of techniques that are categorised as polygon modelling, curve modelling and sculpting.

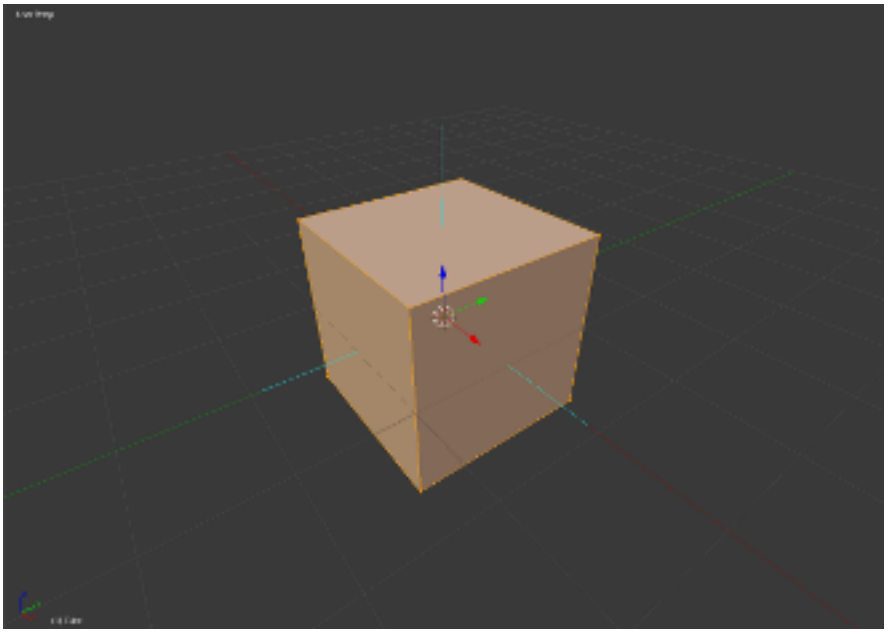
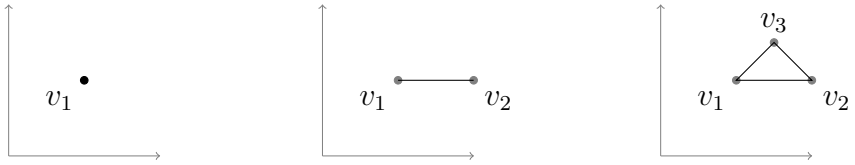


Figure 2.7: A cube displaying the normals to the surface in teal. Normals indicate the exterior direction that surface is facing. This helps the modelling application know the inside and outside of the model.



(a) A vertex (b) An edge (c) A face

Figure 2.8: Illustration of polygon modelling concepts. Subfigure (a) illustrates a single vertex. Subfigure (b) illustrates two connected vertices, this forms an edge between the vertices. Subfigure (c) illustrates three vertices that are connected. The edges form a cycle creating a face.

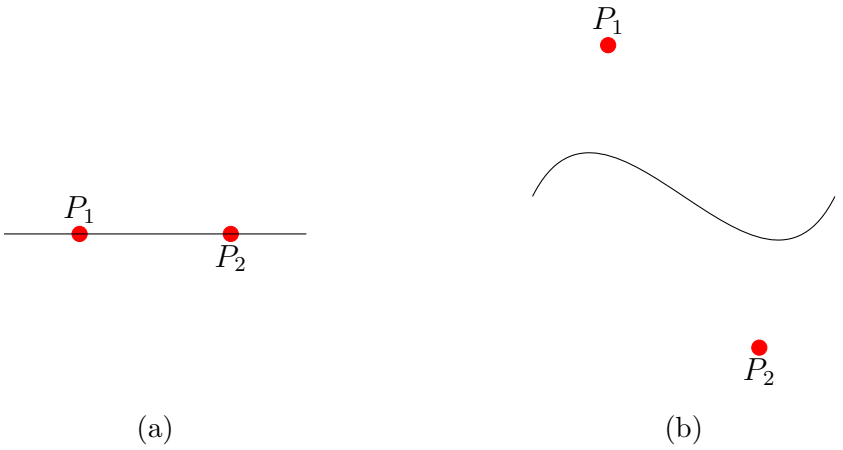


Figure 2.9: Subfigure (a) illustrates a line with weighted control points with the start and end points fixed. Subfigure (b) illustrates the result of moving the control points P_1 and P_2. It shows how moving the control points P_1 and P_2 change the path of the line.

2.4.1 Polygon Modelling

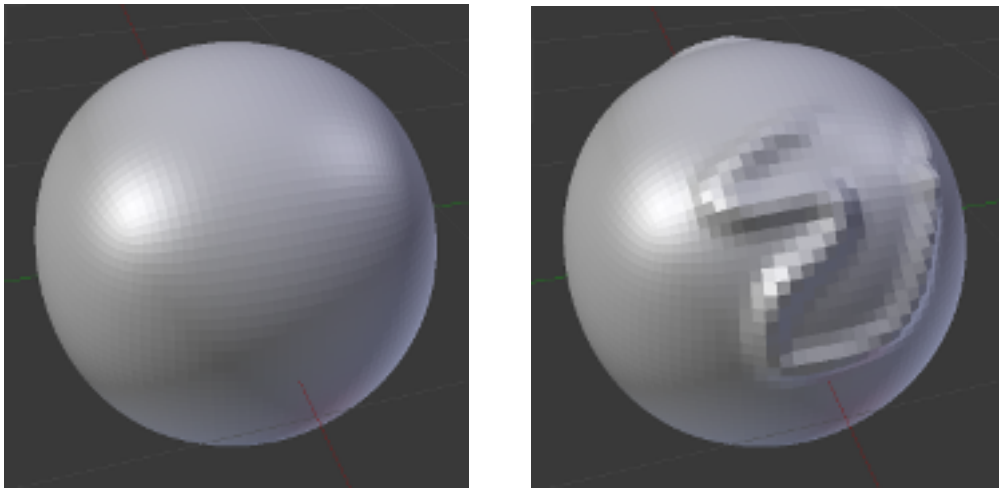
Polygon modelling uses the basic concepts of adjusting and creating vertices, edges and faces to create a mesh. Meshes are generally made by creating a sequence of vertices at specific 3D points and creating edges and faces from them. Once they have been created and adjusted to the creator's satisfaction, the mesh is complete.

2.4.2 Curve Modelling

Curve modelling uses the concept of weighted control points for a curve. A curve has control points that can be adjusted to pull the curve closer to the control point. These curves can be defined in multiple ways such as non-uniform rational basis splines (**NURBS**) (Schoenberg, 1964). Figure 2.9 shows how control points influence a straight line to produce a curve.

2.4.3 Sculpting

3D sculpting is relatively new to the 3D modelling world. It allows manipulation of a mesh in a way that is organic. Sculpting allows the user to smooth, grab and pull the surface as if it was made from clay. Various techniques are used to obtain these effects, such as using polygon modelling as described in Section 2.4.1 and volumetric based methods. Volumetric techniques create a 3D volume around the surface of a mesh and allow the user to push and pull



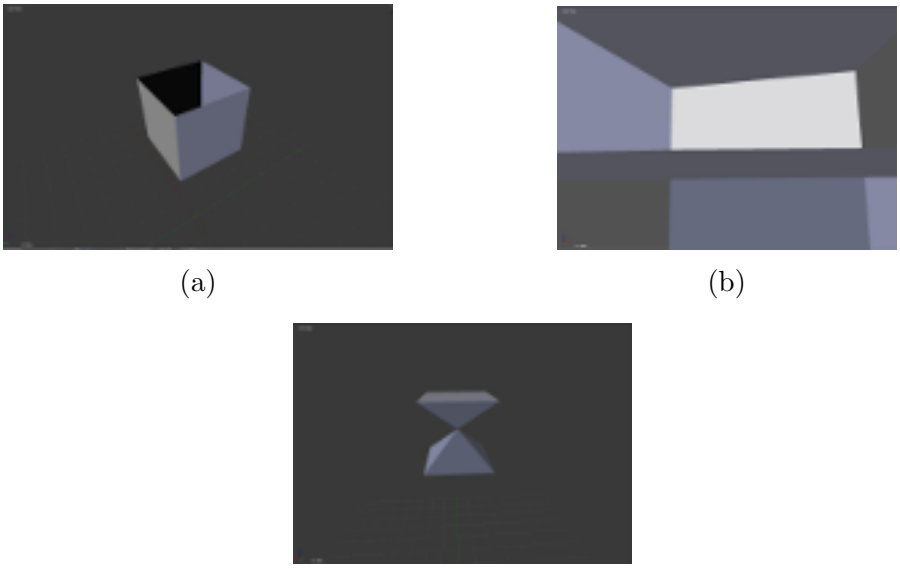
(a) (b)

Figure 2.10: This illustrates the sculpting of a sphere, before in Subfigure (a) and after in Subfigure (b).

the volume as demonstrated in Figure 2.10. This allows the user to add much higher detail to smaller areas while retaining the shape of the rest of the mesh.

2.4.4 Manifold Mesh

Meshes created in 3D applications do not have to adhere to the same rules as objects in reality. For this reason a mesh can fall into one of two categories, a manifold mesh or a non-manifold mesh. A manifold mesh is mesh that can be represented in the real world and a non-manifold mesh is a mesh that cannot be represented in the real world. A face of a manifold mesh has two sides, one side of this must face the internal region of the mesh and the other must face the external region of the mesh. The faces of the mesh are infinitely thin and therefore cannot have both sides facing an external or internal region, if this is the case it is a non-manifold mesh. The manifold mesh must contain a finite volume. To represent an object such as a plank, a rectangle with a finite width, depth and height must be constructed. Therefore, a manifold mesh would consist of a finite volume (a closed surface) and each face would have a side facing the internal and external region to be classified as a manifold mesh. If one of these conditions are not met, it would be classified a non-manifold mesh. If the internal region of the mesh is ambiguous, the mesh is a non-manifold mesh. A non-manifold mesh is a mesh that could also contains things such as disconnected vertices, edges and internal faces. Figure 2.11 illustrates three different non-manifold meshes.



(a) (b)

(c)

Figure 2.11: These subfigures illustrate different non-manifold meshes. Subfigure (a) illustrates a non-manifold mesh. This mesh is non-manifold due to the inability to distinguish the inside and outside region of the mesh (i.e it is not a closed surface). Subfigure (b) shows the inside of a cube mesh with additional internal face. This internal face has both of its sides facing the interior of the outer model and is therefore a non-manifold mesh. Subfigure (c) illustrates two pyramids connected by a single vertex. This is a non-manifold mesh because the connecting point of the two pyramids can be infinitely small point.

2.5 Kinect Sensor

The Kinect is a motion sensing device created by Microsoft for their home gaming console, the Xbox. This device has an RGB camera and an infra-red camera. Figure 2.12 shows the two iterations of the Kinect so far: the first version for the Xbox 360 and the second for the Xbox One. The infra-red camera allows the Kinect to generate an image that stores the depth information of the environment. This allows the Kinect to generate two images of the environment, a regular RGB image and a depth image, as shown in Figure 2.13. The outputs from the Kinect are called RGB-D images. The depth image allows us to compute a 3D point for each of the pixels, assuming valid depth data. Some regions of the depth image may be too close or too far for the sensor to correctly detect depth. If this is the case, those regions will be report as zero depth in the image. Assuming the calibration matrix K is known, the 3D points can be computed using the techniques mentioned in

(a) (b)

Figure 2.12: Subfigure (a) illustrates the Kinect for Xbox 360. Subfigure (b) illustrates the Kinect for Xbox One.

(a) RGB image (b) Brightened depth image

Figure 2.13: These figures are the example of the output from the Kinect sensor. Subfigure (a) is a RGB image from the Kinect. Subfigure (b) is (a brightened version of) the depth image.

Section 2.1.2.

2.5.1 Generating the World

The Kinect can be used to rebuild the world over time as a sequence of point clouds from the perspective of the Kinect. This is done by capturing a sequence of RGB-D images $I_1 \dots I_i \dots I_n$, then generating a point cloud $P_1 \dots P_i \dots P_n$ corresponding to each image. Each of these generated point clouds contains a collection of points from the perspective of the Kinect at a given time step. If the position and orientation of the camera is known for when each of the RGB-D images were taken, then the point clouds can in principle be translated, rotated and combined, assuming the scene is stationary (Hartley and Zisserman, 2004). This will in theory provide a single point cloud that represents all the captured information about the 3D environment. Essentially this

is the approach employed in the first part of the proposed system discussed in this thesis.

2.6 Summary

This chapter outlined key mathematical methods and concepts used in the thesis. This included the concept of the pinhole camera, which allows for transformations between world coordinates and image coordinates, as well as transformation matrices and quaternions for representing translation and rotations. This was followed by approaches to solving linear and non-linear least squares problems. The basic concepts of 3D modelling were introduced and an overview of the Kinect sensor was provided. The next chapter reviews existing techniques that are used for 3D reconstruction and mesh generation.

Chapter 3

Related Work

This chapter discusses work that has been done in previous systems for reconstructing 3D environments, including the problems encountered and solutions explored by other researchers. The problem of environment reconstruction is generally stated as the simultaneous localisation and mapping problem (SLAM). Typically, the trajectory of the camera needs to be recovered and the environment rebuilt at the same time. However, if this is not the case and the poses are known, this is referred to as mapping with known poses. This chapter reviews a number of solutions to the SLAM problem. Following this, alternative approaches to 3D reconstruction are discussed. After tackling the problem of 3D reconstruction, a series of techniques for generating meshes from point clouds are discussed.

3.1 Motion Estimation

Motion estimation is the process of computing the transformation between sequential 2D images. Motion estimation problems are generally addressed using hardware solutions, including mounting sensors such as optical encoders and magnetic sensors to provide odometry. These sensors can be mounted on a robot or vehicle to provide an estimation of motion during data capture. This is used in combination with inertial motion units to detect changes in orientation and estimate the vehicle's rotation.

The proposed system considers using a hand-held camera which does not capture odometry information thus we only discuss approaches of this kind. The techniques referred to below are often called registration or scan-matching. They operate by 'registering' the initial or source image to a 'reference' or destination image, which computes the motion from the initial image to the reference. Each of these images can be converted to a set of points in 3D space.

Iterative Closest Point

The iterative closest point (ICP) algorithm uses multiple techniques to compute the motion between two sets of points. The algorithm has many variants since parts of the algorithm can be replaced by alternative techniques. The ICP algorithm has three iterated stages: the first is point association, the second is point rejection, and the last is a minimisation step. ICP works by iteratively attempting to make new associations between different point pairs and minimise an error function during each iteration. There are multiple ways in which the point associations, the point rejections and the minimisation of the error function can be performed. Rusinkiewicz and Levoy (2001) have compared various combinations of these approaches to find the most efficient variants of ICP.

The first variant of ICP, detailed in Besl and McKay (1992), creates its association between the two sets of points by computing the Euclidean distance between a point from the first set to each point in the second. The point in that second set that has the smallest distance is associated with the point in the first set. At this stage associations with a large value can be discarded if necessary as the point rejection step, depending on the overlap area that the two point sets have. After this, the ICP algorithm attempts to minimise an error function that computes the estimated transformation matrix from the first point set to the second. The transformation is estimated in two parts, the first attempting to compute the quaternion that represents the rotational component of the transformation by using the approach in Horn (1987). The algorithm then estimates the translation component as the change in the centres of mass between the point clouds to construct the estimated transformation matrix. The point cloud is transformed by the estimated transformation matrix and the process is repeated by performing all three stages again due to the point association technique used. The association technique has no way to ensure that the matches are correct and thus its results are iteratively refined. Then a point-to-point error metric is used to determine the quality of the estimated transformation in which the sum of the squared distance between points in each correspondence pair is calculated. This process stops once the error has reduced to an acceptable tolerance or a certain number of iterations have passed.

The next approach by Low (2004) replaces the point association and minimises a different error function compared to Besl and McKay (1992). To create the association, a projective data association approach is used. Two depth images are used to store the point cloud data and the association is made between their pixels. This occurs by projecting a pixel from the source depth image onto the target image; the position where the projection lands is the associated pixel. Since this method uses a projective data association technique, it requires that the changes between the two images be relatively small

because linearisation is performed during motion estimation. This technique uses the associated pixels from the depth images to compute the 3D position of each pixel. Once the associations of the 2D pixels are made and are converted to 3D, the point-to-plane error metric is used (Chen and Medioni, 1992). The point-to-plane error metric aims to minimise the sum of the squared distance between the source point and the tangent plane at its correspondence destination point. The point-to-plane error metric allows point rejection by testing two metrics; the similarity between the tangents to the source and destination point and the Euclidean distance between these points. If tangents deviate beyond a threshold or the euclidean distance exceeds a set threshold, the point pairs are rejected. This technique is used in the proposed system, and is further discussed in Section 5.4.

The scan-matching techniques often only use a subset of points that have strong associations between them to improve robustness and reduce computational complexity. However, other techniques such as Steinbrücker, Sturm and Cremers (2011) make use of all the pixels present in an image. Steinbrücker, Sturm and Cremers (2011) present an energy-based approach for RGB-D images. An energy function is presented that aims to obtain the best rigid body transformation from one RGB-D image to the other. This technique aims at minimizing the back-projection error to find a rigid body transformation from the special Euclidean group SE(3) representing the camera motion such that the second image exactly matches the first. For large motion this technique implements a coarse-to-fine approach iteratively improving on the rigid body transformation. The presented approach was compared to a state-of-the-art implementation of ICP known as Generalized-ICP (GICP). GICP proved more robust with larger camera motion than the technique presented; however, Steinbrücker, Sturm and Cremers (2011) provided more accurate results in regions of smaller motion while being faster than the GICP method.

Huang and Bachrach (2017) use a feature-based approach to achieve real-time motion estimation using the Kinect sensor. This technique is designed to operate on a low cost device mounted on a quadrocopter to estimate its local position and stabilize the quadrocopter for autonomous flight. This approach combines multiple techniques to provide high performance with six stages to the motion estimation: The first stage is image preprocessing, where the incoming depth image is processed through a Gaussian filter, the RGB image is converted to greyscale and a Gaussian pyramid is constructed. This allows the detection of features on a larger scale. The second stage is feature extraction; the FAST (Rosten and Drummond, 2006) feature extractor is used on each level of the pyramid to extract the features, and the depth image is used to extract the associated depth. The third stage is generating an initial rotation estimate; this is performed using a homography-based 3D tracking algorithm (Mei, Benhimane, Malis and Rives, 2008). This technique computes a warping function that is used to warp the source image to the target image. This al-

lows the system to constrain the search window for the feature-matching. This stage could also use the addition of an IMU to compute the initial rotation. The next stage is feature matching, which computes the sum of the differences between the pixels of the feature from the source image and the destination image (Howard, 2008) as the metric for feature similarity. The fifth stage is inlier detection; this ensures that the Euclidean distance between two features at one time should match their distance at another time. Finally, stage six does the final motion estimation using Horn's absolute orientation method (Horn, 1987) to compute rotation and a non-linear least squares solver to minimise the reprojection errors of features in the environment.

Estimating the motion of the camera using ICP is generally performed using an inter-frame approach. This is done by using one image as the reference frame and computing the motion from the reference frame to the next frame called the target. Computing the camera's motion in this way causes the location estimate to drift as the motion errors accumulate for every image processed. To prevent this from happening, the system needs to correct this drift using SLAM techniques and loop closure techniques.

3.2 SLAM Solutions

Rebuilding scenes in 3D requires knowing the position of the camera and the distance from the sensor to objects in the scene. The simultaneous localization and mapping problem (SLAM) was originally formalized in 1986 by Smith and Cheeseman (1986) in which a generalised method for estimating the spatial location of objects was presented, followed by improvements in Smith, Self and Cheeseman (1987). Informally, the SLAM problem involves determining where a given robot or vehicle is within an environment while simultaneously mapping the environment. This section outlines a number of approaches to tackling the SLAM problem.

Informally SLAM works by tracking key features, known as landmarks, detected in images or other sensory data to determine the camera or sensor's position and orientation. In an extreme case, every pixel of an image might be considered a landmark.

A probabilistic formulation of SLAM was pioneered in the work of Smith and Cheeseman (1986) on probabilistic spatial representation. This laid the groundwork for a 6 degrees of freedom model for estimating relationships between camera position and environmental objects in three dimensions, as well as the associated uncertainty. Durrant-Whyte (1988) produced work on transforming the uncertain points, curves, and surfaces from one coordinate frame to another.

Major breakthroughs occurred in the late 1990's and early 2000's due to the decreasing price of sensory devices and faster processors. This gave standard

desktop computers and mobile devices the ability to adequately perform the complex calculations required by SLAM systems.

SLAM systems are categorized as being online SLAM or offline SLAM; online SLAM focuses on estimating the current location of the sensor, typically while streaming data into the system while offline SLAM attempts to reconstruct the entire trajectory of the sensor correcting the previous estimates of the sensors location.

The two main variations of SLAM solutions, derived from the Bayes filter (Arulampalam, Maskell and Gordon, 2002), are the extended Kalman filter (EKF) (Gamini Dissanayake, Newman, Clark, Durrant-Whyte and Csorba, 2001) and the Rao-Blackwellized particle filter (RBPF) (Montemerlo, Thrun, Koller and Wegbreit, 2002). Both techniques can be performed online or offline. Other offline SLAM techniques exist, such as GraphSLAM. SLAM algorithms generally use a motion model for predicting the motion of the camera and updating the estimated position. These motion models are generally velocity or odometry-based. Extensions to these methods such as submapping and state augmentation are discussed in Bailey and Durrant-Whyte (2006). The EKF and the RBPF both use landmarks to help estimate the correct camera trajectory. The use of landmarks allows the system to correct the current estimated pose of the robot. Further detail is provided on these two main approaches to SLAM below.

3.2.1 Extended Kalman Filter

The extended Kalman filter approach to the SLAM problem developed by Gamini Dissanayake *et al.* (2001), was the first solution to the SLAM problem with proof that the system converges as more observations are taken. That is, the covariance of each landmark decreases as successive observations are made. However, Harris and Pike (1988) were the first to use Kalman filters to accurately estimate the position of 3D landmarks from an image sequence.

The approach developed by Gamini Dissanayake *et al.* (2001) is used to estimate hidden model states using observation data. The approach estimates model states by fusing multiple sources of noisy data to find a joint state estimate. However, the Kalman filter makes two key assumptions: that the uncertainty of the position follows a Gaussian distribution and the model states are linear. The extended Kalman filter (EKF) (Kalman, 1960) introduces generalizations that allow the Kalman filter to be applied to non-linear models. This is done by performing local linearisation with a first-order Taylor expansion. This approach maintains a state estimate of the robot as well as landmark locations found in the environment.

The estimator maintains a matrix that stores the covariance between each estimated covariance of their pose, as well as the variance of the robot's pose. This requires a large amount of memory to maintain the covariance matrix,

limiting this approach to considering only a couple of hundred landmarks. The accuracy of the map and the vehicle location will converge to a lower bound defined by the initial uncertainty of the vehicle. (Generally, this is irrelevant when building a new map since the starting position is known.)

Variants of the extended Kalman filter, as described in Thrun (2005), two notable examples are the unscented Kalman filter and the sparse extended information filter.

Unscented Kalman Filter

This gives a lower uncertainty and provides better linearisation of the model dynamics than the Taylor expansion with higher order non-linear functions (Julier and Uhlmann, 1997). This is slower than the standard EKF and still assumes that the error distribution is Gaussian.

(Sparse) Extended Information Filter

The EIF uses the canonical form (Maybeck, 1982) to represent Gaussian distributions. Instead of a covariance matrix, a precision matrix, the inverse of the covariance matrix is maintained. In addition, an information vector, which is the precision matrix multiplied by the mean is maintained. There is little to no difference in performance between EKF and EIF: The EIF method provides a slow prediction step and an efficient correction step, while the EKF algorithm provides an efficient prediction step and a slow correction step (Nettleton, Gibbens and Durrant-Whyte, 2000). However, the EIF forms the basis for the sparse extended information filter (Thrun, Liu, Koller, Ng, Ghahramani and Durrant-Whyte, 2004). This is an approximation that decreases the EIF's computational complexity. This is achieved by performing a sparsification step that removes the dependence between the locations of landmarks and the robot. Removing the dependence is achieved by setting the values in the precision matrix to zero. The SEIF does this by using two sets of landmarks: an active set and passive set. The active set of landmarks are a set of landmarks in the area that include the currently observed landmarks. This set of landmarks is usually set to a fixed size. The SEIF maintains all dependencies for the active landmarks while it removes the dependence between the robot and the passive landmarks.

3.2.2 Rao-Blackwellized Particle Filter

The Rao-Blackwellized particle filter introduced by Montemerlo *et al.* (2002) or more commonly known as fastSLAM provides a solution to the SLAM problem that avoids assuming a Gaussian distribution for the motion model. This approach can represent thousands of landmarks (as opposed to EKF,

which can generally only represent hundreds due to memory constraints). The Rao-Blackwellized particle filter was based on the work in three key papers: Murphy (2000), Doucet, de Freitas, Murphy and Russell (2000) and Gamini Dissanayake *et al.* (2001).

The Rao-Blackwellized particle filter is the combination of two ideas. The first is the particle filter, similar to the Monte Carlo localization algorithm by Dellaert, Fox, Burgard and Thrun (1999). The particle filter generates samples around a particle and computes an error for each sample, from these samples it selects a subset of particles with the lowest error as the new set of particles, this is discussed in more detail later in this section. This process filters out particles with high error values. The particle filter is used to sample different estimates for the pose of the camera. The second is making conditional landmark estimates by using EKFs. The RBPF uses a key insight from the Rao-Blackwell theorem (Blackwell (1947) Rao (1945)) to lower the uncertainty of the estimated camera position in the system. Instead of jointly estimating the position of the camera and the landmarks directly, it uses a particle filter to estimate the trajectory of the camera only. If the best estimate for the trajectory is known, the landmark locations can be retrieved. This exploits the dependency between knowing the location of the estimated camera and the location of the landmarks. The particle filter simulates having hundreds of samples (particles), known as the pose hypotheses. The particle filter will then apply a motion command based on the motion model to all the particles. It assumes that each particle that is moved is perturbed by Gaussian noise. For each of these particles, the system computes the best estimate for the new position of the camera. If the system uses RGB-D data, the system can compute the best particle as the one with the smallest projection error from all the estimated landmarks. Once the system has computed the error of the particles, it selects a new sub set of particles with a higher chance of picking samples with low error and a lower chance of picking particles with larger errors. This causes the particles with high errors to be less likely to be chosen, and thus slowly eliminates the particles with high errors over time. This is known as particle depletion.

3.2.3 GraphSLAM

Another popular version of offline SLAM employs a PGM-based approach (Lu and Milios, 1997). It is a least squares approach to the SLAM problem and attempts to recursively minimise the error function. This approach builds a graph that connects each position and an observation as a node with motion estimates represented by the edges in the graph, as shown in Figure 3.1. The motion estimate probabilistically constrain the relative pose at each node. This allows the graph to relate the position of nodes to those of other nodes. Thus, if the system revisits a location and views the same landmarks again, it can

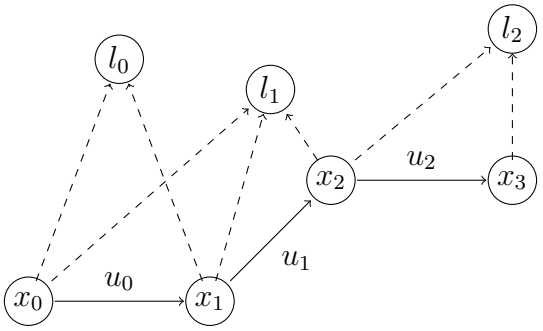


Figure 3.1: This illustrates the graph created during GraphSLAM. The nodes x_0 to x_3 illustrate the position and edges to the nodes l_0 to l_2 illustrate the observations from each position. The edges u_0 to u_2 represent the estimated motion between the nodes.

create additional constraints between the current node and other nodes that have viewed the same landmark (known as a virtual constraint). Using this idea, the system needs to find a node configuration that maximises a goodness-of-fit metric parametrized by the motions on the edges. Once the metric has been maximised, the system has reconstructed the optimal trajectory under the model. With this optimal trajectory, the system can generate the map using RGB-D data. This optimal node configuration produces a smoothing effect of the camera's trajectory. This approach to SLAM is often referred to as smoothing and mapping (SAM). A new incremental approach of SAM has been created by Kaess, Ranganathan and Dellaert (2008). GraphSLAM can also be run online directly by optimising the graph after adding a new node to the graph.

Alternative techniques to 3D reconstruction are discussed in the subsequent section.

3.3 Alternative Solutions

This section describes alternative techniques and approaches to 3D reconstruction. Specifically, we describe the approaches of bundle adjustments (Triggs, McLauchlan, Hartley and Fitzgibbon, 1999), the Kinect Fusion algorithm (Newcombe, Izadi, Hilliges, Molyneaux, Kim, Davison, Kohli, Shotton, Hodges and Fitzgibbon, 2011*a*), parallel tracking and mapping (Klein and Murray, 2007) and dense tracking and mapping (Newcombe, Lovegrove and Davison, 2011*b*).

3.3.1 Bundle Adjustments

Bundle adjustments is used as the final step in most 3D landmark-based reconstruction algorithms. It follows a similar approach to the graph-based approach described in Grisetti, Kummerle, Stachniss and Burgard (2010). The term bundle refers to a group of light rays leaving a 3D feature and converging on a camera centre, which are adjusted optimally with respect to both feature and camera positions. This is done by posing the problem as a geometric estimation problem that uses the combined 3D feature coordinates, camera poses, and calibration as the parameters. The parameters are estimated using a non-linear least squares formulation with the cost function based on the projection errors. Triggs *et al.* (1999) describe the bundle adjustments algorithm, along with the various assumptions, misconceptions and techniques around it.

3.3.2 Kinect Fusion

The Kinect Fusion system by Newcombe *et al.* (2011*a*) is a system designed to create a 3D reconstruction of a fixed area and incrementally update the scene. This system is able to leverage GPU programming to achieve real-time performance. To reduce noise, this system uses bilateral filtering (Tomasi and Manduchi, 1998) and a volumetric-based approach is used to build the model, as described in Curless and Levoy (1996*a*). This approach averages the estimated location of each surface measurement from the environment using the RGB-D images by means of a truncated signed distance function (TSDF). The volumetric-based approach removes the need for landmarks and uses scan-matching techniques to best correct the motion by using a frame-to-model match. Typically, systems use a frame-to-frame approach which computes motion from the one frame to another, these frames are input to the system from the real world. Using the frame-to-frame approach, relative motion is computed between the frames, however this becomes an issue when the motion is not computed accurately enough. The frame-to-frame approach assumes that the camera is in the position on the first input frame, over time with the inaccuracies the position of the camera and the expected view for the next frame will not be aligned. The misalignment of the position and the frame is known as drift. The frame-to-model approach does not assume the position of the camera, it uses the camera to compute a virtual image from the estimated location in the environment. The virtual image is a synthetic image generated from previous input images, this is described in more detail in Chapter 5.

The main difference in this system, compared to other techniques, is its ability to refine the model using the method described in Curless and Levoy (1996*a*). This filters out noise and helps reduce the drift during the motion estimation by using a frame-to-model approach as opposed to a frame-to-frame

approach. The frame-to-model approach is achieved by the TSDF volume. The TSDF volume is a 3D volume of TSDF values, each one representing a sub-volume in 3D space. Each of the TSDF values in the sub-volume are positive or negative depending on the distance and direction the sub-volume is from the closest surface measurement. The model that is represented in the TSDF volume is defined as having a surface at all boundaries between positive and negative values. These are called zero-crossings. Multiple range images are integrated into this volumetric representation. This allows the TSDF values to take on an average value which reduces the sensor noise.

Correcting the motion by using the frame-to-model approach only works when precise sensors such as the Microsoft Kinect are used. In this case, the accuracy of the depth information allows the motion estimation to be more accurate, thus reducing the drift. The frame-to-model approach raycasts the volumetric model to create a virtual image and then computes the motion from the virtual image to the incoming image instead of using the two frames from an image sequence. Using this technique, the position of the camera view is not assumed at any point and is generated from the estimated location of the camera. The position of the camera is based on the relative position of the volumetric model. This system has two main limitation in that it is designed to reconstruct a fixed volume and due to the frame-to-model approach in its motion estimation, fast camera movements and motion blur will prevent the system from computing the motion between the model and frame correctly. This system can be used with SLAM techniques such as GraphSLAM to optimise and correct the trajectory. For the work reported in this thesis, this system is chosen to reconstruct the environment due to its ability to reduce noise and its robustness to changes in the scene (Izadi, Kim, Hilliges, Molyneaux, Newcombe, Kohli, Shotton, Hodges, Freeman and Davison, 2011). This technique is presented in Chapter 5.

Extensions to the Kinect Fusion system have been made in Whelan and Kaess (2012) by removing the fixed volume limitation using a wrap-around indexing system for the volume. This removes the limitation that fixes the volume's location and allows the volume to move large distances while performing the reconstruction. This is achieved by discarding old sections of the TSDF volume and overwriting them. This feature is incorporated into the system that is proposed in this thesis.

3.3.3 Parallel Tracking and Mapping

Klein and Murray (2007) developed a real-time mapping and tracking algorithm named PTAM (parallel tracking and mapping). This is a feature-based reconstruction algorithm that focuses on reconstructing the environment using a single RGB camera. This system is divided into two threads. One thread is used for tracking camera motion and the second thread for building the

map. The map building uses batch optimisation techniques (such as bundle adjustments) to allow it to run in real-time. The main contribution of this algorithm is that the authors focus on computing the motion of the camera (which is hand-held) without odometry commands as in SLAM.

3.3.4 Dense Tracking and Mapping

The dense tracking and mapping system was developed by Newcombe *et al.* (2011*b*). It was developed as a single-camera (monocular) system that creates dense maps, avoiding the reliance on feature-based techniques. It creates a dense 3D model, and uses it for frame-to-model tracking similarly to the Kinect Fusion system. The model is built in patches using multiple frames of the same area for the reconstruction. As with the Kinect Fusion system, the system is stated to be limited by fast camera motion and motion blur.

3.3.5 Benchmarks

General data sets and evaluation tools for SLAM have been presented in Sturm, Engelhard, Endres, Burgard and Cremers (2012) for the SLAM community. These include 39 data sets of indoor environments and automated tools to compute the relative trajectory error from the ground truth. In their work they use one performance metric, the accuracy of the estimated camera trajectory. This only focuses on the trajectory that was estimated and does not consider the accuracy for the reconstructed environment. In these data sets, the camera is moving rapidly between frames, and the images are often blurred due to this motion.

3.4 Depth Estimation

In order to use the techniques mentioned in the previous section, the system may need to estimate the depth of landmarks or regions and compute the camera's motion between successive frames.

With increases in processing power and faster algorithms, much progress has been made in estimating depth from camera information using structure-from-motion (SFM) and multi-view stereo (MVS) systems (Hartley and Zisserman, 2004). Systems that use monocular setups usually need to estimate the depth of the scene using a sequence of images. This tends to cause the system to operate offline due to the computationally complex operations required to estimate the depth (Hartley and Zisserman, 2004). These systems are often based on Harris and Pike (1988), which used an image sequence to generate 3D features and accurately estimate their positions using Kalman filters. These

systems have managed to achieve real-time performance in recent years with systems such as that described by Davison, Reid, Molton and Stasse (2007).

Stereoscopic (MVS) systems make use of two or more cameras that allow the system to estimate depth by performing triangulation. These systems require two cameras next to each other to take images simultaneously to generate two images of the same environment from different viewpoints. Using these viewpoints, it is possible to estimate the depth of a feature visible from both images, as described in Hartley and Zisserman (2004).

Many modern systems make use of laser range finders to estimate depth. These are called time-of-flight (ToF) sensors. A ToF sensor uses the speed of light to compute distance from time differences: It emits a light beam and measures the time taken for its reflection to be sensed. This removes the computational complexity of estimating depth by techniques such as triangulation and is able to compute the depth to each pixel as opposed to triangulation which generally only computes the depth to each feature, but requires special hardware.

3.5 Mesh Reconstruction

Reconstruction algorithms do not initially produce meshes; instead they produce point clouds. A point cloud is a set of points in 3D space that may represent a surface in the environment. Since it is a set of points, no information about the objects' surface connectivity has been explicitly recorded. Instead, this needs to be reconstructed. Reconstructing the surfaces from a set of points can create many possible interpretations of the surface of the object. Remondino (2003) summarizes a number of techniques for converting point clouds to meshes.

Point cloud based mesh construction techniques can be separated into two types:

- Systems based on object measurements, such as triangulation and laser scanning.
- System that do not use measurements, such as computer animation software and 3D modelling applications.

Algorithms based on object measurements are the most relevant to this study, and generally are divided into the following categories.

Surface-oriented approaches focus on generating surfaces and do not distinguish between closed (water tight) or open surfaces, such as in Hoppe, DeRose, Duchamp, McDonald and Stuetzle (1992).

Volume-oriented approaches focus on constructing a closed surface, these approaches are commonly based on the Delaunay triangulation of a set of 3D points, such as in Boissonnat (1984) and Curless and Levoy (1996*a*). The most commonly used derivative of Delaunay triangulation is known as the Crusts algorithm (Amenta, Bern and Kamvysselis, 1998). Improvements on the original Crusts algorithm have let to the Cocone algorithm (Dey and Giesen, 2001) and, later, the water-tight Cocone algorithm (Dey and Goswami, 2003).

Both of the categories above include methods that aim to reproduce the surface of the object using either all or a subset of the points in the point cloud. Surface oriented approaches are further subdivided into the following groups.

Parametric representation
These use parametric equations to represent surface patches. Multiple patches can be pieced together to form a continuous surface. This usually uses formulations such as B-splines or Bezier curves such as in Terzopoulos (1988).

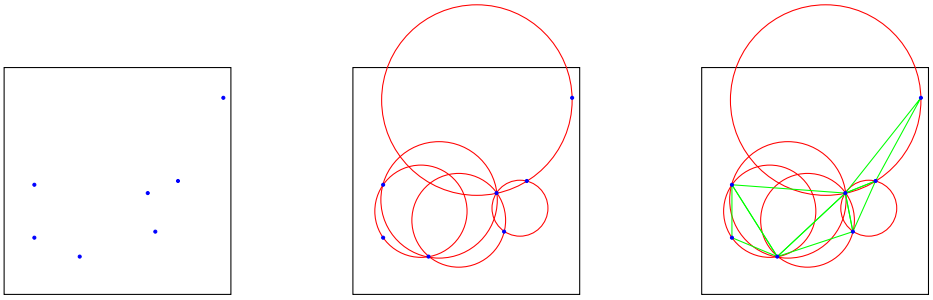
Implicit representation
Attempts to find a smooth function that passes through all the points such as in Gotsman and Keren (1998). This however, is susceptible to noise in the point cloud data.

Approximated surfaces
These do not always contain all the original points, but add new points as close as possible to points in the original point cloud. Such methods use a distance function to approximate the correct mesh such as in Hoppe *et al.* (1992).

Interpolated surfaces
This approach uses the original points and interpolates between these points. This can result in points in the original set being excluded from the final surface such as in Bernardini *et al.* (1999). This is discussed further in Chapter 6.

3.5.1 Mesh reconstruction techniques

One of the most popular techniques used in surface reconstruction is Delaunay triangulation (Delaunay, 1934). This works by considering spheres between three random points in a subset of points: if a sphere does not contain any other point inside it, the sphere is considered valid. The three points of a valid sphere are used to produce a triangle that is used as a face in the model. The triangulation procedure also attempts to maximize the minimum angle of all the internal angles of all the triangles. This prevents the algorithm from



(a) (b) (c)

Figure 3.2: Subfigure (a) shows some 3D points projected onto a 2D plane. Subfigure (b) shows some possible circles generated during the Delaunay triangulation. Subfigure (c) shows the generated triangles using Delaunay triangulation.

generating skinny triangles, where the base is very small compared to the height. Delaunay triangulation forms the basis for many other reconstruction techniques including those described in Boissonnat (1984), Bernardini *et al.* (1999) and Isselhard, Brunnett and Schreiber (1997).

Boissonnat (1984) describes a surface-orientated method using nearest neighbours structures and a projective approach to generate the mesh. It aims to operate in a small local area and projects the points from the point cloud onto a plane and creates triangles between these projected points on the plane. The triangles created on the 2D plane will be created in the 3D point cloud as well. Figure 3.2 illustrates the points projected on a 2D plane in a local region and the circles generated during Delaunay triangulation which will create triangles between them.

A more powerful and popular version of the work by Boissonnat (1984) is known as the Power Crust algorithm, as described by Amenta, Choi and Kolluri (2001). The Power Crust algorithm produces a surfaced mesh and an approximate medial axis. The medial axis transform (Bium, 1964) is a skeletal shape representation which has been proposed as a tool for various applications in shape recognition and manipulation. It represents a solid by the set of maximal balls completely contained in the interior of the point set, as contemplated in Boissonnat (1984). The approach is to first approximate the medial axis transform of the object and to then use an inverse transform to produce the surface representation from the medial axis transform.

One problem with the technique of Delaunay triangulation is that it assumes the availability of arbitrary precision mathematical operations. This results in situations where floating point errors and overflow occur, so that fixed-width binary representations can cause this method to incorrectly incorporate triangles. A different technique to Delaunay triangulation was devel-

oped by Bernardini *et al.* (1999), known as the ball pivot algorithm (BPA). This technique follows an analogous method to building triangles from spheres, constructing the mesh by simulating a sphere rolling over the point cloud. Instead of finding spheres to fit the points exactly, a single sphere of a certain fixed radius is chosen in advance. This sphere is rolled over the point cloud and when the sphere touches three points a face is generated between them. The sphere is initially placed on the point cloud by finding three random points that the surface of the sphere can touch at the same time. The sphere should not contain points inside and the direction of the points (surface normals) should be in the same direction. Once placed on the initial three points the algorithm creates a triangle using the points as vertices. This generated triangle is the first face that the algorithm has generated. While each of its edges form a boundary of the reconstructed area. The algorithm selects one of these edges and pivots the sphere along the edge until the sphere makes contact with another point from the point cloud. A new triangle is created between the edge that the sphere is pivoting on and the new point. The process of pivoting the sphere along an edge and generating triangles continues until the sphere does not make contact with any new points and can not create any new triangles. This technique has two main challenges. The first is that each point requires surface normal information. The second is that the reconstruction quality depends on the initially selected sphere size. These additional challenges are balanced by the algorithm's inherent simplicity and is chosen to be incorporated into the proposed system due to this. It is discussed in more detail in Chapter 6.

Another common reconstruction technique is Poisson surface reconstruction (Kazhdan, Bolitho and Hoppe, 2006). This poses the surface reconstruction problem as a spatial Poisson process (Weil, Hug, Baddeley, Capasso, Bárány, Villa and Schneider, 2006). It aims to approximate a 2D slice of the 3D model using a function. This is achieved by imagining a 2D plane passing through a model and finding a function that represents the intersection of the model with the plane. Further improvements of Poisson surface reconstruction include implementing a parallelized Poisson surface reconstruction technique, as well as the ability to stream point cloud data into memory for large scale models, as described by Bolitho, Kazhdan, Burns and Hoppe (2009).

Gopi and Krishnan (2000) employed a projection-based approach to surface reconstruction similar to Boissonnat (1984). This algorithm makes three key assumptions for it to operate correctly. The first is that the points are locally uniform, ensuring that the points in the region of interest are relatively close to one another. The second is that it is able to distinguish different layers of the object. This entails that there is a minimum distance between each side of the object. The third assumption is that the normal deviation between any two triangles on a vertex is less than 90°. For any model this is easily accomplished by increasing the sampling of the object's surface. The algorithm picks a data

point from the point set called the reference point. This reference point is the initial starting point for the triangulation. The technique then finds all the triangles incident on the reference point within a specified distance from the reference point. The algorithm then performs a breadth-first search, iterating through the vertices adjacent to the reference point repeating the process of finding the incident triangles on the adjacent point.

The algorithm accomplishes the reconstruction in three stages: the bucketing stage, the pruning stage and the triangulation stage. The bucketing stage uses the reference point and orthographically projects points in the local area onto a 3D grid. This grid uses the Z axis as the depth and the X and Y axes encodes the position of the point, multiple points that project to the same X and Y location are sorted by the depth value. The next stage of the algorithm is the pruning stage. This is designed to remove points until the remaining points are the incident triangles on the reference point. This stage initially uses the 3D grid as the search space for these points. Following this, the L_2 metric is applied from the reference point to reduce the number of possible triangles created. The L_2 metric rejects points that lie outside a sphere from the reference point. The remaining points are referred to as the candidate points for the reference point. These points are then sorted by angle from the reference point and undergo a visibility test. The visibility ensures that the no mesh boundary lies between the reference point and the candidate point. If a mesh boundary does lie between them, the point is rejected. The remaining points are passed into the next stage, the triangulation stage. The triangulation stage is the final step in generating the incident triangles. This is accomplished by connecting the candidate points around the reference point the in the sorted order of angle.

In addition, this method does not need any additional information such as surface normals.

Other function-fitting algorithms use the marching cube intersection algorithm outlined in Lorensen and Cline (1987) for the final stage in performing the mesh reconstruction. These function-fitting algorithms always use an indicator function that determines if a region is inside or outside of the model. This technique of generating triangles looks at the volumetric properties of the indicator function in a 3D grid. Given the function value for a cube and its neighbours it is possible to determine the triangle that should be generated. Depending on the neighbourhood type, generally a Von Neumann neighbourhood or Moore neighbourhood, there are a maximum number of unique neighbour configurations that specify whether a triangle should be generated. For example, in the 2D Figure 3.3, we demonstrate that the triangle needs to be generated in the square due to specific neighbours being inside the model.

(a) (b)

Figure 3.3: This figure demonstrates a configuration that requires the marching cube intersection method of generating a line in 2D. If this were in 3D, it would generate a triangle. White indicates the square is outside the model and black indicates the square is inside the model. The green line indicates the generated edge in the 2D case.

3.5.2 Mesh Reconstruction Benchmarks

Stanford has created a repository (Curless and Levoy, 1996*b*) of point clouds that are used to evaluate reconstruction algorithms. The Stanford repository contains four models: a bunny, a drill bit, a Buddha statue and a dragon. These models were scanned using a Cyberware 3030MS optical triangulation scanner as a series of range images. They provide the point cloud in multiple sections in their local space as well as a transformation for each section of the model to put them in the same coordinate system.

These models were obtained from the Stanford University Computer Graphics Laboratory. The bunny model, often referred to as the Stanford bunny, is generally considered the standard for testing mesh reconstruction algorithms. For more complex tests, the Buddha and dragon point cloud are used. These point clouds will be used as an initial test to verify that our implemented mesh generation algorithm works correctly the resulting meshes are discussed further in Chapter 7.

3.6 Summary

This chapter considered various aspects of approaches to 3D reconstruction. This included techniques for estimating motion and depth, correcting drift and reconstructing models. Motion estimation techniques used in current 3D reconstruction techniques usually use sensors such as inertial motion units and optical encoders. These provide excellent initial estimates for the motion of a robot or vehicle. However, our system targets hand-held cameras, as these require the least specialized hardware. Due to this, we considered scan-matching approaches and more specifically ICP algorithms. ICP algorithms can have a variety of different subcomponents that can be modified for various user cases.

These subroutines consist of point selection, point association, point rejection, the error function and finally the minimisation technique applied to the error function. ICP algorithms operate by using two sets of points and attempting to find an association between each point in one cloud with a point in the other. Once an association is made, the ICP algorithm attempts to minimise an error function that transforms a point to its associated point in the other point cloud. However, scan-matching approaches are not perfect at estimating motion and cause the estimated trajectory to drift over long sequences. The depth estimation problem has a multitude of established techniques, including software and hardware solutions. Software solutions are based on multi-view geometry. These established techniques typically require two or more viewpoints of the same object with known distances between the different viewpoints, allowing triangulation of the light beams from each camera. For monocular systems, the depth is estimated using a similar approach, except the motion between each frame is estimated. Time-of-flight sensors are the hardware solutions. The most common and easiest to access time-of-flight sensor was developed by Microsoft, known as the Kinect sensor. To reduce the complexity of depth estimation, we make use of the Kinect sensor for obtaining 3D information about the environment.

The most common approach to 3D reconstruction is the use of a SLAM system. We discussed the estimation techniques that use Kalman filters such as the particle filter and the EKF. These techniques work by saving landmarks and maintaining probabilistic information about the location of the landmarks. By incorporating multiple location estimates of landmarks, the system is capable of reducing the uncertainty about the location of camera and the landmarks. The main variants of the Kalman filter, such as the UKF and SEIF, were outlined and their differences briefly explored. This assumes some characteristics of the motion and creates a smoothing of the camera's trajectory. A different solution to SLAM was the graph-based variant that attempts to maximise the goodness of fit model. This builds a graph using the motion estimates, observations and the positions of the camera to find an optimal configuration of the nodes in the graph. Unfortunately, SLAM systems are based on a motion model and generally require odometry information. This requires the system to have a fairly predictable motion, which is not the case with hand-held cameras. However, a motion model for hand-held cameras has been created as described in Davison, Reid, Molton and Stasse (2007). Newer, less common techniques were discussed such as PTAM, DTAM and the Kinect Fusion system. Since PTAM and DTAM focus on using a single RGB camera to perform the reconstruction, these systems were avoided as the depth would only be recoverable to a scale. The Kinect Fusion system was discussed since it was created for use with a depth sensor. This removes the complexity of estimating depth and provides a fast and efficient approach to modelling a 3D environment. This system also provides techniques to remove noise from

the incoming sensor data and minimise drift using the frame-to-model motion estimation approach discussed in Chapter 5.

The final step was to find an approach to rebuild a point cloud into a mesh. Various techniques were described and explored. Most commonly, the techniques were based on Delaunay triangulation. To remove the challenge of numerical stability with Delaunay triangulation, the BPA was chosen. This removed the problem that could be caused if the estimated radius causes a numerical overflow.

The next chapter specifies the approach to the 3D reconstruction problem proposed in this thesis, and how the proposed system will be evaluated.

Chapter 4

Methodology

This chapter focuses on how the aims and objectives of this thesis will be reached. In particular, we describe how we collect data from the environment, estimate the motion of the depth sensor, achieve online motion estimation speeds, reduce noise from the sensor data, fuse successive sensory data sets together, enable generation of large scale environments, extract the implicit and explicit structure of the environment, generate a mesh from the final point cloud, integrate colour data into the mesh and export the meshed point cloud (Section 1.1.1).

The methodology is divided into three sections: the 3D reconstruction process, the mesh generation process and the evaluation of the system. The reconstruction section presents a method of building a 3D point cloud from a camera feed. The mesh generation section focuses on converting the generated point cloud from the reconstruction phase into a mesh. Finally, the evaluation section explains how the results from the reconstruction and mesh generation sections will be analysed and evaluated. The proposed combined system is shown in Figure 4.1.

4.1 Reconstruction

To perform the 3D reconstruction, a Microsoft Kinect sensor will be used to collect sensory information from the environment. The reconstruction will be performed using the techniques described in Newcombe *et al.* (2011*a*). This approach is chosen due to the system's ability to operate at high speeds using GPU programming. This allows the system to create the virtual model at online speeds. This process begins with smoothing the raw pixel data obtained from the depth sensor with a simple bilateral filter as described in Tomasi and Manduchi (1998).

After the bilateral filter has been applied to the raw data, the filtered data can be integrated into a single volume using the volumetric model represen-

tation from Curless and Levoy (1996*a*). The volumetric model representation uses a truncated signed distance function (TSDF) to represent the surface of the model. This allows multiple data sets obtained from the sensor to be fused into a single volume. The technique used to fuse the data sets also provides a method of removing outliers in the filtered sensor's depth data by averaging it. This helps reduce the remaining noise in the filtered sensor data.

The camera motion estimation is implemented using ICP, specifically the method presented in Low (2004). This implementation is chosen because the algorithm employs multiple independent calculations as opposed to the implementation described in Besl and McKay (1992). This allows its computations to run in parallel, enabling a faster reconstruction process.

The ICP algorithm uses the points obtained from a ray casting procedure as input, as discussed in Curless and Levoy (1996*a*). These points are used as the destination points during the motion estimation, while the points generated from the incoming depth image are used as the source points.

The system implemented in Newcombe *et al.* (2011*a*) has an important restriction in that the volume which it is capable of reconstructing is limited. To enable larger scale model reconstruction, a wrap-around indexing system is used. This technique is modelled after Whelan and Kaess (2012) and removes the boundary constraints from Newcombe *et al.* (2011*a*). However, still limits the local reconstruction space to the size of the volume.

To extract the point cloud, orthogonal rays are cast through the volume to detect locations of zero-crossings. These zero-crossing points will form the vertices of the point cloud. To build the point cloud rapidly, most of this system is implemented in Nvidia's CUDA framework for fast GPU computations.

4.2 Mesh Generation

The point cloud extracted from the volume, it must be converted into a mesh. For this, the ball-pivot algorithm by Bernardini *et al.* (1999) is used. This algorithm allows for a natural generation of the mesh by simulating a ball rolling over the surface of the point cloud. Once the mesh has been constructed, it is converted into a common format that allows it to be used in most 3D modelling tools.

4.3 Evaluation

The system evaluation is divided into two sections: the reconstruction phase, which contains the evaluation of the generated point cloud, and the mesh generation phase, which contains the evaluation of the surfaced mesh.

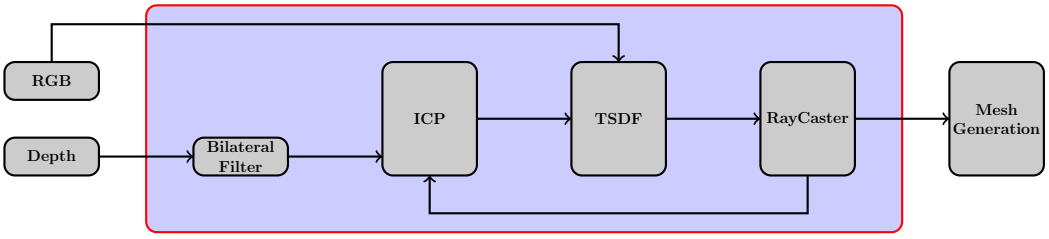


Figure 4.1: An illustration of the proposed system and its different subcomponents. The system takes input in the form of a depth image and an RGB image. The depth image passes through the bilateral filter and propagates to the TSDF component. The RGB image is inserted directly into the TSDF component where the results are fused. The ICP component uses the input from the bilateral filter and the ray caster to compute the best motion estimate for the system. The TSDF component uses the motion estimate from the ICP component to fuse the depth image with the RGB image from the correct position. Following this, the ray caster generates a virtual image to provide the ICP component with a new model view. The ray caster also extracts points from the TSDF volume when points move out of the mapping area.

4.3.1 Reconstruction

The reconstruction algorithm is evaluated based on the reconstruction created from several synthetic data sets (Section 7.1). These synthetic data sets will be obtained using Blender and it's Python interface to generate RGB-D images. Blender is an open-source 3D modelling and rendering tool that allows designers and artists to create photo-realistic scenes (Blender Online Community, 2018). It supports a full 3D pipeline, including modelling, rigging, animation, simulation, rendering, compositing, motion tracking, video editing and game creation. The system will reconstruct the synthetic 3D world which will be evaluated by comparing the shape and position of the point cloud with the ground truth model, artefacts generated and the accuracy of the estimated camera trajectory. When analysing the reconstructed trajectory, the accumulated error over the full frame sequence will be analysed for each axis. This analysis will compare the translation data from the estimated trajectory for each axis against the ground truth translation data. For the rotational data, the error analysis compares the Euler angles and quaternions of the estimated orientation against the ground truth orientation data. Thus, the reconstructed point cloud will be evaluated indirectly by evaluating the accuracy of trajectory of the camera.

4.3.2 Mesh Generation

The mesh is evaluated by performing two types of experiments. The first type of experiment involves taking a selection of meshes, removing (but keeping the vertices) the faces and edges, and then passing them through the mesh generation process. This provides an evaluation of the mesh generation in isolation from earlier subsystems in the combined system. The second set of experiments will be a continuation of the reconstruction experiments: after the point clouds have been generated from the sensor data, they will be passed through the mesh generation component. The meshes in the first type of experiments will be compared to the original synthetic meshes. Results of the second group of experiments are evaluated qualitatively, the meshes will be examined and compared to the expected output. The generated meshes will be analysed by inspecting their faces and their known topologies.

4.4 Summary

In this chapter a proposed approach to generating 3D point clouds using the Kinect Fusion system was proposed with a few modifications. In addition, a mesh reconstruction algorithm was identified that can convert the point clouds generated from the proposed system to meshes. The evaluation of the results from the modified Kinect Fusion system and the mesh generation component are discussed.

The next chapter focuses on the implementation of the above 3D reconstruction techniques that are used to build point clouds.

Chapter 5

Kinect Fusion

This chapter describes a modified implementation of the Kinect Fusion system by Newcombe *et al.* (2011*a*). This implementation is used to produce a 3D point cloud.

5.1 System Overview

The original Kinect Fusion system implemented by Newcombe *et al.* (2011*a*) is outlined in Figure 5.1. This system is extended by (a) integrating the RGB image into the 3D volume, and (b) implementing the moving volume concept as described in Whelan and Kaess (2012) (Section 5.5.2). The Kinect Fusion system has four components: the bilateral filter component, ICP component, truncated signed distance function (TSDF) component and the ray caster component. The images obtained from the Kinect are RGB-D images which are susceptible to errors in the depth measurements. Therefore, the depth images from the Kinect are passed through a bilateral filter to filter out noise by generating an average reading for each pixel using similar pixel values in its local area. The resulting filtered images are then passed through to the ICP and TSDF components in turn. This system uses a coarse-to-fine approach to estimate the relative motion between successive frames. In order to map the RGB data to the model a 3D volume is used to store the data. At each position in this volume an estimate for the RGB colour is integrated whenever a new depth and RGB image is integrated. This enables the system to produce a point cloud that has associated colour data. The moving volume concept allows the TSDF volume to virtually translate in the environment, allowing large scale mapping. The TSDF volume stores the estimated surface until it is required to move the volume. The ray caster then extracts the point cloud from that region and resets it.

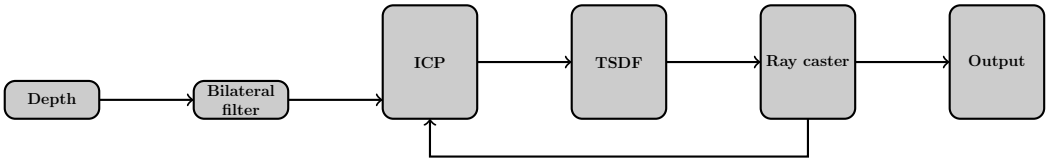


Figure 5.1: An outline of the original Kinect Fusion system.

5.2 Pyramid Construction

An image pyramid is a set of images derived from a single image such that at each level the image resolution is halved from the previous level. Starting from the original image at level 0, the image at level $i+1$ is a sub-sampled version of the image at level i, such that the image at level $i+1$ has half the resolution of the image at level i. This process continues until some predetermined number of levels has been generated. Figure 5.2 illustrates the sub-sampled images of the pyramid. A common operation on an image pyramid is applying a filter to it: in this case a bilateral filter will be applied to each image in the pyramid, as discussed further in the next section. The image pyramid is generated for each incoming depth image from the Kinect sensor. This modifies the Kinect camera parameters for each level l as follows:

$$K^{-l} = \begin{bmatrix} f_x^{-l} & 0 & o_x^{-l} \\ 0 & f_y^{-l} & o_y^{-l} \\ 0 & 0 & 1 \end{bmatrix}.$$

5.3 Bilateral Filter

Bilateral filtering is a technique used in image processing to filter out noise and smooth an image while preserving edges (Tomasi and Manduchi, 1998). This technique works by combining the concepts of range and domain filtering.

When applying the bilateral filter to an image, the domain is the spatial location of a pixel, as shown in Equation 5.1, and the image information for that pixel location are the range as shown in Equation 5.2.

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{5.1}$$

$$y = f(\mathbf{u}) \tag{5.2}$$

Two filters are applied: a closeness filter and a similarity filter. The closeness filter, c, is a simple filter that uses the geometric distance between the reference point $\mathbf{u}$ and a nearby point ξ. This filter thus operates on the domain.

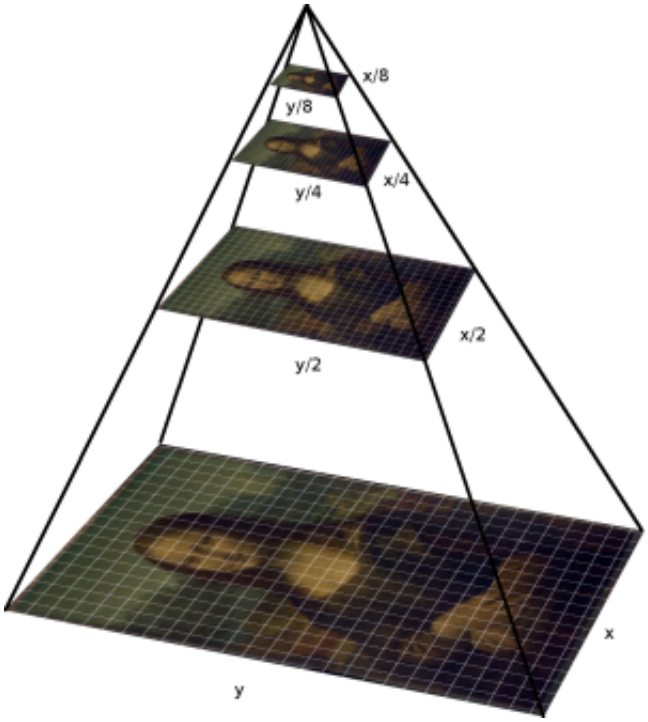


Figure 5.2: The image pyramid with the original image at level 0 (bottom), and the coarsest sub-sampled image (with the lowest resolution) at level 4 (IIPImage, 2018).

Domain filtering attempts to filter a digital signal based on the input values of some function. This modifies the input values by applying a filter function to each of the values. For example given a function $y = g(x)$, the filtering function d is applied to the x values. The new y value is computed as $y = g(d(x))$. More formally, a convolutional filter c can be applied to the image f centred at $\mathbf{u}$, the closeness function c operates in the domain of f as shown in Equation 5.3:

$$h_c(\mathbf{u}) = k_d^{-1}(\mathbf{u}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi)\, c(\xi, \mathbf{u})\, d\xi, \tag{5.3}$$

where

$$k_d(\mathbf{u}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{u})\, d\xi \tag{5.4}$$

is a normalizing factor.

The similarity filter, s, measures the photometric similarity between the pixels. This operates on the range. Range filtering takes the function output values and simply filters the values according to the filter function. For example given a function $y = g(x)$, with the filtering function r, the filter is applied as

$y = r(g(x))$. More formally the range of an image f can be filtered by a range filter s as shown in Equation 5.5

$$h_s(\mathbf{u}) = k_r^{-1}(\mathbf{u}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi)\, s(f(\xi), f(\mathbf{u}))\, d\xi, \tag{5.5}$$

where

$$k_r(\mathbf{u}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(f(\xi), f(\mathbf{u}))\, d\xi \tag{5.6}$$

is a normalizing factor.

The filters c and s are combined to obtain

$$h_{s,c}(\mathbf{u}) = k^{-1}(\mathbf{u}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\xi)\, c(\xi, \mathbf{u})\, s(f(\xi), f(\mathbf{u}))\, d\xi, \tag{5.7}$$

with normalizing factor

$$k(\mathbf{u}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{u})\, s(f(\xi), f(\mathbf{u}))\, d\xi. \tag{5.8}$$

This bilateral filter replaces pixel values by an average of nearby similar pixels, effectively averaging out the noisy values. Using this filter gives a larger influence to closer pixels with similar values than values which are further away and dissimilar.

The filters can be applied as simple difference measurements as shown in Equations 5.9 and 5.10. However, Gaussian-based closeness and similarity functions given in Equations 5.11 and 5.12 are used. This preserves the structure of the image, while removing noise.

$$d(\xi, \mathbf{u}) = (\mathbf{u} - \xi)^2 \tag{5.9}$$

$$r(\xi, \mathbf{u}) = f(\mathbf{u}) - f(\xi) \tag{5.10}$$

$$c(\xi, \mathbf{u}) = e^{-\frac{1}{2}\left(\frac{d(\xi, \mathbf{u})}{\sigma_d^2}\right)} \tag{5.11}$$

$$s(\xi, \mathbf{u}) = e^{-\frac{1}{2}\left(\frac{r(\xi, \mathbf{u})}{\sigma_r^2}\right)} \tag{5.12}$$

The bilateral filter is applied to the incoming raw depth image, R, with $\sigma_r = 4.5$ and $\sigma_d = 30$ for all levels in the image pyramid in the main system. These parameters were determined to be sufficient for the system by manually tuning the parameters until the desired results were obtained. To improve the system's performance, the filters are applied using a 13x13 mask. These parameters provide adequate filtering to smooth the depth image. This is applied to the image pyramid resulting in a new image D at each level.

A new image pyramid is generate with the bilateral filter, the resulting image pyramid is fed into the next component: the iterative closest point component.

5.4 Iterative Closest Point

In the original Kinect Fusion system, ICP is used to compute the relative motion between successive RGB-D frames. This motion estimation step is based on the papers by Low (2004) and Chen and Medioni (1992), and uses the point-to-plane error metric. The point-to-plane error metric is used to compute the distance error between each source point and the tangent plane at the corresponding destination point. The ICP algorithm is solved iteratively for each level during a single time step (motion estimate between frames). The ICP algorithm estimates the motion between two frames by performing the projective data association, computing the error and solving the non-linear system. This is a single iteration of the ICP algorithm but due to the non-linearity of the system, this needs to be solved iteratively. The ICP algorithm iteratively refines the estimate by computing a solution several times on a single level. After several iterations on a level, it moves to the next level using the previous levels estimate. At the end of all the levels the final estimate between the two frames is obtained.

In this section, the symbols s_j and d_j are reused to indicate the source and destination points respectively in the local coordinate system, with j a specific index in the collection of source or destination points. Similarly, s_j^g and d_j^g are used to indicate the source and destination points in the global (or world) coordinate system, and n_j^d is used for the unit normal tangent to the plane surface at d_j, the methods used to obtained these points are detailed further in this section. The points and normals in the local coordinate system can be transformed to the global coordinate system as shown in Equation 5.13 with an associated transformation matrix T and the rotation matrix R extracted from the transformation matrix T.

$$s_j^g = T\, s_j \qquad\qquad n_j^{s,g} = R\, n_j^s \tag{5.13}$$

A vertex and a normal map is constructed for each image in the pyramid, denoted by V_l^s and N_l^s for level l. Given a depth image, D, from the bilateral filter, a vertex $V(\mathbf{u})$ is generated for each pixel $\mathbf{u}$ using the techniques discussed in Section 2.1.2. The vertices generated are stored in a vertex map V_l^s.

$$V(\mathbf{u}) = \begin{bmatrix} D(\mathbf{u})\frac{(u_x - o_x)}{fx} \\ D(\mathbf{u})\frac{(u_y - o_y)}{fy} \\ D(\mathbf{u}) \end{bmatrix} \tag{5.14}$$

From the vertex map V, a unit normal map is generated. Since the vertex map is generated from an image, the surface normal can be generated using the neighbours of the pixel point $\mathbf{u}$ as shown in Equation 5.15.

$$N(\mathbf{u}) = (V(u_x + 1, u_y) - V(u_x, u_y)) \times (V(u_x, u_y + 1) - V(u_x, u_y)) \tag{5.15}$$

This approximates the surface normal vector by computing two direction vectors, from the current vertex to two of its neighbours, and then computing the cross product which results in a vector orthogonal to them. This vector is then normalised to unit length and stored in the normal map.

The vertex and normal maps constructed from the bilateral filtered pyramid are known as the source points. These vertex and normal maps are 2D arrays that store the generated vertex positions and surface normals from the depth image. These 2D arrays are the size of the resolution of each image at the level of the pyramid. For each frame, the ICP algorithm is applied to the input from the bilateral filter as well as from the ray caster. The output from the ray caster is the same as the output of the bilateral filter: a pyramid of vertex and normal maps. The ray caster's vertex and normal maps are called the destination points. These are denoted by $V_l^{d,g}$ and $N_l^{d,g}$, d is used to indicate that the vertex and normal map contain the destination points. Obtaining the ray caster pyramid will be discussed in a Section 5.6.

The ICP algorithm begins by creating a set of point associations from the source to destination points. The output of the ICP algorithm is a 3D rigid body transformation that approximately transforms the source points to the destination points. The ICP algorithm accomplishes this by minimising an error metric. This implementation uses, projective data association and the point-to-plane error metric.

5.4.1 Projective Data Association

Projective data association works by projecting a point from one coordinate system onto another. To transform points between coordinate systems the appropriate transformation matrix is used. Initially, the transformation matrix T_0 represents the starting position and orientation of our camera at time 0. The idea is to generate a 3D point for each pixel using the initial camera transformation T_0 and back-project them onto the image plane of the camera at T_1. This is illustrated in Figure 5.3. Once the 3D points have been back-projected, two pixel coordinates for a 3D point are obtained. $\mathbf{u}$ refers to the pixel location from which the destination point was generated, while the back-projected pixel $\hat{\mathbf{u}}$ is the source pixel. The pixel $\mathbf{u}$ is now associated with the pixel $\hat{\mathbf{u}}$.

5.4.2 Error Function

The point-to-plane error metric used is that described in Low (2004). Figure 5.4 helps illustrate the point-to-plane error between two surfaces. This minimises the sum of the squared distances between each transformed source point and the plane tangent at its associated destination point. More specifically given source points $s_j \in V^s$ and associated destination points $d_j^g \in V^{d,g}$

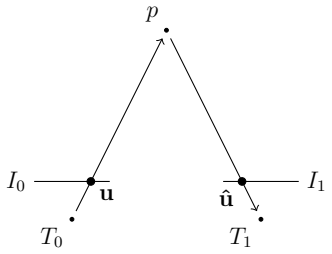


Figure 5.3: This illustrates how a point located at position $\mathbf{u}$ on the image plane I_0 with the associated camera transformation T_0 is back projected onto the image plane, I_1, of a camera located at position T_1. This produces the associated position $\hat{\mathbf{u}}$ on image plane I_1.

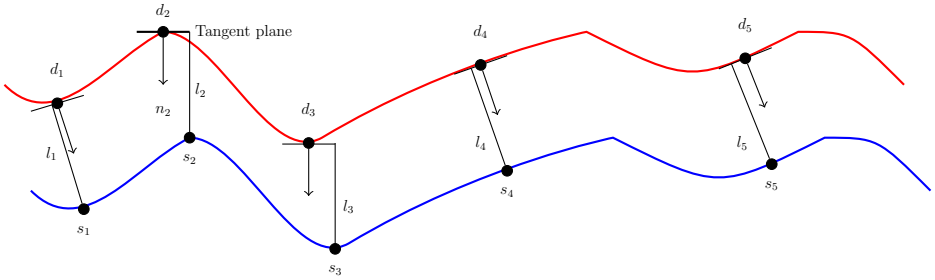


Figure 5.4: Point-to-plane error metric between two surfaces. The destination point with the normal and the associated source point.

and their unit normals, $n_j^{d,g} \in N^{d,g}$, the objective is to find a 3D rigid body transformation that minimises

$$\sum_{\forall j} \left(\left(T s_j - d_j^g \right) \cdot n_j^{d,g} \right)^2 . \tag{5.16}$$

5.4.3 Solving the Non-Linear System

This optimisation problem is solved using a least squares approach. The transformation matrix is represented as a state vector $\mathbf{x}$ with six parameters:

$$\mathbf{x} = \begin{bmatrix} \theta \\ \phi \\ \alpha \\ x \\ y \\ z \end{bmatrix} \tag{5.17}$$

The first three parameters correspond to the rotation around each of the axes (θ for the x axis, ϕ for the y axis and α for the z axis). Due to the non-linearity of

the trigonometric functions in the rotation matrix it has to be solved as a non-linear least squares system. Since the parameters θ, ϕ and α are inputs to non-linear trigonometric functions in the rotation matrix, the rotation matrix needs to be linearised. The transformation matrix T is factorized into the product of a transformation matrix describing the rotation, T_R, and a transformation matrix describing the translation T_T as follows

$$T_R = \begin{bmatrix} \cos\alpha\cos\phi & -\sin\alpha\cos\theta + \cos\alpha\sin\phi\sin\theta & \sin\alpha\sin\theta + \cos\alpha\sin\phi\cos\alpha & 0 \\ \sin\alpha\cos\phi & \cos\alpha\cos\theta + \sin\alpha\sin\phi\sin\theta & -\cos\alpha\sin\theta + \sin\alpha\sin\phi\cos\theta & 0 \\ -\sin\phi & \cos\phi\sin\theta & \cos\phi\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{and } T_T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The suggestion in Rusinkiewicz and Levoy (2001) is used: motion between two successive images is assumed to be relatively small and thus rotation angles between the two frames are small. In this regime, the non-linear trigonometric functions are approximated by using the approximations

$$\sin\gamma = \gamma \text{ and } \cos\gamma = 1.$$

This results in the linearised rotation transformation

$$T_R \approx \begin{bmatrix} 1 & \theta\phi - \alpha & \theta\alpha + \phi & 0 \\ \alpha & \theta\phi\alpha + 1 & \phi\alpha - \theta & 0 \\ -\phi & \theta & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The approximation of T_R uses the estimate of the product of small angles tends to 0 and leads us to $\hat{T}_R$

$$T_R \approx \begin{bmatrix} 1 & -\alpha & \phi & 0 \\ \alpha & 1 & -\theta & 0 \\ -\phi & \theta & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \hat{T}_R.$$

The linearised rotation transformation and translation transformation are combined to create $\hat{T}$ as follows

$$\hat{T} = T_T\hat{T}_R.$$

The state vector in Equation 5.17 is used to build the transformation matrix $\hat{T}$. Now that the rotation matrix has been linearised, Equation 5.16 is rewritten

using $\hat{T}$ in Equation 5.18. In this case the Cholesky decomposition is used to compute the solution.

$$T_{opt} = \arg\min_{T} \sum_{\forall j} \left(\left(\hat{T} s_j - d_j^g \right) \cdot n_j^{d,g} \right)^2 \tag{5.18}$$

Further detail about how this system is solved is described below.

5.4.4 Iterative Closest Point Algorithm

The ICP algorithm requires the vertex and normal maps that were generated from the pyramid construction, V_l^s and N_l^s, and also those from the ray caster (discussed in Section 5.6), $V_l^{d,g}$ and $N_l^{d,g}$. In this section T_i describes the motion estimate of the camera at frame i, while $T_{i,k}$ describes the motion estimate at frame i after the k^{th} iteration of solving the non-linear system. The index k represents the iteration of the ICP algorithm when solving Equation 5.18.

At time frame i, the current estimate of the camera's transformation matrix, T_i, is referred to as the predicted transformation and the camera's previously estimated transformation at time $i-1$, T_{i-1}, is referred to as the current transformation. Initially, the predicted transformation matrix is set to the current transformation matrix

$$T_{i,0} = T_{i-1}. \tag{5.19}$$

A single iteration of the ICP algorithm is detailed and starts at the highest level of the vertex map pyramids (the vertex maps with the lowest resolution). For each pixel location $\mathbf{u}$ in the ray caster vertex map, the corresponding global destination vertex $V_l^{d,g}(\mathbf{u})$, is transformed to the camera space $V_l^d(\mathbf{u})$. This is then transformed to homogeneous coordinates in the image space, producing an associated pixel $\hat{\mathbf{u}}$, using T_{i-1}^{-1} and the camera matrix K as described in Section 2.3. The image coordinates $\mathbf{u}$ and $\hat{\mathbf{u}}$ are associated points from the ray caster vertex map to the bilateral filtered vertex map. Using the associated vertices and normals, $V_l^s(\hat{\mathbf{u}})$, $N_l^s(\hat{\mathbf{u}})$, $V_l^{d,g}(\mathbf{u})$ and $N_l^{d,g}(\mathbf{u})$, the techniques used in the previous section are used to compute the new motion estimate and can be seen in Algorithm 1. The algorithm is applied to each level of the pyramid iteratively, estimating the predicted transformation matrix at each level. Once the association is made between $\mathbf{u}$ and $\hat{\mathbf{u}}$, the vertex and normal from the source points, $V_l^s(\hat{\mathbf{u}})$ and $N_l^s(\hat{\mathbf{u}})$, are converted to global space, $V_l^{s,g}(\hat{\mathbf{u}})$ and $N_l^{s,g}(\hat{\mathbf{u}})$, seen in line 16 and 17 of Algorithm 1. Before $\mathbf{u}$ and $\hat{\mathbf{u}}$ are truly declared as associated points, the vertices, $V_l^{s,g}(\hat{\mathbf{u}})$ and $V_l^{d,g}(\mathbf{u})$, and normals, $N_l^{s,g}(\hat{\mathbf{u}})$ and $N_l^{d,g}(\mathbf{u})$, are tested to ensure they are within a minimum distance from each other and the surface normals are in a similar direction.

The `DistanceThreshold` and `NormalThreshold` are the parameters used to ensure that the vertices and normal vectors associated with $\mathbf{u}$ and $\hat{\mathbf{u}}$ are

similar enough to be used to estimate the next iteration of the predicted transformation. If $\mathbf{u}$ and $\hat{\mathbf{u}}$ are similar enough, they are added as an equation to the linear system as illustrated in Algorithm 2. These equations are generated to minimise Equation 5.18. In this implementation, the `DistanceThreshold` is set to 0.025 meters and the `NormalThreshold` is set to 0.65. This is the result of the dot product between the two unit length vectors. These parameters were varied to test the limits of this system. It was empirically determined that the `DistanceThreshold` could be set as high as 0.05 meters before the camera tracking started to fail and the `NormalThreshold` could be set as low as 0.55 before the camera tracking became unreliable. The non-linear system is solved using the Cholesky decomposition to compute the incremental state estimation vector in Equation 5.17. Since Equation 5.18 aims to solve a non-linear problem, the new state vector needs to be applied to the predicted transformation, refer to Section 2.3.2. The update of the predicted transformation is done by building a transformation matrix T_δ from the newly estimated state vector $\mathbf{x}$. The update is computed as

$$T_{i,k+1} = T_\delta T_{i,k}. \tag{5.20}$$

In this implementation a coarse-to-fine approach using a pyramid of 3 levels applying ten iterations to level 2, five iterations to level 1 and four iterations to level 0 is used. With the new pose estimate calculated from the ICP component, a truncated signed distance function is used to build a consistent 3D model from the streaming depth images.

5.5 Truncated Signed Distance Function

To create a 3D model, the volumetric method described in Curless and Levoy (1996*a*) is used. Since the ICP algorithm provides the camera pose for each depth image, the position of the camera in global space is assumed to be known. This allows us to integrate each depth image into a global 3D space. This technique uses a 3D grid to represent the global 3D space that the depth images will be integrated into. Each element of the 3D grid is called a voxel and it represents a unit of a volume. This 3D grid, known as the truncated signed distance function (TSDF) volume, is made of equally sized voxels per dimension in this implementation. The underlying voxel size is limited to the memory of the GPU that is used. The 3D grid is defined by three parameters the grid size, $\mathbf{gs}$, the voxel cell size, $\mathbf{vcs}$, and the voxel centre, $\mathbf{vc}$.

$$\mathbf{gs} = \begin{bmatrix} \mathbf{gs}_x \\ \mathbf{gs}_y \\ \mathbf{gs}_z \end{bmatrix} \qquad \mathbf{vcs} = \begin{bmatrix} \mathbf{vcs}_x \\ \mathbf{vcs}_y \\ \mathbf{vcs}_z \end{bmatrix} \qquad \mathbf{vc} = \begin{bmatrix} \mathbf{vc}_x \\ \mathbf{vc}_y \\ \mathbf{vc}_z \end{bmatrix} \tag{5.21}$$

Data: K, V^s, N^s, $V^{d,g}$, $N^{d,g}$, T_{i-1}, $level_{max}$, iterations
Result: T_i

$k = 0$;
$T_{i,k} = T_{i-1}$;
for $i \leftarrow 0$ **to** $level_{max}$ **do**
  for $it \rightarrow iterations$ **do**
    A = empty matrix (N x 6);
    b = empty vector (N);
    for $\mathbf{u} \leftarrow [0,0]$ **to** $\mathbf{u}_{max}$ **do**
      DestinationGlobal $\leftarrow V_l^{d,g}(\mathbf{u})$;
      DestinationLocal $\leftarrow T_{i-1}^{-1}$ DestinationGlobal;
      if DestinationLocal$_z$ $<$ *0* **then**
        continue;
      end
      $\hat{\mathbf{u}} \leftarrow K$ DestinationLocal;
      $\hat{\mathbf{u}} \leftarrow \hat{\mathbf{u}}/\hat{u}_z$;
      SourceGlobal $\leftarrow T_{i,k}\ V_l^s(\hat{\mathbf{u}})$;
      $ng \leftarrow T_{i,j}[:3,:3]N^s(\hat{\mathbf{u}})$;
      distance $\leftarrow$ |SourceGlobal $-$ DestinationGlobal|;
      normal $\leftarrow N^{d,g}(\hat{\mathbf{u}}) \cdot ng$;
      if distance $<$ DistanceThreshold ***and***
      normal $>$ NormalThreshold **then**
        `AddEquation(`A, b,DestinationGlobal,SourceGlobal, $N_l^{d,g}(\mathbf{u})$`)`;
      end
    end
    $\mathbf{x} \leftarrow$ `SolveCholesky(`A, b`)`;
    $T_{incremental} \leftarrow$ `StateVectorToTransformationMatrix(`$\mathbf{x}$`)`;
    $T_{i,k+1} = T_{incremental}T_{i,k}$;
    $k = k + 1$;
  end
end
return $T_{i,k}$

Algorithm 1: Kinect Fusion ICP algorithm for all levels of the vertex and normal pyramid.

```
1 Function AddEquation(A, b, vmg, vg, N_l^{d,g}(u)):
2     a = vg × N_l^{d,g}(u);
3     row = [a_x, a_y, a_z, N_l^{d,g}(u)_x, N_l^{d,g}(u)_y, N_l^{d,g}(u)_z];
4     A.append(row);
5     b.append(N_l^{d,g}(u) · (vmg − vg))
6     return;
7 end
```

Algorithm 2: This is the pseudocode for the `AddEquation` function. It augments the matrix A and vector b by appending an additional row onto each.

Two variables are ultimately stored per voxel, a TSDF value and a weight. The TSDF value is a truncated version of the estimated signed distance from the voxel to the closest surface (we discuss truncation of this value in Section 5.5.1). Since the camera is moving, the average of the TSDF values for each camera position is stored in the voxel, the distance is calculated along the line of sight of the camera. The distance d reported by the raw depth image, R, is truncated such that the value lies within an arbitrary small value $\pm\mu$ of the 3D point.

Since the Kinect sensor is being used, there are certain assumptions made when using the volumetric representation described here. The information gathered from each pixel in the depth image occurs on the corresponding line of sight of the sensor. Given an arbitrary distance, d_a, and the distance along the line of sight of the camera corresponding to the pixel $\mathbf{u}$, $(K^{-1}\mathbf{u})R(\mathbf{u})$, the points corresponding to $d_a < (K^{-1}\mathbf{u})R(\mathbf{u}) - \mu$ are free space. Similarly, points with $d_a > (K^{-1}\mathbf{u})R(\mathbf{u}) + \mu$ are behind the front of the surface model from the perspective of the sensor and the sensor provides no information beyond this point, as shown in Figure 5.5. The actual surface is considered to lie somewhere between $(K^{-1}\mathbf{u})R(\mathbf{u}) - \mu$ and $(K^{-1}\mathbf{u})R(\mathbf{u}) + \mu$. To represent this uncertainty, the signed distance function is used.

5.5.1 Signed Distance Function

The signed distance function (SDF) is a function $G(\mathbf{x})$ represented by multiple samples. These samples are the weighted signed distance to a voxel position $\mathbf{x}$ for each depth measurement. The signed distance, $g(\mathbf{x})$, is defined as follows: given a voxel v, it is converted to world space to produce a 3D point p with v located along the line of sight of the camera and its associated pixel location $\mathbf{u}$. The surface location $R(\mathbf{u})$ is converted to world space using the predicted transformation, T_{i+1}, and the camera matrix K as

$$T_{i+1}K^{-1}\mathbf{u}R(\mathbf{u}).$$

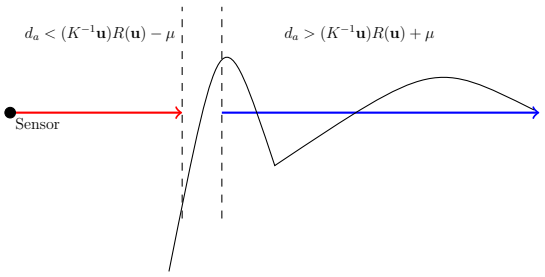


Figure 5.5: This figure illustrates the ground truth (black line) and the sensors line of sight from the depth camera. The red line illustrates the free space in front of the surface while the blue line illustrates the region where no depth information is available. The gap between the two lines indicate the uncertainty margin of size 2μ.

The signed distance for the frame is then computed as follows

$$g(\mathbf{x}) = |p - T_{i+1}K^{-1}\mathbf{u}R(\mathbf{u})|.$$

The SDF is constructed by combining a series of signed distances and weights $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots g_n(\mathbf{x})$ and $w_1(\mathbf{x}), w_2(\mathbf{x}), \ldots w_n(\mathbf{x})$ from a series of range images. These functions are joined using a weighted sum given in Equations 5.22 and 5.23 to produce the SDF function value $G(\mathbf{x})$ and the weight function $W(\mathbf{x})$.

$$W(\mathbf{x}) = \sum_{i=1}^{n} w_i(\mathbf{x}) \tag{5.22}$$

$$G(\mathbf{x}) = \frac{\sum_{i=1}^{n} w_i(\mathbf{x})g_i(\mathbf{x})}{W(\mathbf{x})} \tag{5.23}$$

The results of $G(\mathbf{x})$ and $W(\mathbf{x})$ are stored in the TSDF volume at each voxel. The functions $G(\mathbf{x})$ and $W(\mathbf{x})$ are represented in the TSDF volume with 16 bits per function value, i.e. they are limited in size. The function $G(\mathbf{x})$ is truncated resulting in a value between g_{min} and g_{max} as shown in Figure 5.6. The function $G(\mathbf{x})$ is then referred to as the TSDF. Figure 5.7 shows the result of combining two range image surfaces in the TSDF volume. The true position of the surface is located where a zero-crossing is found. This zero-crossing is found by examining the values stored in the voxels. After integrating multiple range surfaces into the volume, the voxels where the values change from positive to negative along the line of sight of the camera are called zero-crossings. A zero-crossing is the location of a estimated surface.

The weighting functions allow us to give more influence to SDF values that are more certain rather than the newer SDF values that are being integrated into the volume. This leads to a more consistent view of the surface. The weighting function is typically dependent on the sensor used. In many cases

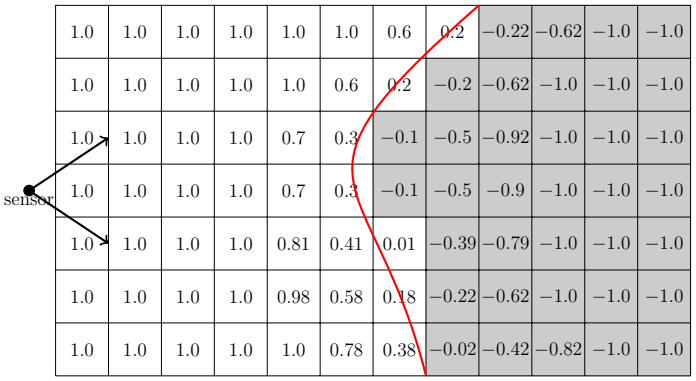


Figure 5.6: Truncating the depth values around the surface. The truncation values used here are $g_{min} = -1$ and $g_{max} = 1$. The values of 1 should be larger for the cells that are further away from the red surface and the values of -1 should be smaller in the cells that are further from the red surface. However, they were truncated to these values.

there is a reduction in the certainty of the measurement as the distance to the surface increases.

Since this system is designed to be able to be run online and incrementally take input from streaming RGB-D data, Equations 5.23 and 5.22 are written using incremental updates in Equation 5.24 and 5.25.

$$G_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x})G_i(\mathbf{x}) + w_{i+1}(\mathbf{x})g_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})} \tag{5.24}$$

$$W_{i+1}(\mathbf{x}) = W_i(\mathbf{x}) + w_{i+1}(\mathbf{x}) \tag{5.25}$$

However, for a simple average, the weighting function can be set to a constant, $w_{i+1}(\mathbf{x}) = 1$. This allows the system to treat each input measurement equally, making newer measurements weigh the same as older measurements. In this implementation the simple average is used.

The updated region of the TSDF volume is limited to the region around the surface estimate. This is done to prevent interference between the visible surface and any other surfaces that may be present behind the surface. This is dictated by the values g_{min} and g_{max}. Figure 5.8 demonstrates a situation where there may be surface interference. Assume the camera integrates two range images from two different locations from two different sides of a surface. Updating all the voxels in the TSDF volume will override some surface measurements. In the system, only voxels in the TSDF volume that are within 0.1 meters of the surface measurement will be updated.

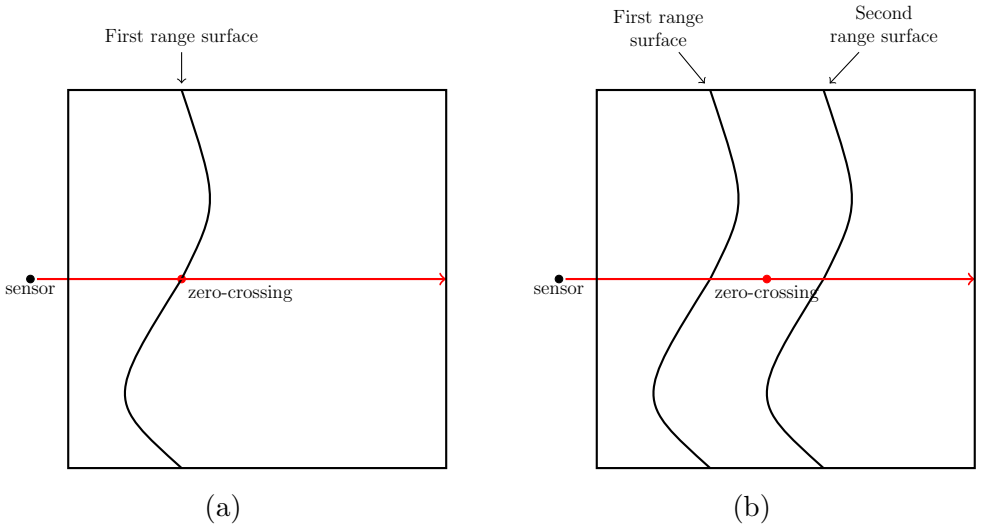


Figure 5.7: Subfigure (a) shows the zero-crossing of a single integrated range surface into the TSDF volume along the line of sight of the sensor shown in red. Subfigure (b) shows the new estimated zero-crossing when a second range surface is integrated into the TSDF volume. In Subfigure (b) the distance between the range surfaces is exaggerated for visual clarity.

5.5.2 Implementation and Moving Volume

The TSDF volume is initialised using the grid size (**gs**), voxel centre (**vc**) and voxel cell size (**vcs**). An additional parameter d, the distance the volume represents along each axis is added. The volume size is set to $512 \times 512 \times 512$ voxels. This is due to the memory currently available on common graphics cards. Using a volume of $512 \times 512 \times 512$ 32-bit values would require about 536 megabytes of memory, while extending this volume to $1024 \times 1024 \times 1024$ voxels requires about 4.3 gigabytes.[1] A second volume is also used that is responsible for storing the colour data for each voxel from the RGB images. The colour volume data uses 32-bit values per voxel: 8-bits are used for each of the red, green and blue channels. Additionally, there are 8-bits remaining for an alpha value which is not used in this implementation. The update of the colour value per voxel is performed by averaging the each colour channel over all the input RGB images.

Given a distance that the volume needs to represent, d, the volume is subdivided such that each voxel represents a uniform volume by setting

$$\mathbf{vcs} = \frac{d}{\mathbf{gs}}.$$

Assuming the world origin lies at voxel (0, 0, 0), world space coordinates for

[1] At the time of writing, most common graphics cards are limited to 4 gigabytes.

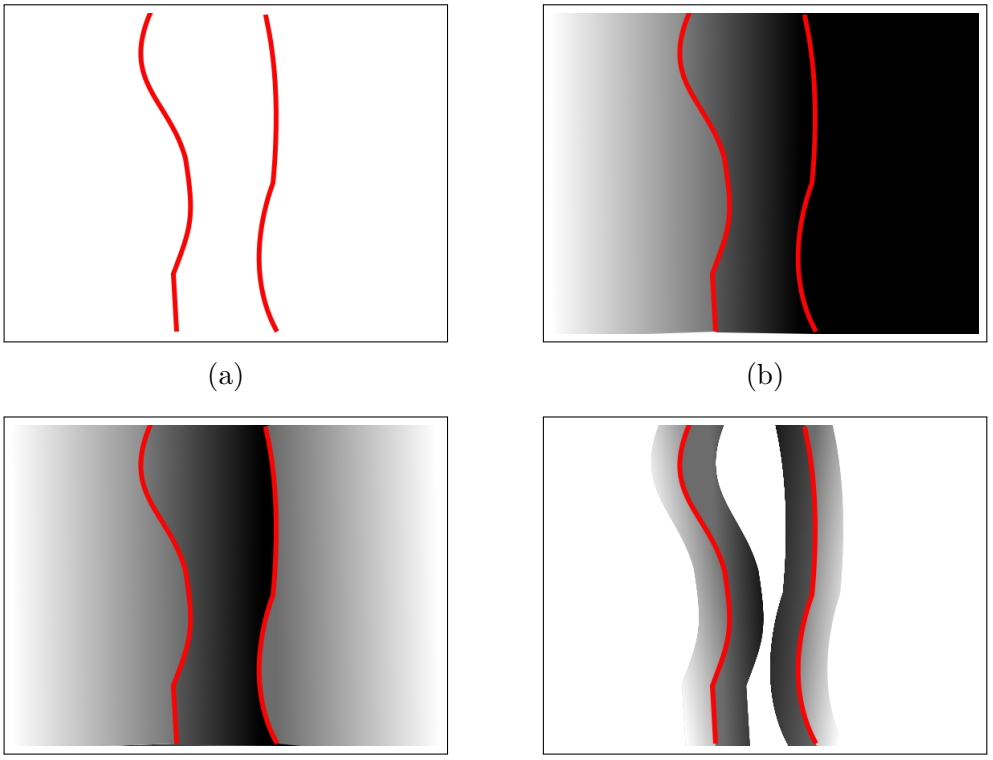


Figure 5.8: This demonstration of the differences between limiting the TSDF update to a region with a threshold from the surface. Subfigure (a) shows two surfaces in the real world. Subfigure (b) shows the TSDF values when the first image is integrated into the TSDF volume without limiting the TSDF update area. Subfigure (c) shows the TSDF values when the second range image is integrated from the rear. Subfigure (d) shows the TSDF values when they are limited to a region around the surface.

voxels can be computed by performing element-wise multiplication between the index coordinates, $\mathbf{x}_{index}$, and the voxel cell size, **vcs**, as shown in Equation 5.27.

$$\mathbf{x}_{index} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5.26}$$

$$\mathbf{x}_{world} = \mathbf{x}_{index}\mathbf{vcs} \tag{5.27}$$

To access a TSDF or weight value from the volume, x, y and z coordinates are required. However, the 3D grid is implemented using a single-dimensional

array representation. The 3D coordinates (x, y, z) are mapped to 1D space using

$$i = x + y(\mathbf{gs}_x) + z(\mathbf{gs}_x\mathbf{gs}_y). \tag{5.28}$$

At this point, the volume can only be accessed for all locations (between 0 and 511) for each dimension. With the implementation as described above, the system will be limited by the distance d.

The above follows the original paper by Newcombe *et al.* (2011*a*): the TSDF is limited in the volume that can be reconstructed. To address this limitation, the technique described in Whelan and Kaess (2012) that allows the volume to 'move' is implemented. This allows the system to fuse depth images without being limited to a fixed location. However, this only allows fusing new depth information inside the current volume location. Instead of moving data with in the TSDF volume, a wraparound buffer is implemented which allows the volume to move efficiently in any direction. This is done using the parameter **vc**, the voxel centre, as demonstrated in Figure 5.9.

To allow the system to use the wraparound buffer, a simple modification to the indexing in Equation 5.28 allows the system to virtually translate the voxels without performing any copy or move operations on the values stored in the voxels. This is done by modifying the indices by adding the voxel centre to each index and taking the modulus with respect to the grid size, **gs**.

The process virtually translates the TSDF volume, once the camera has been moved further than a specified threshold from voxel centre. In this system, the number of voxels, b, the camera needs to move in any direction before the TSDF volume must be shifted is specified. Once the system detects that the camera has passed this threshold, the system extracts and stores the zero-crossings (discussed in Section 5.6) from the region that will no longer be visible in the shifted TSDF volume. The TSDF values and weights in those voxels are then reset.

The camera's position is represented by the transformation matrix T_{i+1} obtained from the ICP algorithm. This is used to determine how far the camera has moved from the voxel centre, **vc**. Once the camera has moved b voxels along any axis, the voxel centre, **vc**, is updated. The translation vector t_{i+1} obtained from the transformation matrix T_{i+1} is used to compute the voxel coordinates. More specifically, the number of voxels moved are added to **vc** as per Equation 5.29. Figure 5.10 shows the update of the threshold boundary and the world space boundary in 2D.

$$\mathbf{vc} = \left\lfloor \frac{t_{i+1}}{\mathbf{vcs}} \right\rfloor \tag{5.29}$$

The calculation to determine the new voxel centre is performed by converting the world coordinates of the new camera location to voxel coordinates.

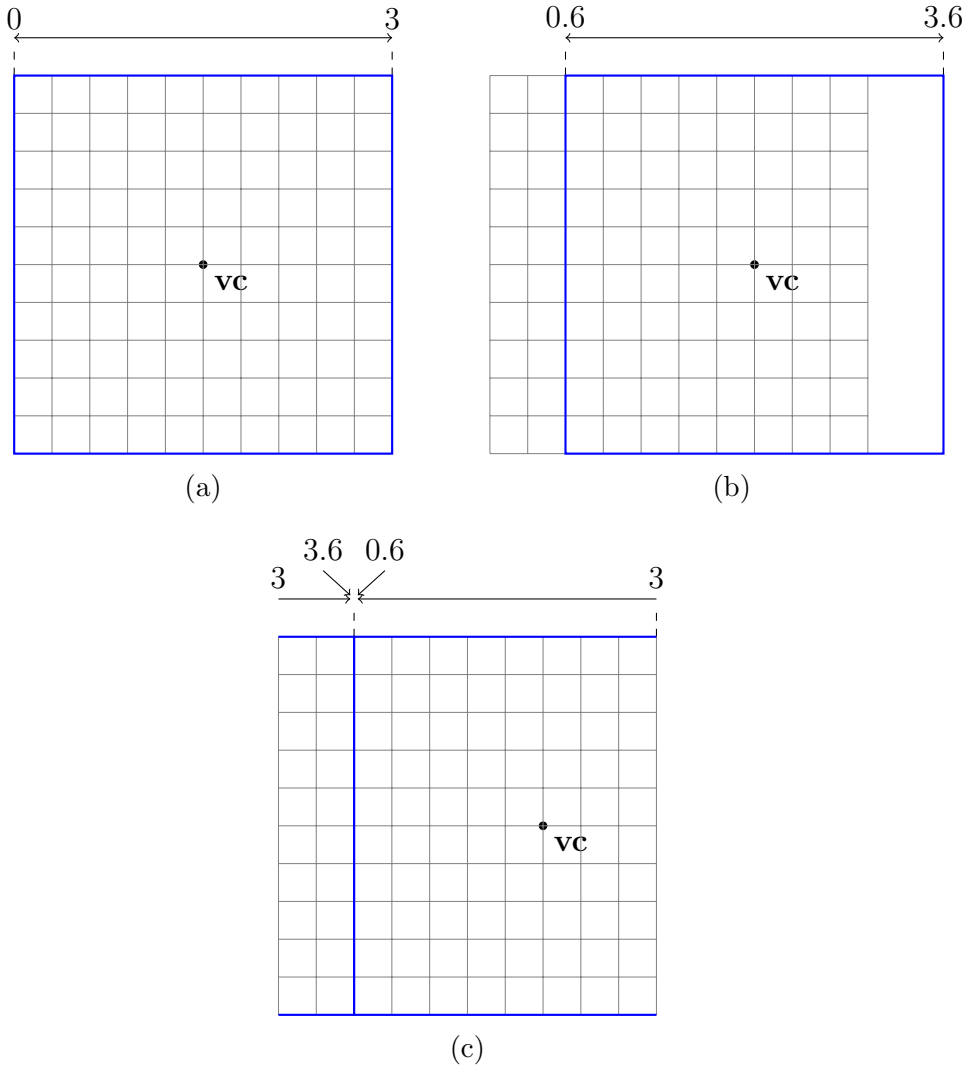


Figure 5.9: Illustration of the TSDF volume being virtually translated two voxels to the right. The blue line indicates the boundary for the world space in each direction. Subfigure (a) shows the initial setup. Subfigure (b) shows the adjusted boundary that maps to space that is outside the volume before the remapping. Subfigure (c) shows the adjusted boundary remapped to voxels that are too far from the voxel centre.

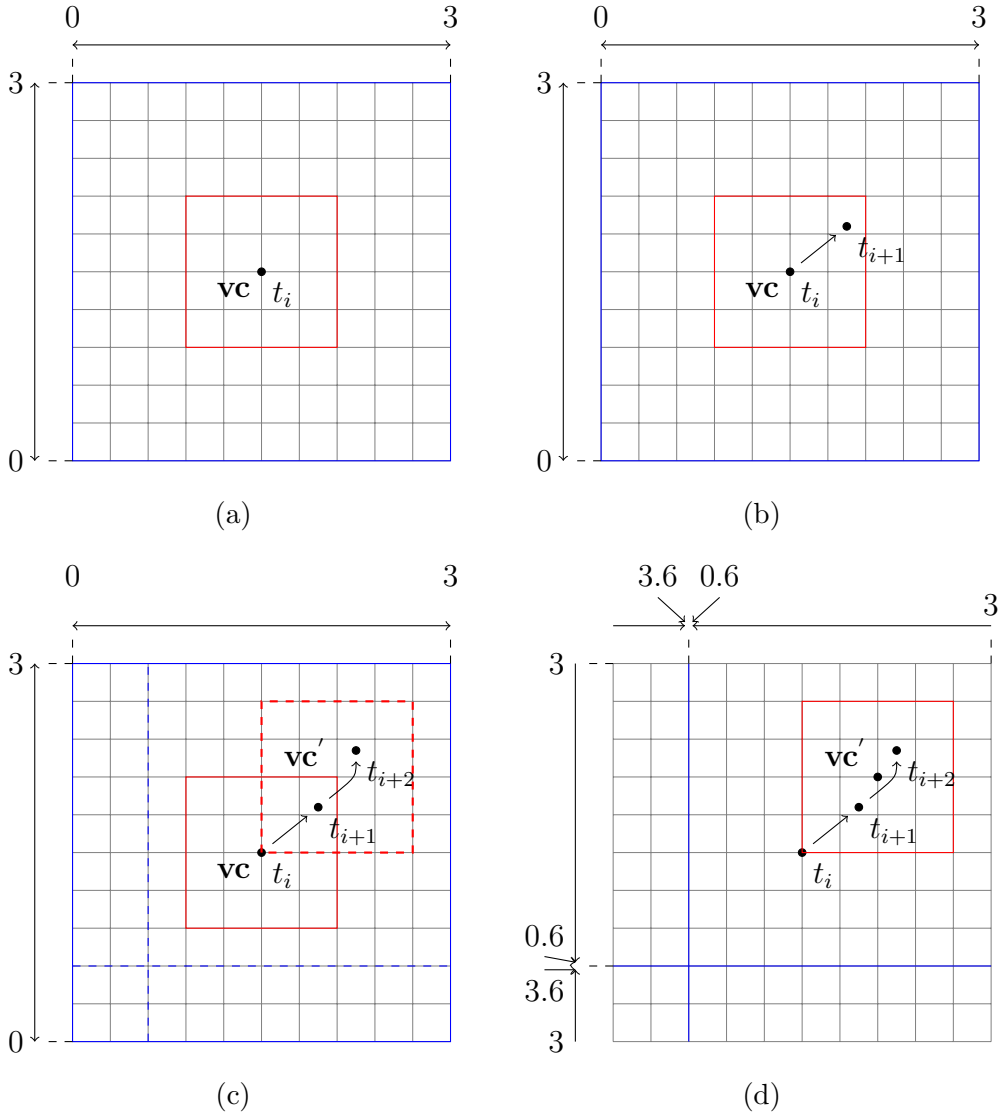


Figure 5.10: Subfigure (a) illustrates the initial setup of the volume with the red box representing the distance b from **vc** and the blue box representing the world space boundary. Subfigure (b) illustrates the camera moving from position t_i to position t_{i+1}. t_{i+1} is still within the threshold boundary b. Subfigure (c) illustrates the camera moving to the next position t_{i+2}. This position is outside the boundary of b. The dashed red box represents the adjusted boundary after moving to position t_{i+2}. Subfigure (d) illustrates the adjusted threshold boundary and the adjusted world space boundary after the voxel centre has been updated to **vc**′.

The TSDF volume now contains multiple integrated range images that represent the 3D surface. The next step is to extract the zero-crossings from the TSDF volume to be placed into the ICP algorithm as the destination points. The next section discusses how this is performed.

5.6 Raycasting

Section 5.4 discussed the use of destination points in the world space that are obtained from the ray caster; this section describes how they are obtained. In typical motion estimation using ranged images, a frame-to-frame technique is applied. These techniques assume that the camera is at a position that has the associated view. This means that the view state of the world is assumed to be correct in the previous frame. This assumption, when incorrect leads to motion drift. In the proposed system, a frame-to-model motion estimation is applied whereby the current model stored in the volume is compared to the incoming data from the RGB-D camera. The relative motion is computed using the surface points of the model stored in the TSDF volume and the source points from the incoming range surface. Since the previous frame is not assumed, if the motion estimation from the previous ICP computation is incorrect, the motion estimate can be corrected further when a new image is obtained from the range sensor.

Once a range surface has been integrated into the TSDF volume, the surface in the camera's frustum needs to be extracted. This is done by performing a ray casting operation. Just as in Section 5.4, the extracted surface points and the unit normals are stored as vertex maps and normal maps in the form of a pyramid. These are then used in the ICP algorithm as the destination vertex and normal maps for estimating the next camera transformation. To extract the zero-crossings, the estimated transformation matrix of the camera T_{i+1} and the camera parameters K are used to create a virtual range image. From the estimated position of the camera, a ray is cast from the camera's position to the image plane for each pixel. These rays are then increased in length until they encounter a zero-crossing in the TSDF volume. The world positions where these zero-crossings occur are stored in the vertex maps. As mentioned in Section 5.5, such a zero-crossing corresponds to the surface of the model. The vertex maps and normal maps are used for the frame-to-model ICP alignment. The vertex maps can also be used for visualising the current model's state and for point cloud extraction.

Since the ICP algorithm uses a coarse-to-fine approach with a multilevelled pyramid, the ray caster has to create its own multilevelled pyramid from the TSDF volume. This is achieved by scaling the camera parameters K, such that it produces images that have the same parameters as each of the images from the pyramid generated from the incoming RGB-D data. For each level

of the pyramid, l, the new camera matrix is computed as K^{-l}.

The transformation matrix T_{i+1}, obtained from the ICP algorithm is used as the position and orientation for the virtual camera, with $\mathbf{t_{i+1}}$ the translation vector and r_{i+1} the rotation matrix of the virtual camera. A vertex map is generated using the same camera parameters and image dimensions as the real RGB-D camera in our case, the Kinect sensor. A ray is cast for each pixel coordinate of the virtual range image from the camera's position to the image plane. This is done by using the camera matrix K^{-l}. This ray is then normalised to produce a unit vector. More specifically, Equation 5.30 is used to produce a unit vector.

$$\texttt{RayNext} = T_{t+1}K^{-l}\mathbf{u} \qquad \texttt{RayDir} = \frac{\texttt{RayNext} - \mathbf{t_{i+1}}}{|\texttt{RayNext} - \mathbf{t_{i+1}}|} \tag{5.30}$$

The ray caster is configured by choosing an increment size known as the `StepSize`. Initially, `RayDir` is shortened to 1 mm to determine the first voxel the ray intersects. Thereafter, a scaling factor j for `RayDir` is maintained. After each iteration, j is updated and `RayDir` is scaled to extract the TSDF value of the next voxel. The ray's length is incrementally increased until a zero-crossing between the current voxel and the next voxel is found. This approach has the possibility of stepping over voxels depending on the `StepSize`. However, the main reason this approach is used, is due to this algorithm being implemented on the Nvidia's CUDA framework that operates using single instruction multiple data. Single instruction multiple data (SIMD), allows for fast parallel execution of code that operates using the same instructions at the exact same time with different data. In this case, for multiple pixels at the same time. Figure 5.11 (a) is used as an example in this section. Figure 5.11 (b) shows rays being cast from the virtual camera and generating 2D points at the positions where the TSDF values become negative. A zero-crossing is found on a ray when the first voxel along the ray with a negative TSDF value is found. The corresponding 3D world position is computed using the location of the voxel where the zero-crossing occurs. This is computed by converting the voxel location to world coordinates and is then stored in the vertex map. The normal map is computed as in Equation 5.15. Algorithm 3 shows the ray casting operation performed for each pixel, $\mathbf{u}$, in the depth and colour map. The vertex maps extracted from the ray casting operation store the 3D world coordinates of the surface points of the model.

In addition to the vertex maps and normal maps being extracted, a colour map is also extracted, for the highest resolution image only. This is done by extracting the value stored in the colour volume, as mentioned in Section 5.5.2, at the same position where the zero-crossing is detected. From the vertex map, a virtual range image can be created as visualisation by applying the associated inverse transformation matrix and storing the resulting z value for each pixel.

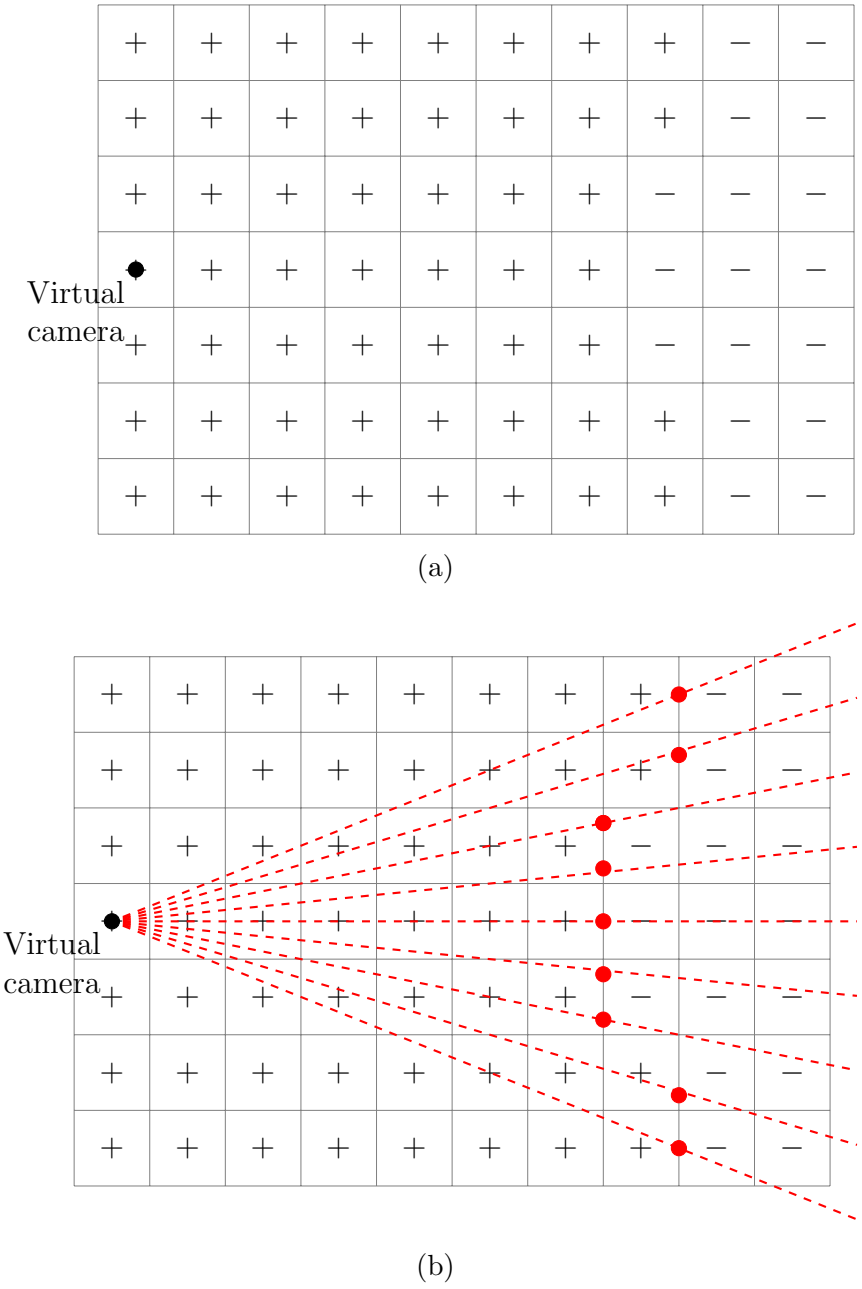


Figure 5.11: Subfigure (a) illustrates the sign of the truncated signed distance function volume and the virtual camera location represented by a point. Subfigure (b) illustrates rays being cast from the virtual camera into the TSDF volume with the points of zero-crossings shown as red points.

This will create a virtual range image, along with the colour map extracted during this process, to produce a virtual RGB-D image.

Data: T_{i+1}, $\mathbf{u}$, K, $\mathbf{j}_{max}$
1 $\texttt{RayNext} \leftarrow T_{t+1}K^{-1}\mathbf{u}$;
2 $\texttt{RayDir} \leftarrow \frac{\texttt{RayNext}-\mathbf{t_{i+1}}}{|\texttt{RayNext}-\mathbf{t_{i+1}}|}$;
3 **while** $\mathbf{j} < \mathbf{j}_{max}$ **do**
4 $\mathbf{position} \leftarrow \mathbf{t_{i+1}} + \texttt{RayDir} * \mathbf{j}$;
5 $\mathbf{tsdf} \leftarrow \texttt{GetTsdfValue(position)}$;
6 **if** $\mathbf{tsdf} < 0$ **then**
7 **return position**;
8 **end**
9 $\mathbf{j} \leftarrow \mathbf{j} + \texttt{StepSize}$;
10 **end**

Algorithm 3: The ray casting operation performed for each pixel in the virtual pyramid images constructed.

5.7 Point Cloud Extraction and Hole Filling

The system as discussed so far creates a 3D virtual model by integrating new range images into the TSDF volume and using the ray caster to assist in computing the relative motion between the model and the new RGB-D data. The task of extracting the surface of the 3D model stored in the TSDF volume as a point cloud is now considered. A simple approach to this is using the vertex map and the associated normal maps that result from the ray casting operation. In our implementation however, the ray caster's vertex map is not used to generate the final surface that is used for the model. This implementation rather aims to extract a point cloud that has a uniform distance between each point that represents the surface and extract implicit information about the model's structure. The uniform point cloud assists the mesh reconstruction algorithm in generating the final model which will be discussed in Chapter 6. The TSDF volume allows us to perform some basic hole-filling for the surface. This gives the mesh generation algorithm more information about the model's surface, which would in turn enables a more accurate representation of the model to be obtained. Figure 5.12 illustrates the hole-filling procedure.

In Figure 5.12 (b) the ray caster is used to extract the surface points for use in the ICP algorithm from the current point of view of the camera. Since the TSDF volume uses negative values to indicate the interior of the model, more information about the surface points are implicitly present. Every zero-crossing provides a surface measurement in the TSDF volume. In certain

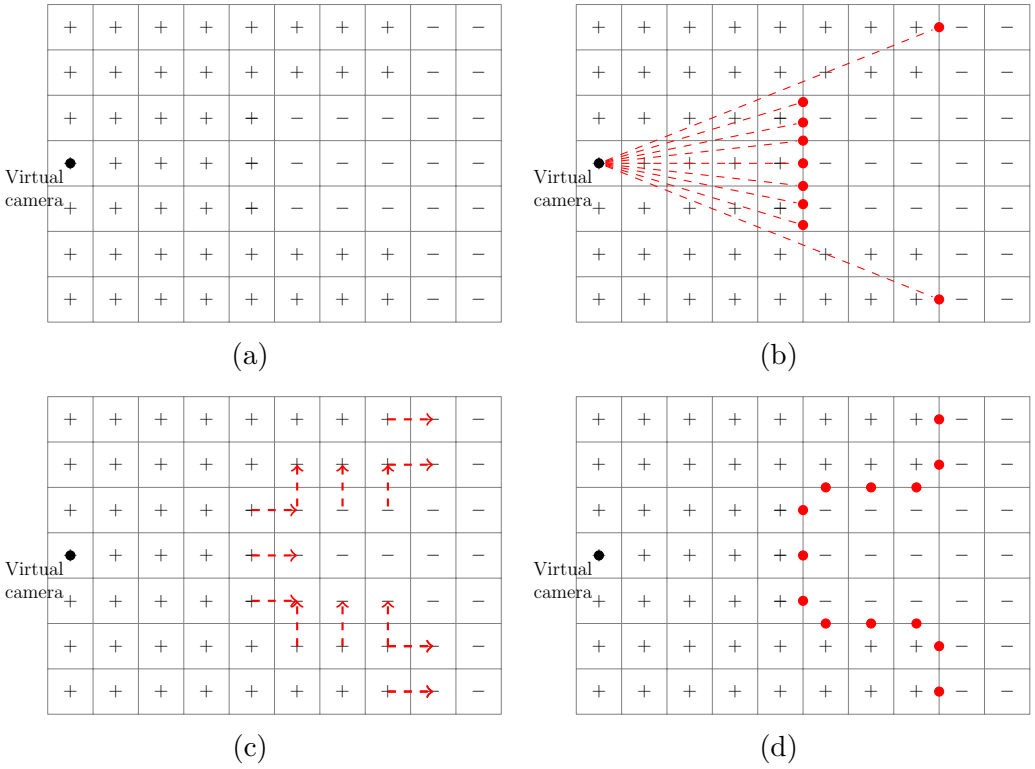


Figure 5.12: The dashed lines show where a zero-crossing is being checked, the red circles indicate the 3D point detected by the ray. Subfigure (a) shows the signs of the TSDF volume entries. Subfigure (b) shows the points extracted from the TSDF volume using the regular ray casting techniques. Subfigure (c) shows rays demonstrating the hole-filling techniques that extracts the hidden structure of the model by checking for zero-crossings along the axes. Subfigure (d) shows the points generated by checking each axis.

areas that have not been directly measured there is implicitly a surface. In Figure 5.12 (c) it is shown that there are multiple areas that contain zero-crossings that could be used as surface points if extracted. Instead of using the original ray caster for the point cloud extraction, a second ray casting algorithm is used. This algorithm simply casts a ray along each axis. Once the ray detects a zero-crossing, a point is generated. The ray continues to detect these crossings until the ray passes through the volume. This ray casting technique is only executed in the area that needs to be reset, as discussed in Section 5.5.2.

Fusing the generated points would produce a dense point cloud, with the density determined by the voxel cell size. A voxel grid filter removes points that are too close to each other, and is used to reduce the size of the generated point cloud for further processing: The resulting point cloud is then output for

processing by the mesh generation algorithm (discussed in the next chapter).

5.8 Summary

This chapter described the approach to integrate multiple range images along with the associated RGB images to generate a model represented in a 3D volume. The model is then extracted from the 3D volume to generate a 3D point cloud. The algorithm has the ability to extract surface points from the implicit information provided by the TSDF volume using the hole-filling techniques mentioned. The point cloud extracted from this system is passed into the next component, mesh generation, which aims to create edges and faces for the point cloud. This process is described in the subsequent chapter.

Chapter 6

Mesh Reconstruction

Once the points have been extracted from the TSDF volume and fused into a single point cloud, they are provided to the mesh reconstruction algorithm. The ball-pivoting algorithm (BPA) by Bernardini, Mittleman, Rushmeier, Silva and Taubin (1999) is used to reconstruct the mesh.

Given a point cloud, the purpose of mesh reconstruction is to create edges between points and faces between edges to produce a 3D surface model.

6.1 Overview

The BPA is a conceptually simple approach for surface reconstruction of a point cloud. This technique is based on the idea of rolling a ball along a surface. This algorithm is explained in $\mathbf{R}^3$, but illustrations of the analogous process are given in $\mathbf{R}^2$ for simplicity. A basic example of the operation of the BPA is illustrated in Figure 6.1. The algorithm requires a ball of given radius p, and should be large enough that the ball cannot pass through the point cloud anywhere without making contact with three points.

The ball attempts to pass through the point cloud which results in it making contact with three points: Figure 6.1 (a) illustrates this with two points in $\mathbf{R}^2$. Edges are created between the initial three points the ball makes contact with. Once the ball has made the initial contact, the pivoting operation can begin. The pivoting operation involves selecting one edge in contact with the ball and pivots the ball on this edge. The ball is pivoted until it comes into contact with any other point as shown in Figure 6.1 (b). Once this occurs, two new edges are created, between the two points on the pivot edge and the new point. This process continues until no new edges are created. In this implementation the creation of edges are performed in a breadth-first manner. After this, the ball is placed in the point cloud such that it does not make contact with any point it has already touched. Then, the pivoting operation is repeated from the three new points. An example of this in $\mathbf{R}^3$ is illustrated in Figure 6.2.

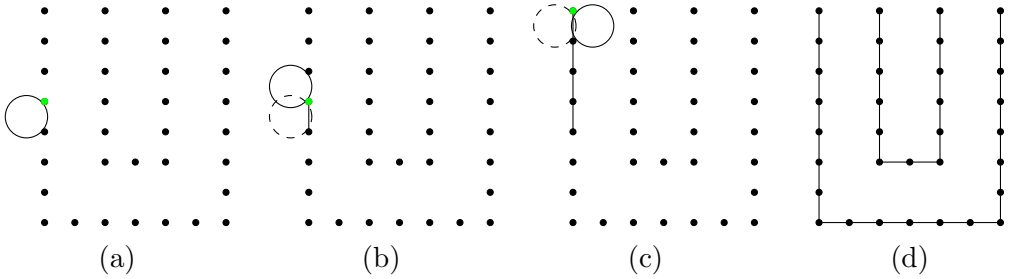


Figure 6.1: Subfigure (a) shows the initial contact the ball has with the point cloud. This creates edges between the points it makes contact with. The ball then pivots about a point until it makes contact with any other point. Subfigure (b) shows the ball after the pivoting operation. Subfigure (c) shows the ball pivoting until it again makes contact with a point it has already hit. The ball is too small to make contact with the next point. The ball starts its pivoting operation from the original two points of contact and moves in the opposite direction to complete the rest of the algorithm. Subfigure (d) illustrates the completed mesh.

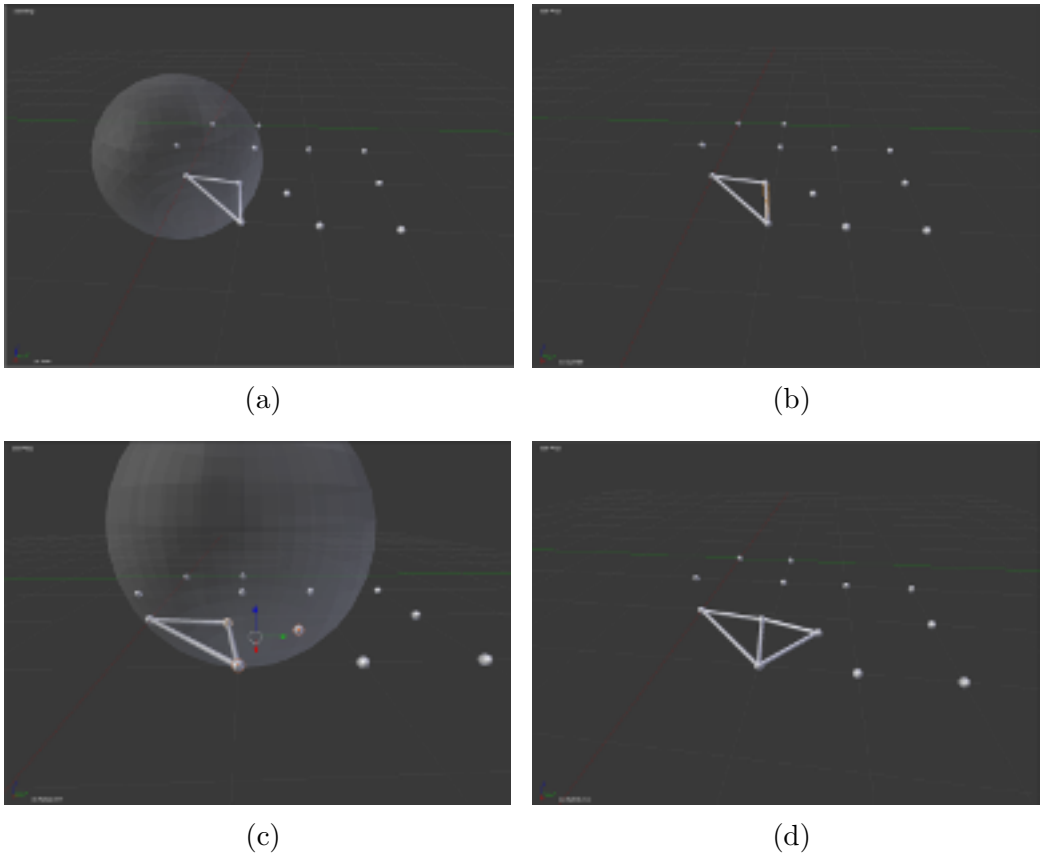
Point clouds generally do not contain equidistant points or are not uniform, and therefore the ball may not be large enough as shown in Figure 6.1 (c). This can result in gaps, where edges were unable to be rebuilt using the ball. Instead, a larger ball could be used as shown in Figure 6.3. However, picking a larger ball instead produces an overly smoothed version of the surface and could use a subset of the point cloud. This shows that using a large ball can result in a loss of detail. To avoid these issues in Section 6.2.4 we discuss a method of varying the ball size.

6.2 Pivoting Algorithm

Given a 3D point cloud of a manifold mesh, a suitable surface can be constructed using a p-ball. Assuming the point cloud is dense enough that the p-ball cannot pass through it without touching at least three points, the reconstructed mesh is an approximation of the true surface.

The algorithm follows the advancing-front paradigm to incrementally build a triangulated mesh. The BPA requires a list of points, σ, a list of each point's associated surface normal vector, σ^n, the radius p of the ball that is used during the reconstruction, and in our implementation the RGB value associated with each point.

The advancing-front paradigm progressively adds mesh elements starting from the boundaries of the mesh area and works its way outwards to the unmeshed area. This causes the boundary to advance throughout the point



(a) (b)

(c) (d)

Figure 6.2: This figure illustrates the pivoting operation in $\mathbf{R}^3$. Subfigure (a) shows the the initial ball touching three points in the point cloud with the edges between each of these points. Subfigure (b) illustrates the highlighted edge that the pivoting operation will be performed on. Subfigure (c) illustrates the sphere touching the two original points from the initial three and the new third point that resulted from the pivoting operation (highlighted in orange). Subfigure (d) illustrates the newly created edges resulting from the pivoting operation.

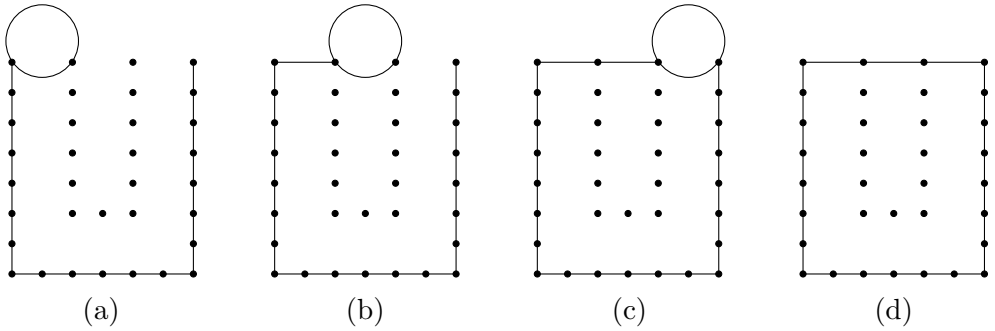


Figure 6.3: Using a large ball. Subfigure (a) - (d) show that the surface is overly smoothed. This problem is addressed in Section 6.2.4

cloud, incrementally building up the mesh. Pseudocode for the BPA is given in Algorithm 4, and is referred to throughout the rest of the chapter. A collection of fronts are represented by F: each element F_i represents a single front. Each front consists of a list of edges. All the edges in a front form a loop representing the boundary of the region that has been reconstructed, this only applies to the 3D case. In this chapter the terms loop and front are used interchangeably. Initially, the first front consists of three edges obtained from the `FindSeedTriangle` operation. Each edge $e_{i,j}$ on the front is stored in a data structure called a `FrontEdge`. A `FrontEdge` contains the two end-points of the edge, σ_i and σ_j, the opposite vertex σ_k that completes the triangle formed by the ball, the centre of the ball that touches all three vertices, and the edge state. The edge status is either active or boundary.[1]

Due to the possibility of having discontinuous regions in the point cloud, a collection of fronts are used instead of a single front. This is due to the ball pivoting along a `FrontEdge`. When the pivoting operation touches a new or already used point the front is updated using the `Join` and `Glue` operations discussed in Section 6.2.3 to keep the front in a consistent state. Figure 6.4 shows a collection of fronts, F_1, F_2 and F_3. The original implementation described by Bernardini, Mittleman, Rushmeier, Silva and Taubin (1999) obtained points in a region of the point cloud by using a 3D grid to subdivide the volume of the point cloud into cubic subvolumes of side length $(2p)$. Points would then be placed in the corresponding subvolume. However, if the subvolume is small the number of subvolumes created to represent the volume could be too large. This approach allows for a constant lookup time to access the relevant subvolume and then a linear search can be performed through the points in that subvolume. Initial testing of this method worked well; however, this approach

[1]The original implementation also uses a frozen state. The frozen state is used to allow streaming of the point cloud data. However, this was not implemented in this system due to time constraints.

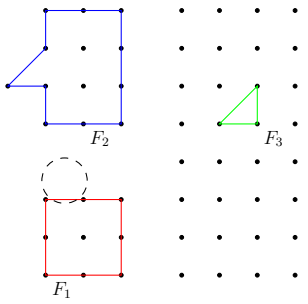


Figure 6.4: Fronts F_1 and F_2 on the point cloud and a new front F_3 obtained from the `FindSeedTriangle` operation. Note that the ball is unable to pivot between F_1 and F_2.

uses a large amount of memory. When the point cloud covers a large area with a high point density too many subvolumes are created and the system is unable to allocate the necessary memory. Thus, in the proposed system a kd-tree is constructed to store the points from the point cloud. This provides fast spatial queries with a low construction time. In the following subsections, the details of obtaining a valid seed triangle are discussed, followed by the pivoting operation, and the required `Join` and `Glue` operations for updating the fronts.

6.2.1 Finding a Valid Seed Triangle

A front starts with three initial `FrontEdges` in it. These `FrontEdges` are needed to perform the pivoting operation to construct the mesh. They are obtained by finding a seed triangle (line 16 of Algorithm 4). Finding a valid seed triangle creates a new front for the algorithm to operate on. The seed triangle is found by

- Picking a point σ_a that has not yet been used in the reconstruction process.
- Obtain a list of all the points within a $2p$ radius of σ_a.
- Build a triangle by considering σ_a with all the point pairs σ_b and σ_c of points from the list.
- For each triangle: (a) Verify the normals of each of the points are facing a similar direction, this is referred to as the *consistent normals* condition, Figure 6.5 illustrates the pivoting operation with inconsistent normals. (b) Construct a p-ball where the centre lies in a similar direction of the point normals and touches all three vertices, σ_a, σ_b and σ_c, and ensure that it does not contain any other points from the point cloud within the ball, as shown in Figure 6.6.

```
   input: σ, σ^n, p
 1 while TRUE do
 2     while e_{i,j} = GetActiveEdge() do
 3         σ_k = BallPivot(e_{i,j});
 4         if σ_k != null AND (NotUsed(σ_k) OR OnFront(σ_k)) then
 5             Join(e_{i,j}, σ_k);
 6             if e_{k,i} ∈ F and e_{i,k} ∈ F then
 7                 Glue(e_{i,k}, e_{k,i}, F);
 8             end
 9             if e_{j,k} ∈ F and e_{k,j} ∈ F then
10                 Glue(e_{k,j}, e_{j,k}, F);
11             end
12         else
13             MarkAsBoundary(e_{i,j});
14         end
15     end
16     if e_{i,j}, e_{j,k}, e_{k,i} = FindSeedTriangle() then
17         CreateNewFront(e_{i,j}, e_{j,k}, e_{k,i});
18     else
19         break;
20     end
21 end
```

Algorithm 4: A overview of the ball-pivoting algorithm. The inputs are a list of points, σ, a list of normal vectors, σ^n and the ball size p. The `BallPivot` function returns the new vertex the ball makes contact with. The explanations of the other methods are in the ensuing subsections.

The first reported triangle to have these properties is considered a valid seed triangle. For each `FrontEdge` created the *opposite vertex* is the vertex on the ball that is not used in the current edge. For example if the valid seed is found using the points σ_a, σ_b and σ_c, the `FrontEdge` containing σ_a and σ_b uses σ_c as the *opposite vertex*. The created `FrontEdge`s are marked as active edges.

In practice, the point cloud has noisy point data, and additionally there may be numerical stability issues depending on the point cloud density and scaling factors. For this reason the search for nearby points uses a radius of $2.2p$ instead of $2p$. Similarly, when determining whether any additional points lie within the generated ball, the distance between the centre of the sphere and the test points are scaled by a factor of 1.05. This gives the effect of ignoring points that are close to the surface of the sphere. The other points are ignored because the system is designed to only handle three points for the `FrontEdge`. This forces any additional contact points to be ignored.

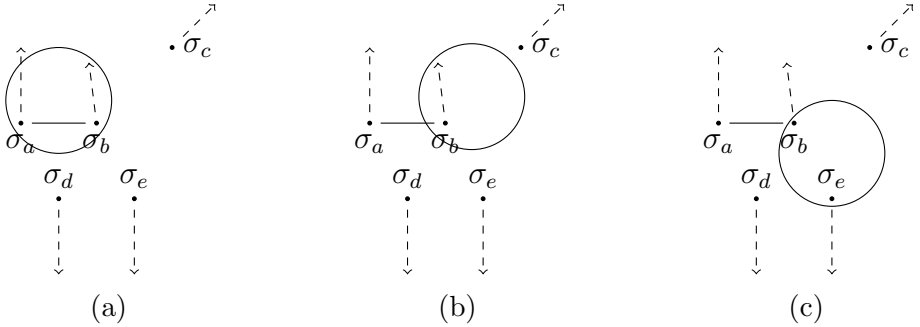


Figure 6.5: This illustrates the 3D points projected on a 2D side view and the use of the normals to ensure the created surface is consistent. Subfigure (a) illustrates the current location of the sphere, a series of points and the normal direction of each point. An edge is created between the points σ_a and σ_b that the ball is making contact with. Subfigure (b) illustrates a single pivoting operation, with the sphere too small to make contact with the point σ_c. Subfigure (c) illustrates the result of pivoting to the point with inconsistent normals, the normals are pointing in opposite directions. Since the normals are inconsistent, the generated surface is rejected.

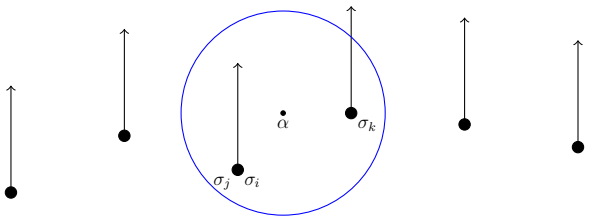


Figure 6.6: The 3D points projected on a 2D side view, σ_i and σ_j are two points that lie on the Z plane, σ_i with a positive Z value and σ_j a negative Z value. This illustrates the direction that the centre of the ball must be relative to the point it is in contact with. For a ball to be valid the centre point must be located in the direction of the normals. If it is not, it is not a valid centre location for a ball. This shows a valid sphere in the outward half-space with its centre located at α.

6.2.2 Pivoting Operation

The pivoting operation starts by selecting an active edge from any front illustrated in line 2 of Algorithm 4. Once an active edge is obtained, the ball is pivoted on the edge and reports the point σ_k as the new point that the ball has touched (line 3). The selected `FrontEdge` contains the data for a triangle formed from a previous pivoting operation or an initial seed triangle obtained as in Section 6.2.1. The `FrontEdge` $e_{i,j}$ that represents the edge between σ_i and σ_j contains information about the edge endpoints σ_i and σ_j, as well as σ_q the *opposite vertex* that the ball has made contact with, and c_{ijq} the ball centre that touches all three vertices. Denote the centre point of the edge $e_{i,j}$ by m. If the ball pivots on the edge $e_{i,j}$, the centre point of the pivoting ball always has to be at exactly $|c_{ijq} - m|$ from m. All possible centre points for the new p-ball forms a circle. This trajectory of possible locations is shown as a dashed circle, γ, in Figure 6.7. The ball-pivoting operation pivots the ball on the edge $e_{i,j}$ and returns the first point in the point cloud that it makes contact with. To simulate the pivoting operation, all points within the radius of $2.2p$ of point m are considered as the new contact point for the pivoting operation. The ball is then fit between the original endpoints σ_i, σ_j of $e_{i,j}$ and each point σ_o from the points considered as the new contact point. Before σ_o is accepted as a valid contact point with the new ball position centre c_{ijo}, the algorithm needs to ensure that no other points lie inside the new ball centred at c_{ijo}, and the normals of the point's σ_i, σ_j and σ_o are facing a similar direction. Once the point is considered valid, the algorithm needs to ensure that the point has the minimum angle between the lines $m - c_{ijq}$ and $m - c_{ijo}$, this ensures it is the first point of contact. This is illustrated in Figure 6.7. Once the centre with the smallest angle has been found from all the points considered as the new contact point, this new point σ_k is reported back to the rest of the algorithm.

6.2.3 Join and Glue Operation

Once the pivoting operation is complete a candidate triangle (σ_i, σ_j, σ_k) is created. If the candidate triangle is added to the mesh, the front needs to be updated to expand the boundary. The update is performed by the `Join` and `Glue` operations. The candidate triangle can still be rejected by the `Join` operation, if the addition of this triangle forms a non-manifold mesh. The front first needs to be adjusted to add the newly created edges to the front, allowing for an expansion of the front. This is accomplished by the `Join` operation. The `Glue` operation is used after the `Join` operation to remove duplicate `FrontEdge`'s.

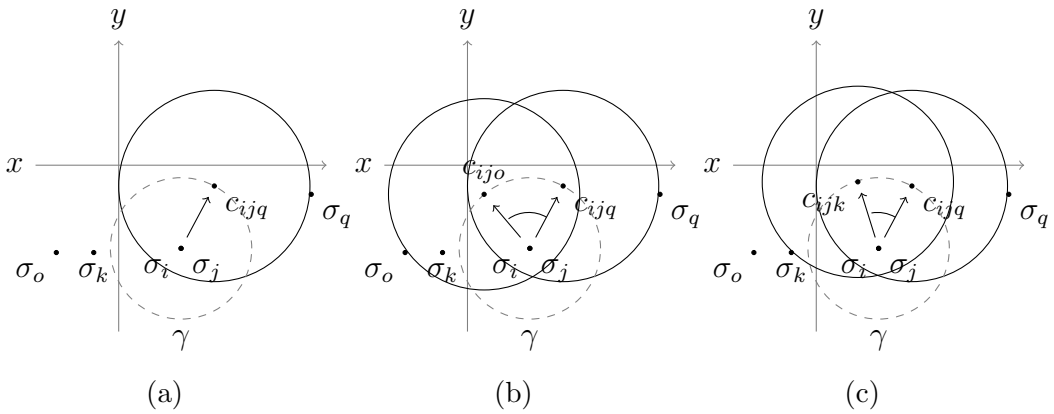


Figure 6.7: This illustrates the pivoting operation viewed from the side view as shown in Figure 6.6. In this illustration σ_o, σ_k and σ_q are coplanar. Subfigure (a) shows the points σ_i, σ_j, σ_q, the ball centre c_{ijq} and the possible locations of a new centre along the trajectory γ. Subfigure (b) illustrates a new ball at centre point c_{ijo}. Subfigure (c) illustrates a new ball at centre point c_{ijk}, with the angle between the vector $m \rightarrow c_{ijq}$ and $m \rightarrow c_{ijk}$ smaller than the angle between $m \rightarrow c_{ijq}$ and $m \rightarrow c_{ijo}$. The ball-pivoting operation thus returns the point σ_k as the first point it makes contact with.

Join Operation

The `Join` operation considers three situations that could affect the topology of the front. The simplest situation is that the new point σ_k has not been used in the reconstruction yet. The front is simply updated to include the new point and adds this triangle to the mesh. The front is updated by removing $e_{j,i}$ and adding $e_{j,k}$ and $e_{k,i}$ as shown in Figure 6.8.

The second situation is if σ_k has already been marked as used. Here there are two possible cases. The first, shown in Figure 6.9, is that the candidate point is part of the internal mesh, and not on any front. If this is the case, generating a triangle using the endpoints of the pivoting edge and the internal point results in generating a non-manifold mesh, (see Section 2.4.4). In this situation, the edge $\sigma_{j,i}$ is simply marked as a boundary edge in the `Join` operation. The third situation is that σ_k is a point that has been marked as used and is located on any of the fronts as shown in Figure 6.10. In this situation the candidate triangle is checked to ensure a non-manifold mesh is not created if it is added to the mesh. This is checked by the number of edges between two points that the candidate triangle is going to add. If there are currently two edges on the front between points σ_i and σ_j, the triangle is rejected. [2] Otherwise, the `Join` operation is performed as stated in the first

[2] Any edge can only have a maximum of two faces if the mesh is manifold (Section 2.4.4).

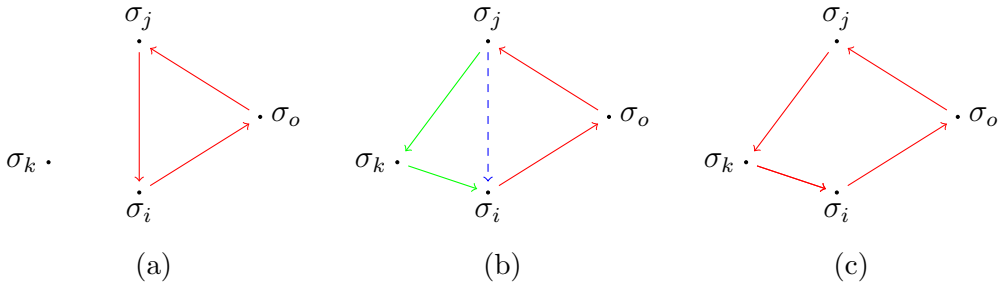


Figure 6.8: In this figure the green lines indicate the new edges created after pivoting around the dashed edge (in blue), with the pivoting edge removed from the front after the pivoting operation. Subfigure (a) shows the initial front in red. Subfigure (b) shows the removal of edge $e_{j,i}$ from the front in blue and the new edges added to the front in green, $e_{j,k}$ and $e_{k,i}$. Subfigure (c) shows the front after the `Join` operation.

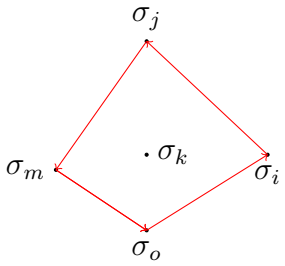


Figure 6.9: This figure illustrates an example for the internal point. If the pivoting operation occurred on edge $e_{i,j}$ and it reported the new point of contact as σ_k (located in the meshed area) this would result in the candidate triangle being rejected as it already exists in the mesh.

situation using the used point σ_k and the endpoints of edge $e_{i,j}$, however this can create coincident edges as shown in Figure 6.11. The `Glue` operation is used to remove such coincident edges introduced by the `Join` operation.

Glue Operation

The `Join` operation can lead to four possible situations in which coincident edges are arranged, shown in Figure 6.12. The possible situations are: a consecutive loop, a closed loop, a split loop and a merge loop.

The consecutive loop occurs when two coincident edges are consecutive in the loop with the presence of other edges in the front. This is illustrated in Figure 6.13. The `Glue` operation simply removes both consecutive edges from

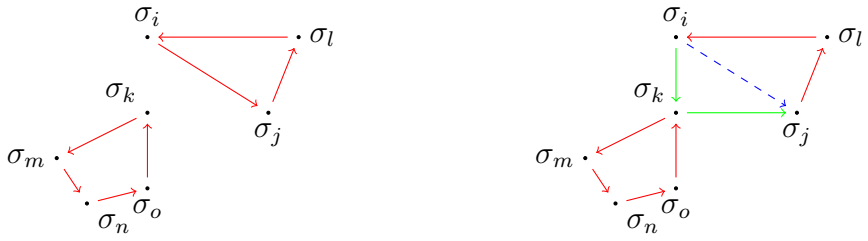


Figure 6.10: Subfigure (a) shows two fronts with the pivoting operation on $e_{i,j}$ that makes contact with the point σ_k. Subfigure (b) shows the updated front. Here the green lines indicate the new edges created after pivoting around $e_{i,j}$ (in blue), with the pivoting edge removed from the front after the pivoting operation.

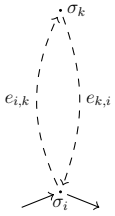
Figure 6.11: Subfigure (a) and (b) shows the third situation in the `Join` operation when pivoting on edge $e_{i,j}$ and reporting a used contact point σ_k that results in creating a coincident edge $e_{k,j}$. In this figure the green lines indicate the new edges created after pivoting around $e_{i,j}$ (in blue), with the pivoting edge removed from the front after the pivoting operation.

the front.

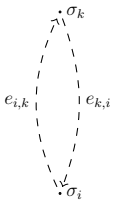
The closed loop is the simplest case of the `Glue` operation, this occurs when two coincident edges form a loop with no other edges in the front. This typically forms when two fronts exist and one of them starts to close in on itself as shown in Figure 6.14 and 6.15. The `Glue` operation simply removes both edges and the front is discarded.

A split loop occurs when the edges are not consecutive and belong to the same front. This occurs when a series of joins are performed that split the front in two, as illustrated in Figure 6.16. In this case, the front is split into two fronts by removing the two coincident edges.

The final situation, a merge loop, occurs when two coincident edges belong to two different fronts (and are thus not consecutive). This is rectified by

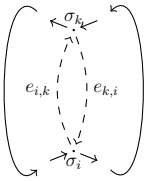


(a) Consecutive Loop



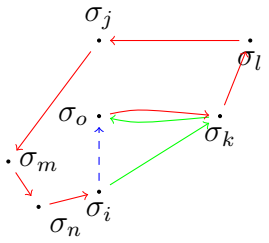
(b) Closed Loop

(c) Split Loop

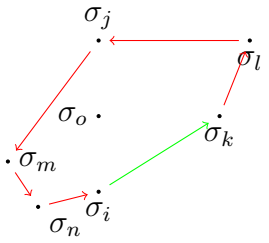


(d) Merge Loop

Figure 6.12: This illustrates the four situations that can occur in the `Glue` operation. The dashed lines indicate the edges that cause the situation to occur and are modified during the `Glue` operation and the solid lines indicate a sequence of edges that are not modified during the `Glue` operation.

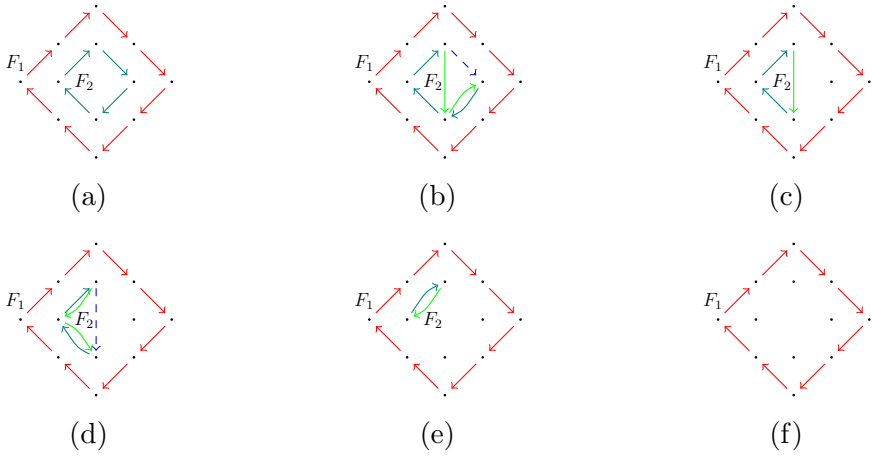


(a)



(b)

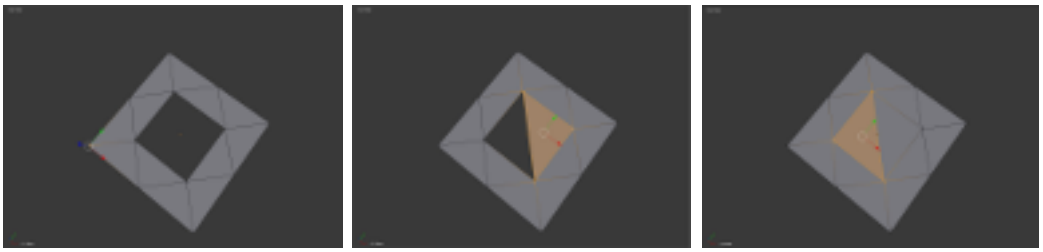
Figure 6.13: In this figure the green lines indicates new edges created after pivoting around the dashed edge (in blue), with the pivoting edge removed from the front after the pivoting operation. Subfigure (a) shows the situation where the consecutive loop occurs after a `Join` operation. Subfigure (b) shows the result after the `Glue` operation.



(a) (b) (c)

(d) (e) (f)

Figure 6.14: In this sequence of figures the green lines indicate new edges created after pivoting around dashed edges (in blue), with pivoting edges removed from the front after the pivoting operation. Subfigure (a) illustrates the initial setup with two fronts F_1 (shown in red) and F_2 (shown in teal). The area between the two fronts is the area that has been meshed, and the internal area of the front F_2 is an unmeshed region. All points in this figure have been marked as used. Subfigure (b) shows the BPA performed on the front F_2 and a consecutive loop deletion to obtain Subfigure (c). Subfigure (c) shows the situation after pivoting over the edge. Subfigure (d) shows the front F_2 with two consecutive loops following each other. Subfigure (e) shows the removal of a consecutive loop, resulting in a single closed loop. Subfigure (f) shows the result after removing the closed loop. For clarity, the meshing process is demonstrated in a 3D viewer in Figure 6.15.



(a) (b) (c)

Figure 6.15: Subfigure (a) illustrates the initial setup as in Figure 6.14. Subfigure (b) illustrates the first pivot operation, reducing the size of the front F_2. Subfigure (c) illustrates the second pivoting operation removing the front F_2 entirely.

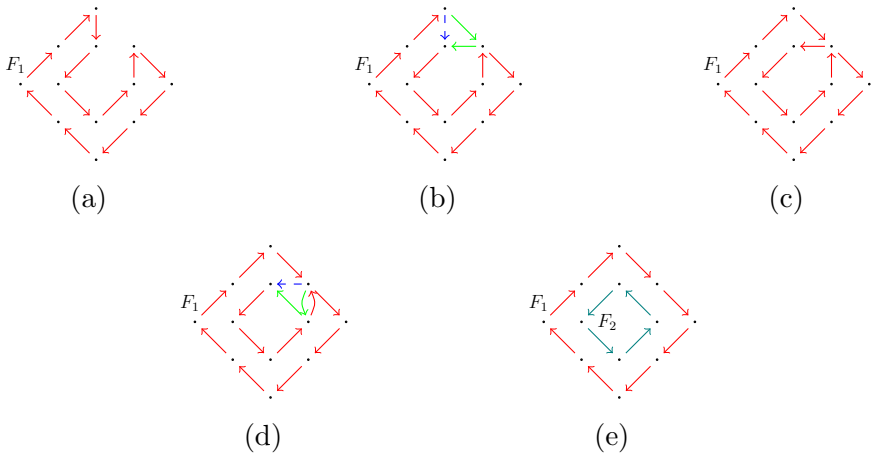


Figure 6.16: Example of how a split loop occurs. In these diagrams the green lines indicate new edges created after pivoting around dashed edges (in blue), with the pivoting edges removed from the front after the pivoting operation. Subfigures (a) - (d) illustrates a single front as the BPA pivots along the front. Subfigure (d) shows the situation where the split loop occurs. Subfigure (e) illustrates the result of the split loop.

removing the coincident edges and merging the two fronts into a single front as shown in Figure 6.17.

6.2.4 Multiple Passes

The effectiveness of the BPA is restricted by the choice of ball radius: The reconstruction process only reconstructs areas of the mesh where points are locally dense enough that the pivoting operation can make contact with other points. This could result in an incomplete and disjoint mesh. This is addressed by, instead considering an increasing sequence $\beta = [p_0, p_1, \ldots, p_n]$ of ball radii. The algorithm is only modified slightly: Once an initial pass of the algorithm is completed, using p_0 as the ball radius, the algorithm must then attempt to find a seed triangle using p_1. This is achieved by iterating through all the edges on all fronts and attempting to fit the p_1-ball between the edge and its opposite vertex. If a seed triangle is found the edge is marked as an active edge. After this, the BPA starts again using the `FrontEdge`'s that have been marked as active with the ball radius of p_1.

Since the BPA is very dependent on the input radii, a smart method of selecting these radii has to be performed. The BPA algorithm depends on the density of the point cloud for the ball radii to make contact with neighbouring points. The scale of the point cloud does not matter in the BPA. A general starting point for selecting a ball radius would be to use a radius slightly larger

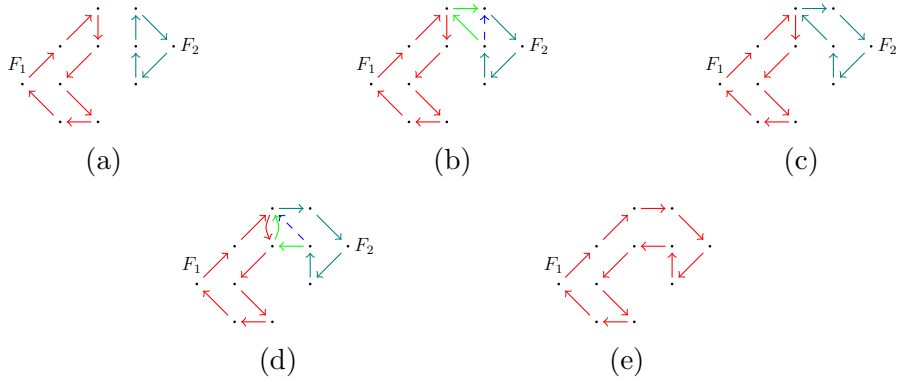


Figure 6.17: Example of how a merge loop occurs. In these figures the green lines indicate new edges created after pivoting around dashed edges (in blue), with the pivoting edges removed from the front after the pivoting operation. Subfigure (a) illustrates two fronts close together. Subfigures (b) and (c) illustrates how a merge loop can occur. Subfigure (d) shows the situation where the merge loop occurs. Subfigure (e) shows the result after a merge loop: the two fronts have been combined into one.

than the average distance between points. Analysing the point cloud, knowing how it was obtained and if there are large disconnected regions of points could significantly help select radii for the BPA. In this thesis the TSDF volume with the voxel grid filter provided a point cloud that has been arranged in a grid-like pattern with points separated by 1 voxel. The size of a voxel can be calculated using the parameters passed into the KinectFusion system.

6.3 Summary

This chapter presented the BPA which is employed to build triangles between the points of a point cloud. The algorithm receives a point cloud, with the associated surface normals for each point as input and attempts to create a sphere that can be placed between three points. These three points form a triangle that forms part of the mesh as the seed triangle. Once a seed triangle is found, pivoting operations are performed using the sphere. The pivoting operation pivots the sphere on one of the triangle's edges until the sphere touches another point in the point cloud such that the normals of these points are facing in a similar direction. A triangle is formed between the pivoting edge and the new contact point. The new edges of the resulting triangle are added to the front as active edges, waiting for a pivot operation to later be performed using them. The front is then updated to correctly represent the boundary of the unmeshed area using the `Join` and `Glue` operations. The pivoting operation is performed until no more active edges are found. Once no

more active edges are found, all the valid triangles found form the faces of the point cloud. Chapter 7 presents a number of experiments using the proposed system for 3D reconstruction and mesh generation and considers the overall performance and results of the system.

Chapter 7

Experiments

This chapter details various experiments to test the accuracy, quality and performance of the implemented reconstruction system, and discusses the results obtained. The experiments are described in two sections, one considering 3D reconstruction and the other mesh reconstruction. The 3D reconstruction section focuses on building point clouds representing the environment. Analysis of the reconstruction focuses on the accuracy of the estimated camera trajectory, and the resulting output point cloud, and considers different approaches to hole-filling using the volumetric information provided. The mesh reconstruction section analyses the results obtained from the BPA.

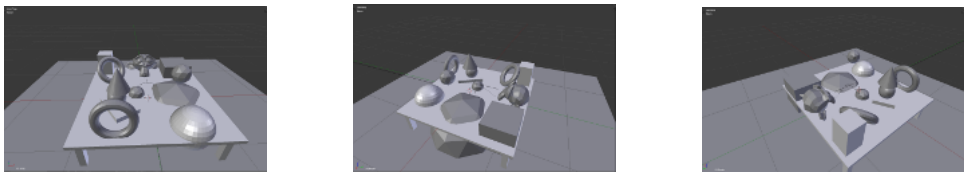
7.1 Data sets

These experiments of the system are performed on data sets from two synthetic scenes generated with Blender. The camera is animated to simulate moving around the scene.

These data sets are constructed from two synthetic scenes for which the associated ground truth of the camera is thus known. The ray tracer program, Blender (Blender Online Community, 2018), is used to create each frame for the synthetic data sets. During a scene render in Blender, a Z map is calculated. The map contains the distance for each pixel from the centre of the camera to the first triangle that is intercepted by the ray generated from the centre of the camera to the pixel centre. This Z map is converted to a depth map and is saved in the same format as the depth information from the Kinect sensor. This allowed the synthetic data set to be passed through the system in the same way as any other data set captured using the Kinect sensor. For generating the data sets, the quality is set to the same standard as the data obtained from the Kinect sensor to simulate data obtained from the Kinect sensor. A brief summary of the synthetic data sets are provided in Table 7.1.

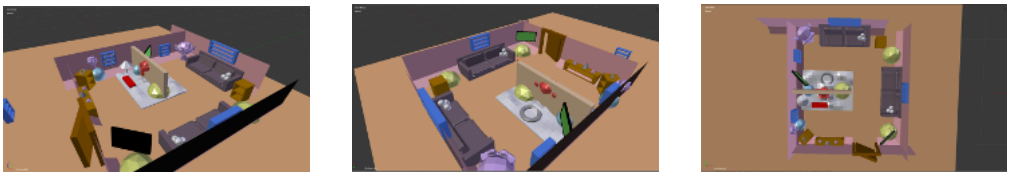
Data set	Scene	Area	Testing purpose
`Translation Test`	Table Top	$2m^2$	Test isolated translation
`Rotation Test`	Table Top	$0m^2$	Test isolated rotation
`Circling Table`	Table Top	$4m^2$	Test combination of translation and rotation
`Moving Around`	Room	$2m^2$	Test combination of translation and rotation

Table 7.1: Properties of the data sets with scenes used and the area that the camera moves through.



(a) (b) (c)

Figure 7.1: Each subfigure illustrates a different view of the `Table Top` Scene.



(a) (b) (c)

Figure 7.2: Each subfigure illustrates a different view of the `Room` Scene.

7.1.1 Synthetic Data Sets

Here the synthetic data sets are described that are used throughout the 3D reconstruction experiments. Two synthetic scenes were generated using Blender and are described as follows:

Table Top : A table with the model of a monkey's head from Blender's built-in models; as well as two torus rings and two hexagons as shown in Figure 7.1. Datasets `Translation Test`, `Rotation Test` and `Circling Table` uses this scene.

Room : A synthetic room reconstruction as shown in Figure 7.2. The `Moving Around` data set uses this scene.

The accuracy of the motion estimation is used to evaluate the system's ability to reconstruct the environment, i.e. the ICP tracking is evaluated. To do this, two data sets were constructed using the `Table Top` Scene, `Translation`

`Test` and `Rotation Test`. The synthetic data sets provide the ground truth translation and quaternions that describe the virtual camera's motion during the capture of the data set. The `Translation Test` data set provides a test that translates the camera through the scene without changing its orientation. Similarly, the `Rotation Test` keeps the camera in a fixed position, and consecutively rotates the camera along each individual principle axis independently. Once the `Translation Test` and `Rotation Test` have demonstrated the system's ability to track the translation and rotation independently, the `Circling Table` data set provides a small-scale test with simultaneous translation and rotation throughout the scene.

The data set `Moving Around` is generated from the `Room` Scene. This data set captures the camera translating and rotating through a virtual room. This provides a simulated test of rotation and translation for a room-sized test more similar to the kind of data encountered in a real-world use case. The rotations that occur within the data sets are illustrated in the camera's local coordinate system while the trajectory of the translation is shown in the world coordinate system. The data sets were captured under the following conditions: The camera was initially aligned with the Y axis in Blender's global coordinate system, with the positive Y axis in the direction of the camera's principal axis. The increasing X axis corresponds to the camera moving right. The increasing Z axis corresponds to the camera moving upwards. The estimated trajectories have been transformed from the camera's coordinate system to Blender's global coordinate system. The differences in the coordinate systems can be seen in Figure 7.3. All the synthetic data sets are passed through our variant of the Kinect Fusion system with the known camera parameters (f_x, f_y, o_x, o_y) = (588.81, 588.81, 320.97, 239.5) and the volume size set to 5.12m for each axis.

Next, a short description of each of the synthetic data sets is provided.

Translation Test

This test from the `Table Top` Scene consists of the camera moving along the global X, Y and Z axes in turn with no rotation performed. This test starts with the camera located at the origin and ends with the camera located at origin. The camera initially moves 0.75 meters forward in the direction of the increasing global Y axis and then returns to the original position. This is followed by moving 0.75 meters downward in the direction of the decreasing global Z axis and returning to the original position. This is followed by 0.75 meters upwards in the direction of the increasing global Z axis and returns to the original position. Finally, the camera moves 0.75 meters right (in the direction of the increasing global X axis) and returns to the original position followed by moving 0.75 meters left along the negative X axis. Where the simulation stops. For the Z and Y axes, it uses 50 frames to cover 0.75 meters

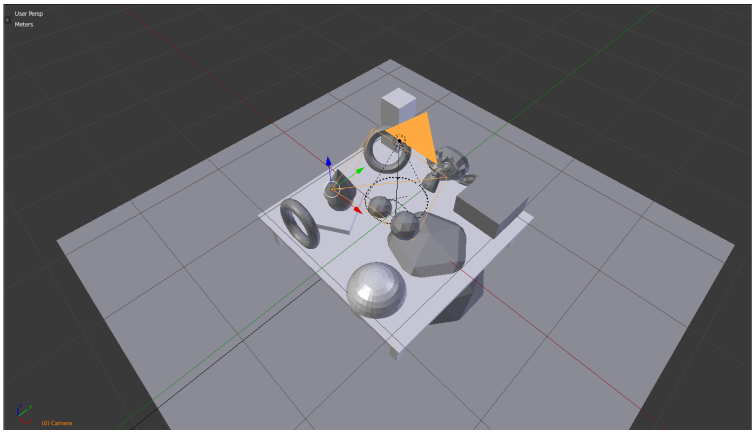


Figure 7.3: An illustration of the global coordinate system in Blender. The red axis illustrates the global X axis, the green axis illustrates the global Y axis and the blue axis illustrates the global Z axis.

but for the X axis it uses 100 frames to cover this distance.

Rotation Test

For this test, the camera performed a series of rotations without translation in the following order: roll (camera's Z axis), yaw (camera's Y axis), and then pitch (camera's X axis). The roll was performed between frames 0 and 400, then yaw between frames 401 and 800, and then pitch between frames 801 and 1000. The roll and yaw each perform a rotation of 90° counter-clockwise and clockwise to return to the original orientation then another rotation of 90° clockwise then counter-clockwise was perform to return the camera to the original orientation. The pitch performs a 45° rotation counter-clockwise and then a 45° rotation clockwise to return to its original orientation. Frames of this data set are illustrated in Figure 7.4.

Circling Table

The next test from the `Table Top` Scene, labelled the `Circling Table`, provides a test for simultaneous rotation and translation of the camera. This test uses 600 frames, with the camera starting at the origin and initially moving to the start of a circular path while maintaining its orientation towards the centre of the scene. While moving to the start of the circular path the camera incrementally performs a pitch rotation equating to 25° degrees. Once the camera has reached the start of the circular path, it begins moving along the path while maintaining the camera's orientation fixed on the centre of the table. Frames from this data set are illustrated in Figure 7.5.

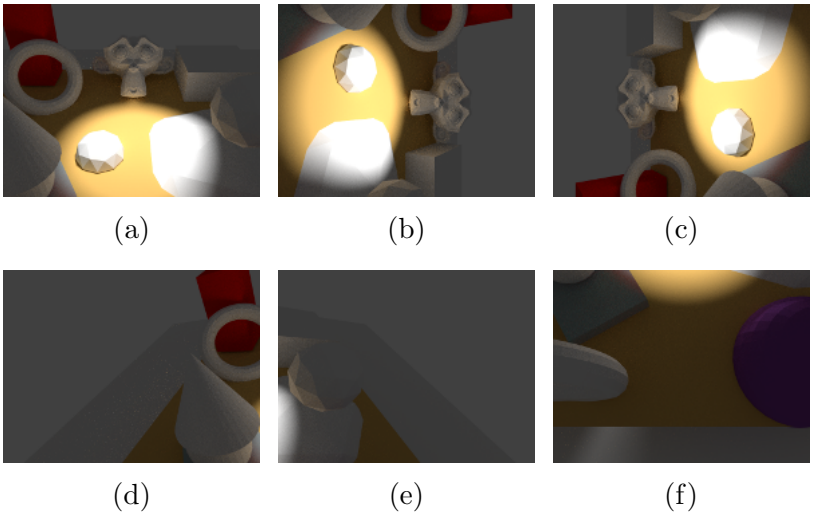


Figure 7.4: An illustration of frames from the `Rotation Test` data set in the order they occur in the data set. Subfigure (a) illustrates the initial position of the camera, before any rotations occur. Subfigure (b) illustrates the first rotation 90° counter-clockwise Subfigure (c) illustrates the second rotation, 90° to the original orientation and then another 90° to the orientation shown. The camera then gets back to its original orientation. Subfigure (d) and (e) illustrates a 45° rotation on the yaw axis from the original orientation. Subfigure (f) illustrates the last orientation, a rotation on the pitch axis.

Moving Around

`Moving Around` uses the `Room` Scene to provide synthetic data of a room. The camera performs simultaneous translation in the X and Y axes while there is no movement on the Z axis. This also contains rotations of the camera along each axis throughout the scene. This test maps an area of $4m^2$ using 2000 frames. The data set can be seen in Figure 7.2.

7.1.2 Real-world Data Sets

Due to the frame rate requirements and volume restrictions, the data sets published by Sturm, Engelhard, Endres, Burgard and Cremers (2012) produced results that do not represent the environment. This is due to the data sets having fast motion of the camera: the assumption the ICP algorithm makes is that the motion between successive frames are small. Thus the proposed system is sensitive to fast motion of the camera. For this system, three data sets were captured using a Kinect sensor recorded moving slowly to prevent blurring of the images and to ensure small motion between each frame. The data sets are named `Room Closed Loop`, `Desk`, and `Lab Room`. These data sets

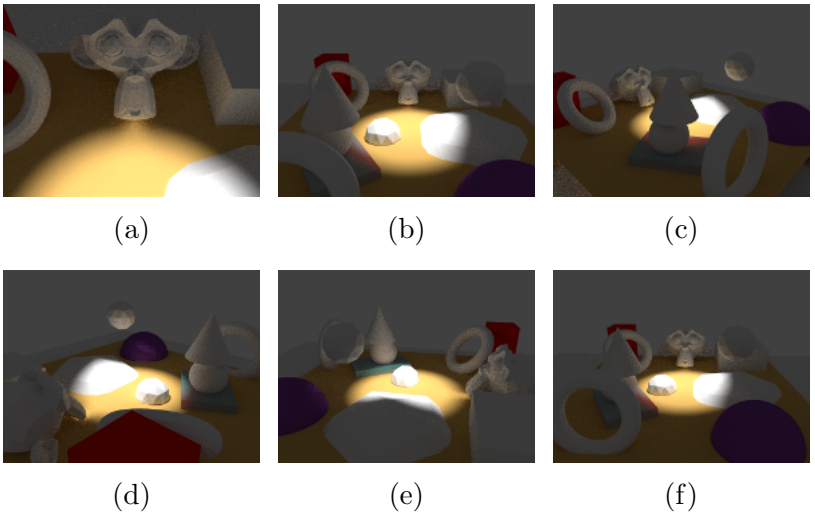


Figure 7.5: An illustration of frames from the `Circling Table` data set. Subfigure (a) shows a frame from the start position. Subfigure (b) illustrates a frame after the camera moves backwards and pitches upwards. After it has reached the frame illustrated in Subfigure (b), the camera starts its circular path around the table, remaining directed at the centre of the table. Subfigures (c) - (f) illustrate frames along the circular path.

are briefly described below.

Room Closed Loop

The `Room Closed Loop` data set is recorded to test typical real-world use of the proposed system. The data set is of a small bedroom of approximate floor size $4m^2$ over 2000 frames. The camera moves in a small circle, performing a 360° rotation along the yaw axis. An overview of this data set is illustrated in Figure 7.6.

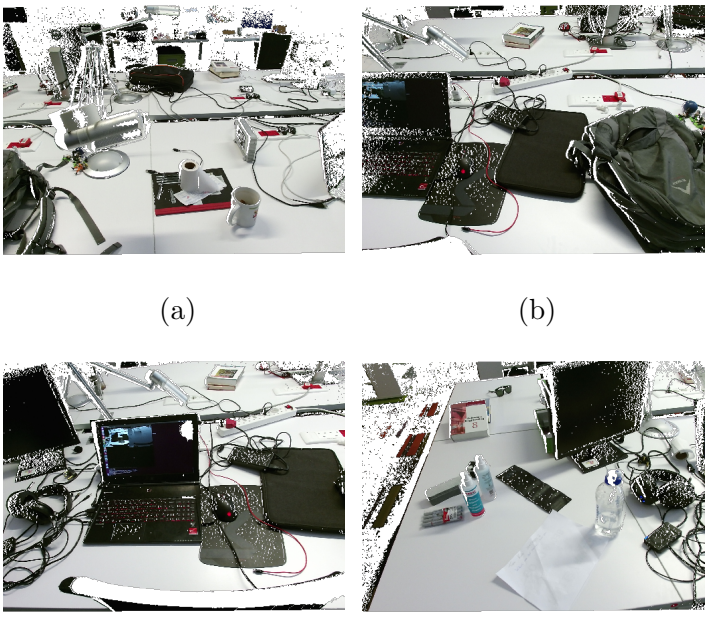
Desk

In this data set, a desk is capture in a lab environment. The camera starts on the right side of the desk and is manually moved approximately 2 meters to the left of its starting position, while capturing the desk. This data set does not perform much rotation during the image sequence and consists of 900 frames.

Lab Room

The `Lab Room` data set was recorded in a rest area of a lab. The data set covers a floor area of approximately $4m^2$. This data set passes through an

Figure 7.6: An overview of the `Room Closed Loop` data set during recording.



(a) (b)

(c) (d)

Figure 7.7: Frames from the `Desk` data set during recording.

(a)

(b)

Figure 7.8: An overview of the `Lab Room` data set.

area capturing a lounge, a TV and a chess set. The data set contains 2000 frames. Sample frames from the data set is illustrated in Figure 7.8.

7.2 Trajectory Analysis

In order to evaluate the accuracy of the system, the camera trajectory estimated by the system is analysed. The deviation of the reconstructed trajectory from the ground truth trajectory is considered. Specifically, the analysis investigates the translation error along each axis, the rotation errors for each axis and the dot product between the ground truth quaternion and the estimated quaternion. Given the estimated transformation matrix, $\hat{T}_i$, and the ground truth transformation matrix T_i, the relative transformation matrix is computed as $\hat{T}_i^{-1}T_i$. The translation obtained from this calculation is used as the error for the translation component while the rotation matrix from the

Component	Average error
Roll error in radians	0.00458
Pitch error in radians	0.00863
Yaw error in radians	0.00618
X Translation in meters	0.02627
Y Translation in meters	0.01702
Z Translation in meters	0.00758

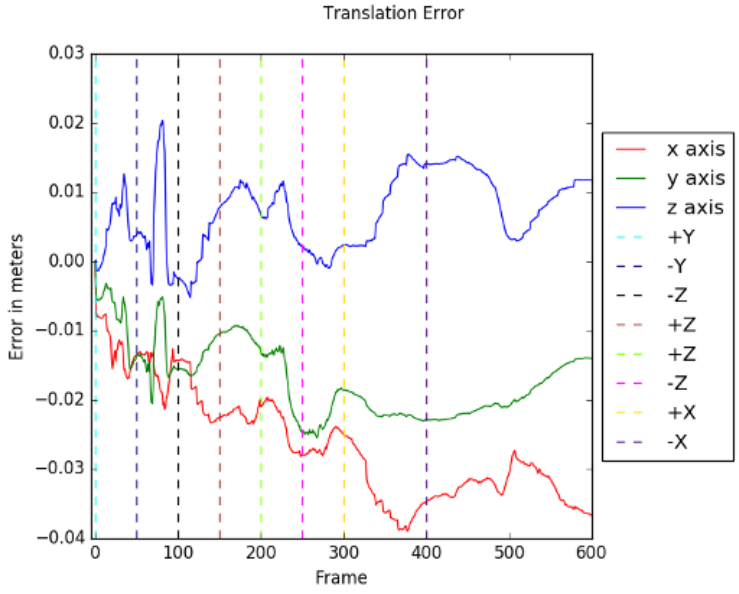
Table 7.2: The magnitude of the error averaged over all frames of each translation and rotation component for the `Translation Test` data set.

relative transformation matrix is converted to Euler angles for each axis and represents the rotational error for each axis. The translational errors are measured in meters, and the rotational errors are measured in radians. The error displayed in the graphs in subsequent subsections are used to demonstrate the system's performance and are computed as follows.

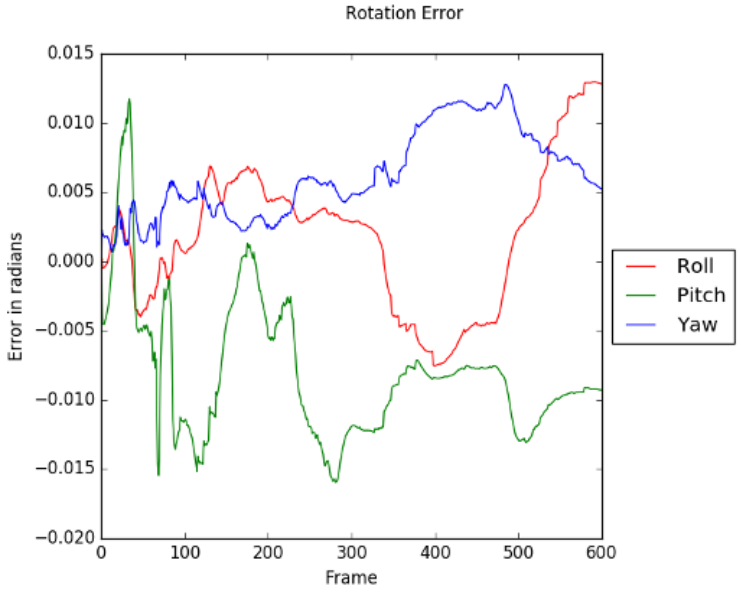
7.2.1 Translation Test

In the analysis of the `Translation Test`, zero change in the quaternion vector and Euler angles error is expected, due to no rotations occurring in this experiment. This experiment is expected to solely report changes in the translation vector. Figure 7.9 graphs the relative error for the translation in meters and rotation in radians. The ground truth is graphed against the estimated position of each frame in Figure 7.10. The absolute ground truth rotation is graphed against the estimated rotation in Figure 7.11.

The results from Figure 7.10 indicate that the motion estimation of the system has followed the ground truth's motion. This indicates that the motion estimation has performed adequately in estimating the camera's position throughout the scene. The translation error seen fluctuates through the movement with most of the error accumulating in the X and Z axis. A maximum deviation of about 0.4 meters in the X axis is observed. Since the system is based on ICP using the depth data from the Kinect, it is expected to obtain a minimal error on the Y axis (the direction the camera is facing). This is due to the system receiving accurate information about the camera moving along the Y axis from the depth information. However, the system estimates the X and Z motion indirectly. Since the trajectory in this data set has no rotation, it is expected that the estimated rotation should not change. However, estimated rotation does deviate from the ground truth. Analysing the rotation error, it is observed that the errors on each of the axes are below 0.02 radians. This gives a clear indication that there is no significant error in the rotational estimate for each of these axes. This conclusion is supported by comparing the dot

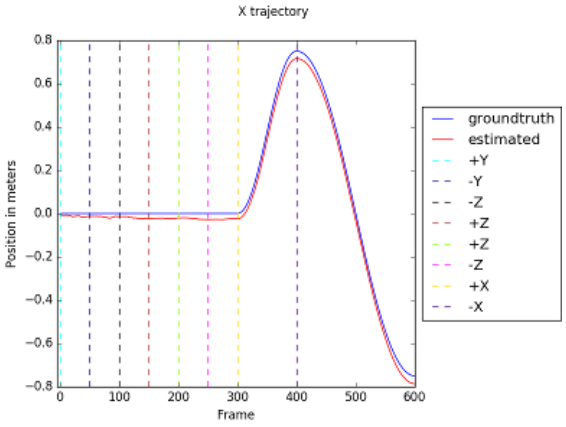


(a)

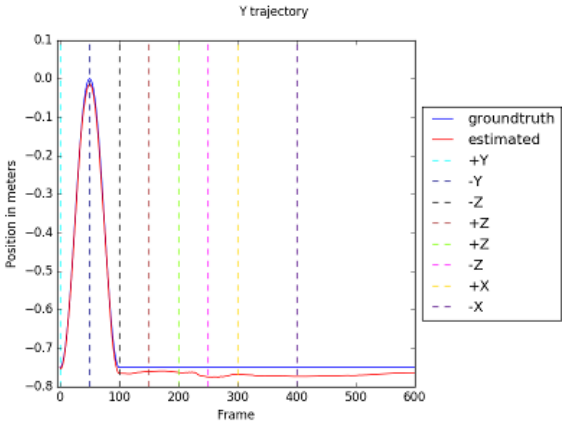


(b)

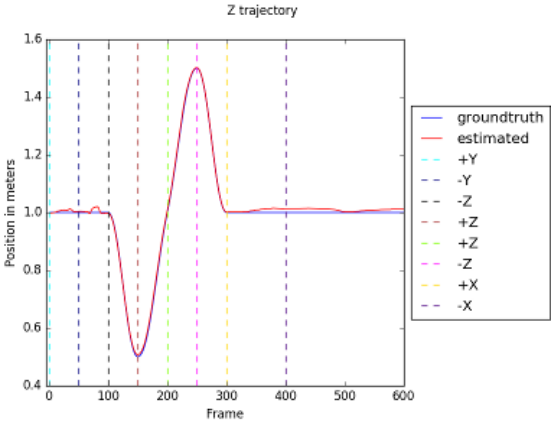
Figure 7.9: Translation error for the X, Y and Z axis and rotation error for the roll, pitch and yaw of the camera are illustrated for the `Translation Test` data set. The graphs assign the horizontal axis as the frame number and the vertical axis as the error. In this test the camera moves along the Y axis for the first 200 frames, followed by movement along the Z axis for 200 frames and finally, moving along the X axis for 200 frames. Subfigure (a) illustrates the translation error while Subfigure (b) illustrates the rotation error for the roll, pitch and yaw. Key points in the graph are marked with vertical dashed lines to indicate a change in motion or points where the camera has returned to its original orientation or position. The translation or rotation shown in the legend indicates the start or continuation of the specified motion.



(a)

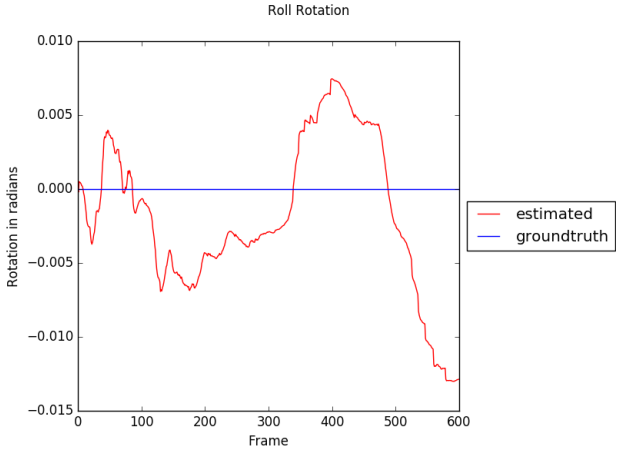


(b)

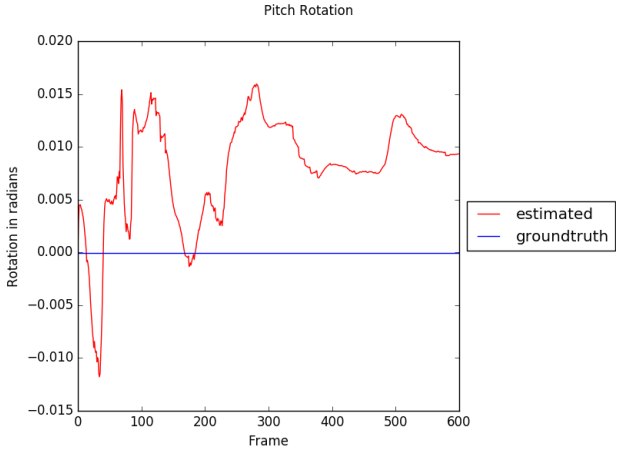


(c)

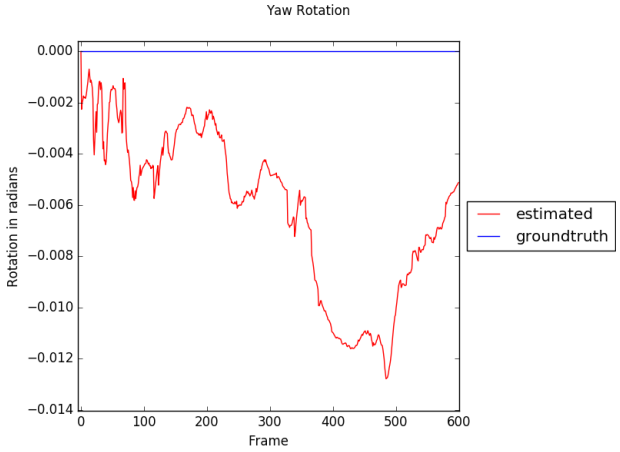
Figure 7.10: The results for the `Translation Test` (Note different scales on Y-axis). Subfigure (a) shows the estimated and ground truth motion along the X axis. Subfigure (b) shows the estimated and ground truth motion along the Y axis. Subfigure (c) shows the estimated and ground truth motion along the Z axis. Key points in the graph are marked with vertical dashed lines to indicate a change in motion or points where the camera has returned to its original orientation or position. The translation or rotation shown in the legend indicates the start or continuation of the specified motion.



(a)



(b)



(c)

Figure 7.11: The results for the `Translation Test`. Subfigure (a) shows the rotation of the estimated rotation to the ground truth rotation that occurred along the roll axis. Subfigure (b) shows the rotation of the estimated rotation to the ground truth that occurred along the pitch axis. Subfigure (c) shows the rotation of the estimated rotation to the ground truth that occurred along the yaw axis.

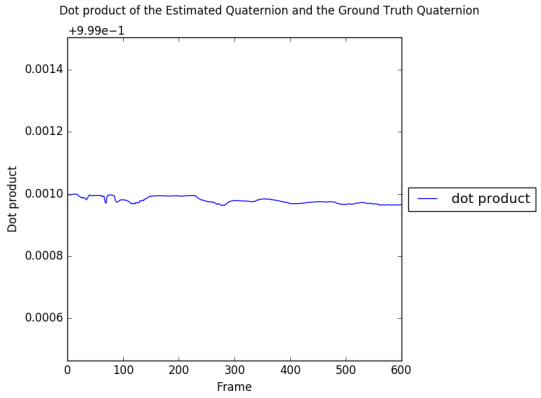


Figure 7.12: The dot product between the ground truth quaternion and estimated quaternion for the `Translation Test`.

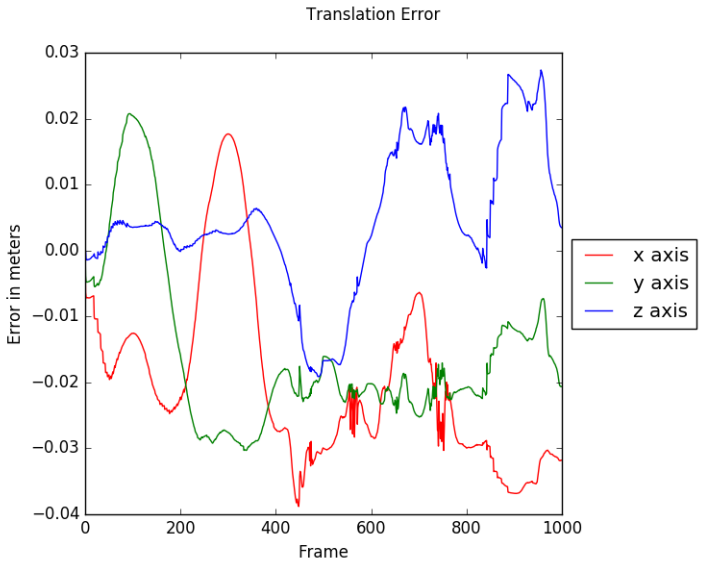
product between the ground truth quaternion and the estimated quaternion in Figure 7.12. The quaternions are unit vectors that represent the rotation of the camera and the dot product can be used to obtain the cosine of the angle between the two vectors. With the results of the dot product close to 1, the vectors are shown to be very similar. Table 7.2 summarises the error in translation and rotation estimates during the frame sequence by presenting the average error for each axis.

7.2.2 Rotation Test

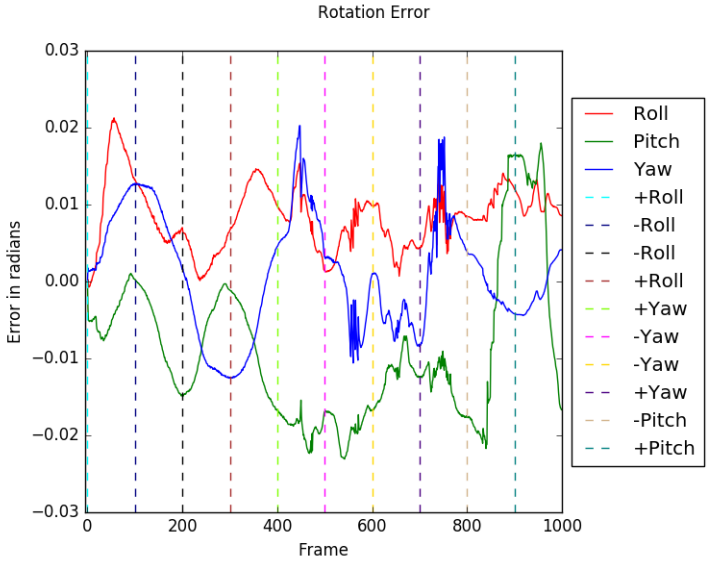
The `Translation Test` data set was used to verify that the system is able to estimate the translation in isolation. The ability of the system to estimate rotation in isolation is tested next using the `Rotation Test` data set.

From the description in Section 7.1.1, the highest rotational errors for the roll axis is expected between frames 0 to 400, for the yaw axis between 401 and 800 and the pitch axis during 801 and 1000. In addition, very low translational errors are expected during the full frame sequence. Examining the rotational errors, illustrated in Figure 7.13 (b), a sudden increase in error occurs on the roll axis at the start of the experiment as the camera rotates on the roll axis to reach the maximum 90 degree counter-clockwise rotation at frame 100. As the camera rotates back to its original orientation, the error in the roll axis decreases until it gets closer to 0 at frame 200. This is in accordance with the expected results for the first 200 frames. However, the pitch and yaw axis also see an increase in error, although not as significant relative to the roll axis.

The camera then proceeds to rotate along the roll axis again in the clockwise direction. A sudden rise and then a fall between frames 200 and 400 is expected for the roll axis and is observed. However, the decrease in the rotational error is not as great as expected. After the roll rotation has been



(a)



(b)

Figure 7.13: Translation error for the X, Y and Z axis and rotation error for the roll, pitch and yaw of the camera are illustrated for the `Rotation Test` data set. The graphs use the horizontal axis as the frame iteration and the vertical axis as the error. Subfigure (a) illustrates the translation error while Subfigure (b) illustrates the rotation error for the roll, pitch and yaw. Key points in the graph are marked with vertical dashed lines to indicate a change in motion or points where the camera has returned to its original orientation or position. The translation or rotation shown in the legend indicates the start or continuation of the specified motion.

performed the error is approximately 0.01 radians. From frames 400 to 1000 no additional rotation is expected on the roll axis and minimal fluctuations are expected around its already accumulated error of 0.01 radians. This is observed throughout the rest of the graph, with the roll axis maintaining its last error of 0.01 fluctuating between 0.007 and 0.013.

Next the yaw rotational trajectory is examined. During frames 0 - 400 no rotation occurs on the yaw axis followed by a 45° rotation counter-clockwise, a 45° rotation clockwise to return the camera to its original orientation, a 45° rotation clockwise, and then the last 45° rotation counter-clockwise to return to its original orientation. No further rotation is performed on the yaw axis. For this data set it is expected that minimal errors should occur during frames 0 - 400 and 801 - 1000 and most of the error to accumulate during the rotation on yaw axis between frames 401 and 800. However, for this data set the error during frames 0 to 400 appear to be similar to the error that occurred during the roll. Since the system is estimating the rotation it is expected that some rotational components would show around the other axes on a smaller scale. At the end of the roll, the yaw axis has an error just below 0. Between frames 401 and 800 the error changes in nature, from a simple oscillation to a graph with a series of sudden spike in an oscillating pattern. This is clearly due to the rotation performed during these frames. Following the rotational motion, after frames 801 when the camera has stopped rotating along the yaw axis, the rotation begins to follow an oscillating pattern as seen between frames 0 and 400.

Next the pitch is examined, it is expected to have minimal errors throughout frames 0 to 800 in which no rotation occurred along the pitch axis. Following this, a 90° counter-clockwise rotation occurred along the pitch axis and an additional 90° clockwise rotation occurred to orientate the camera to its original orientation. The error along the pitch axis initially follows a similar trend to the yaw axis, with an oscillating error between frames 0 and 400. After frame 400, the axis error is maintained and decreases slightly as it approaches frame 800. After frame 800 the error drastically increases and decreases in accordance to the pitch rotation that has occurred during these frames. This is the expected result of the rotation.

In this experiment, no translation was applied at any point, so minimal errors are expected in the estimates of the camera location. Examining the resulting graph, oscillation did however occur in the translational error along all axes. For example, between frames 0 and 400 while the rotation is being applied about the roll axis, which results in oscillating translation errors in the X and Y axis. This appears again during frames 800 and 1000 with the pitch rotation. These errors have a maximum value of 0.03 meters which is negligible.

The results shown in Figure 7.15 indicate that the motion estimation is working. This is supported by Figure 7.14 illustrating the dot product between

Component	Average error
Roll error in radians	0.00842
Pitch error in radians	0.01167
Yaw error in radians	0.00631
X Translation in meters	0.02218
Y Translation in meters	0.01889
Z Translation in meters	0.00882

Table 7.3: The magnitude of the error averaged over all frames of each translation and rotation component for the `Rotation Test` data set.

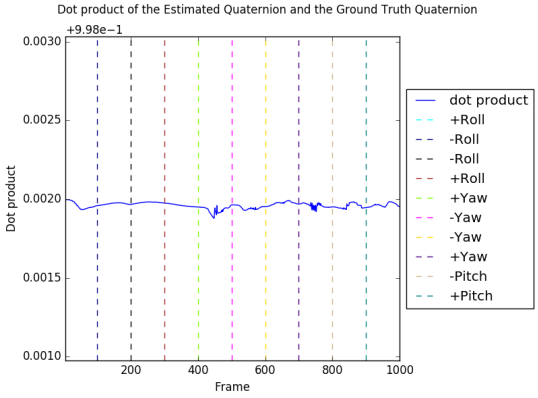


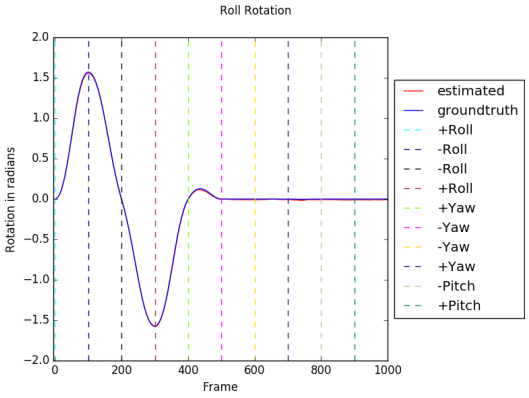
Figure 7.14: The results for the `Rotation Test`. Dot product between the ground truth's quaternion and estimated quaternion. Key points in the graph are marked with vertical dashed lines to indicate a change in motion or points where the camera has returned to its original orientation or position. The translation or rotation shown in the legend indicates the start or continuation of the specified motion.

the estimated rotation and the ground truth rotation; however, it is being affected by the accumulated error over the frame sequence. This is illustrated in the Figure 7.16 as the translational error along each axis increases and does not return to the ground truth as the camera remains static. Table 7.3 summarises the error in translation and rotation estimates during the frame sequence by presenting the average errors with respect to each axis.

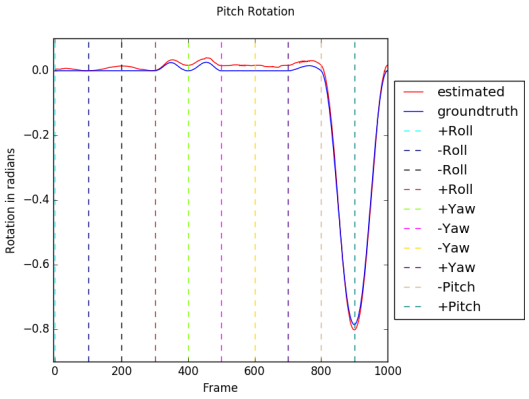
Next a combination of rotation and translation is performed using the `Circling Table` data set.

7.2.3 Circling Table

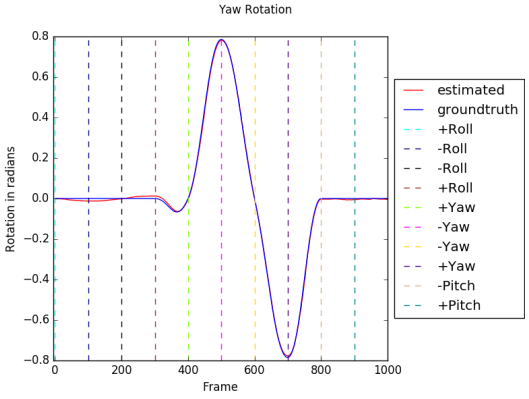
The translation and rotational errors resulting from applying our 3D reconstruction system to the `Circling Table` data set are shown in Figure 7.17.



(a)

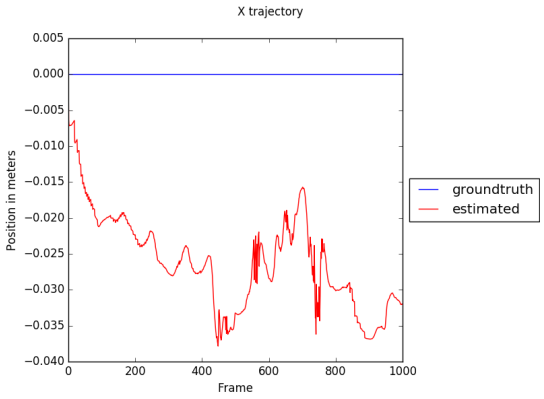


(b)

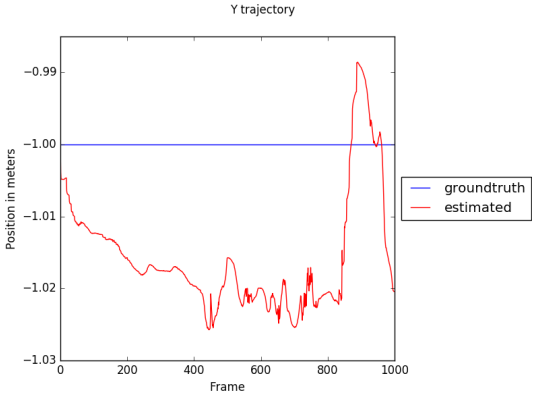


(c)

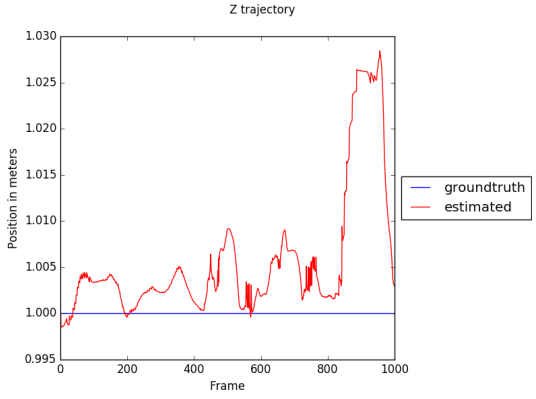
Figure 7.15: The results for the `Rotation Test`. Subfigure (a) shows the estimated rotation and the ground truth rotation that occurred along the roll axis. Subfigure (b) shows the estimated rotation and the ground truth that occurred along the pitch axis. Subfigure (c) shows the estimated rotation and the ground truth that occurred along the yaw axis. Key points in the graph are marked with vertical dashed lines to indicate a change in motion or points where the camera has returned to its original orientation or position. The translation or rotation shown in the legend indicates the start or continuation of the specified motion.



(a)



(b)



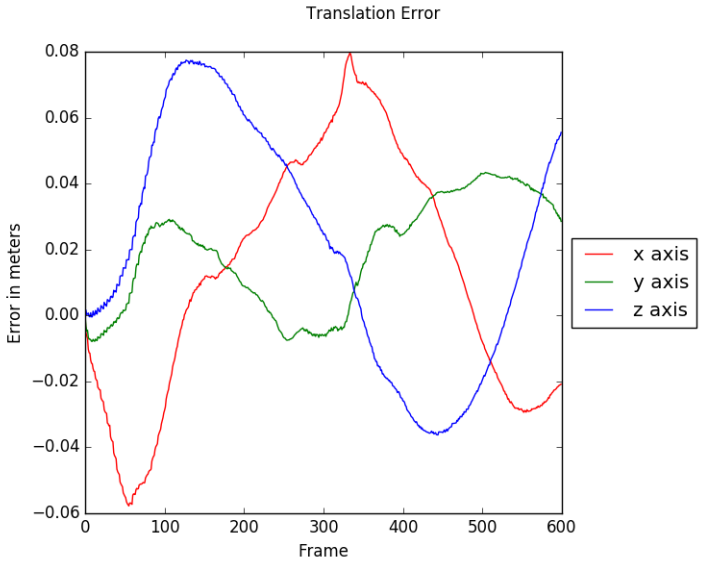
(c)

Figure 7.16: The results for the `Rotation Test`. Subfigure (a) shows estimated and ground truth motion along the X axis. Subfigure (b) shows estimated and ground truth motion along the Y axis. Subfigure (c) shows estimated and ground truth motion along the Z axis.

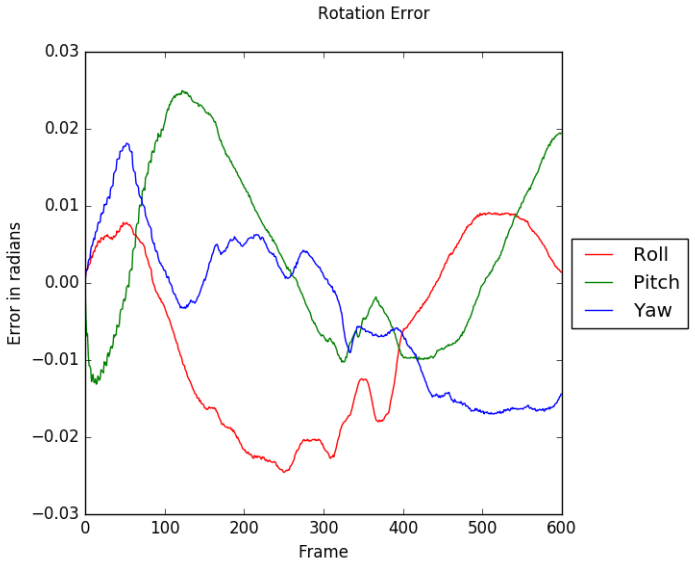
The trajectory is illustrated in Figure 7.18 and the absolute rotation is illustrated in Figure 7.19. Initially, errors for translation and rotation are accumulated during the first 100 frames. During this time, the camera is moving away from the centre of the scene to where the circular path starts, while rotating about the pitch axis. The translation and rotation has sharp increase in the error as the camera moves from the centre of the scene to the start of the path due to the camera covering 1 meter over 100 frames, which is faster than in the rest of the sequence: at a frame rate of 60Hz, this motion corresponds to approximately $1.6\text{m}\,\text{s}^{-1}$. As the camera starts moving more slowly, when it starts circling the table, the ICP is capable of reducing the initial translation errors accumulated during the first 100 frames, as can be seen in Figure 7.18. The translation error slowly follows an oscillating pattern as it circles the table with changes in the X and Y axis. During this motion, the Z axis has no translation and minimal error is expected. It is observed that the Z axis reaches a maximum error of under 0.08m. This experiment illustrates the motion estimators ability to simultaneously estimate the rotation and translation, as can be seen in Figures 7.18 and 7.19. The errors produced in the estimated trajectory are relatively small and insignificant in comparison to the motion achieved in the data set.

7.2.4 Moving Around

The final experiment performs a test of the system that aims to reconstruct a room of about 4m^2, created using the `Room` scene, as illustrated in Figure 7.2. Figure 7.20 shows the translation and rotational errors during the reconstruction process for this experiment. This demonstrates the error in the rotation and translation, showing a maximum translational error of 0.3 meters while the rotational error has a maximum of about 0.15 radians. Figures 7.21 and 7.22 illustrate the absolute trajectory and rotation of the estimated camera and the ground truth. From these figures it is observed that the estimated X and Y axis trajectories are similar to the ground truth trajectories, although it is observed that over time the trajectories slowly drift from each other. This is due to compounded errors resulting in trajectory drift, however the system partly recover from some of the drift over time. The Z trajectory is where the largest error is observed. As shown in Figure 7.20, despite there being hardly any ground truth movement along this axis, the estimated trajectory has deviated around 20 cm towards the end of the sequence. As for the rotational trajectory in Figure 7.21, the estimated camera rotation tracks the ground truth rotation closely. This clearly indicates the system has the ability to estimate the trajectory of the camera with its error related to the length of the sequence. The point cloud generated in this experiment will be used in Section 7.4 as a synthetic data set for the mesh reconstruction algorithm.

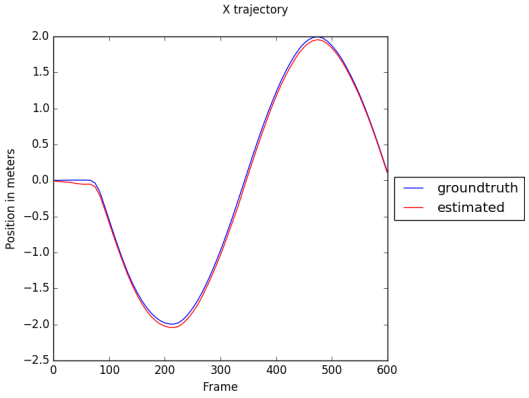


(a)

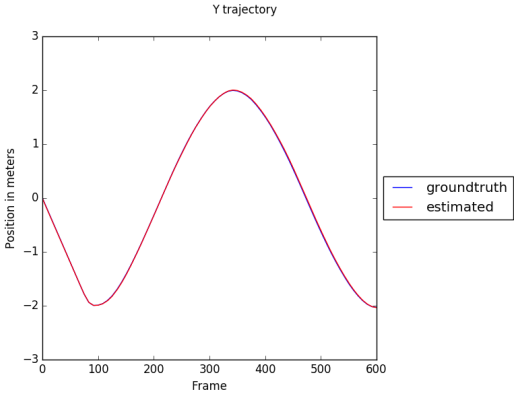


(b)

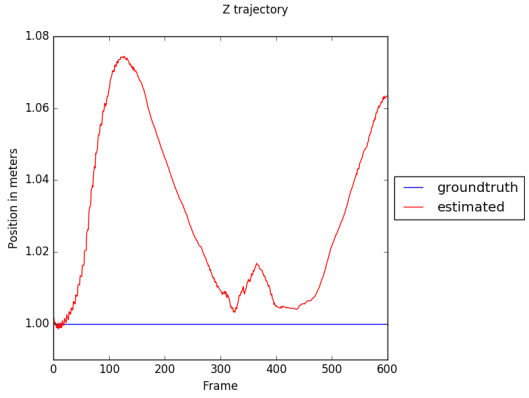
Figure 7.17: Translation error for the X, Y and Z axes and rotation error for the roll, pitch and yaw axes of the camera for the `Circling Table` data set.



(a)

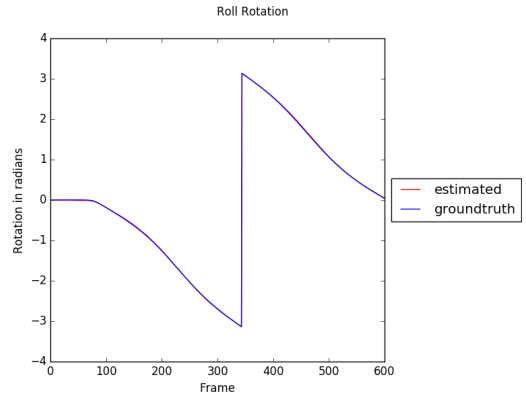


(b)

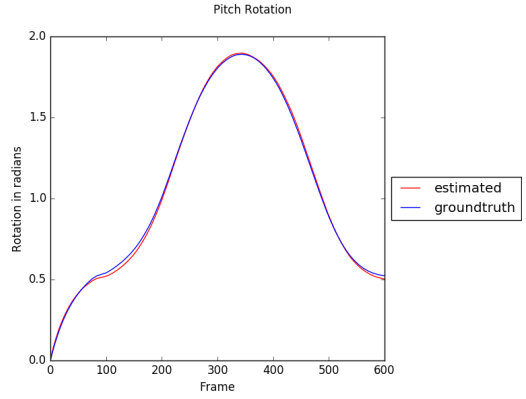


(c)

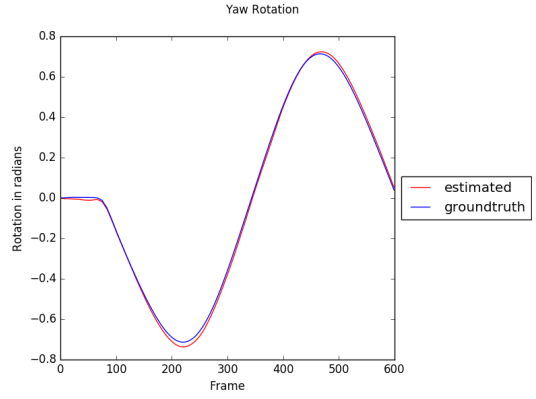
Figure 7.18: Illustrates the results for the `Circling Table` data set. Subfigures (a) - (c) shows the estimated ground truth motion along the X, Y and Z axis respectively. Note the difference in scale between each graph.



(a)

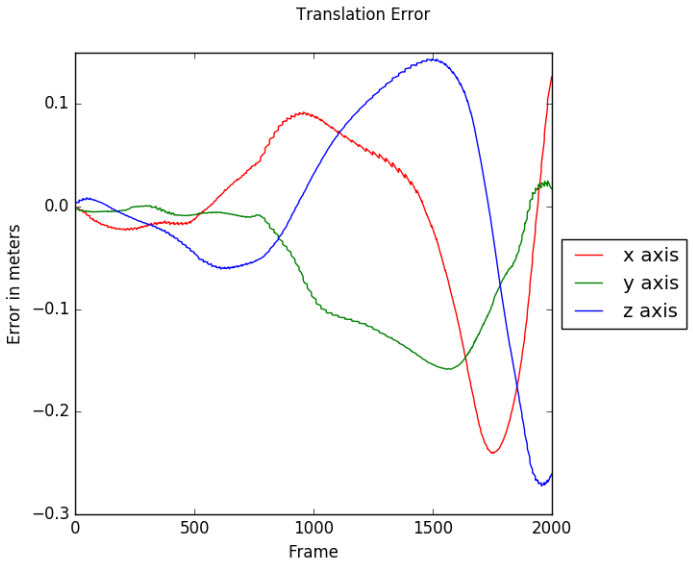


(b)

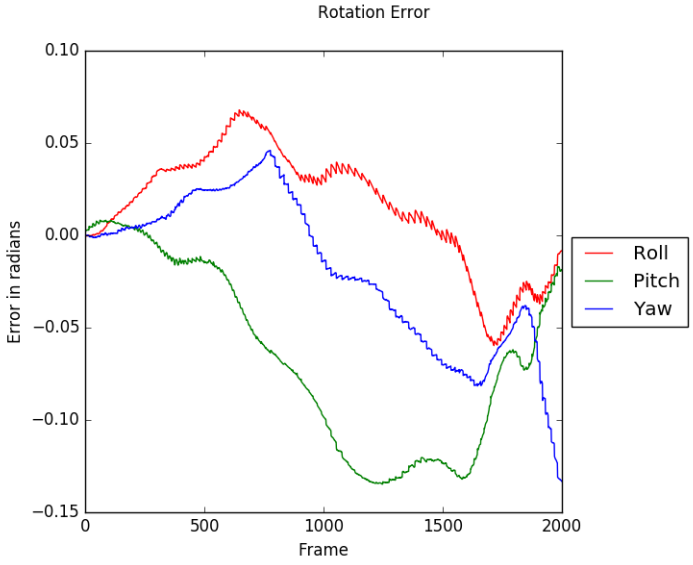


(c)

Figure 7.19: Illustrates the results for the absolute rotation for the `Circling Table`. The values on the Y axis are based on the interval between the ranges $[-\pi, \pi]$. Subfigures (a) - (c) show the estimated rotation and the ground truth rotation along the roll, pitch and yaw axis respectively.

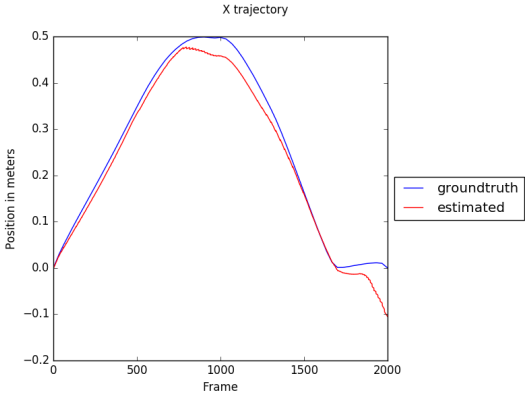


(a)

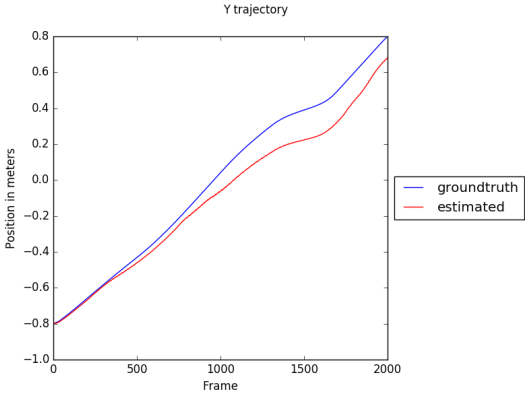


(b)

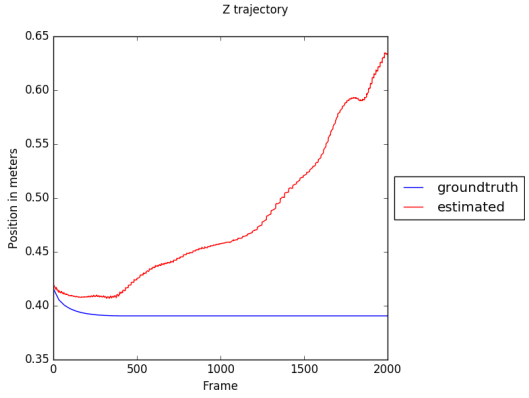
Figure 7.20: Translation error for the X, Y and Z axes (Subfigure (a)) and rotation error for the roll, pitch and yaw of the camera (Subfigure (b)) for the `Moving Around` data set.



(a)

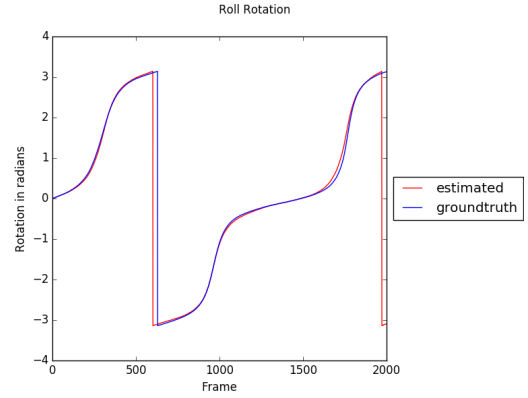


(b)

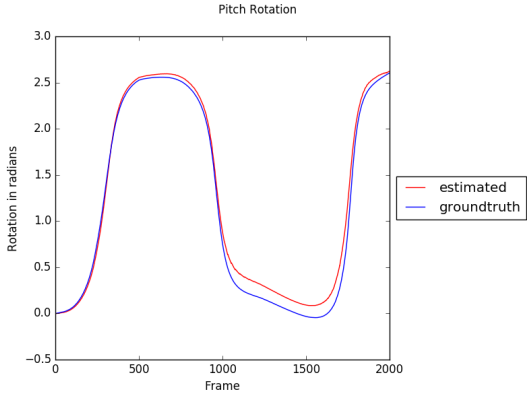


(c)

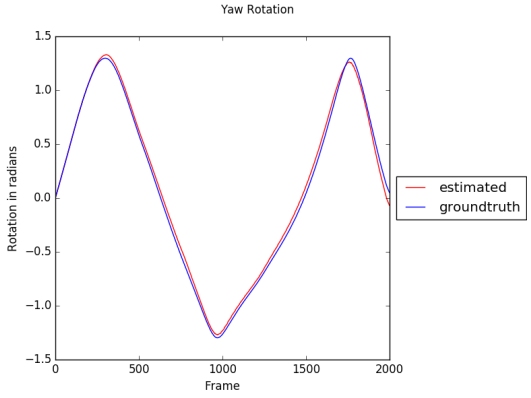
Figure 7.21: Illustrates results for the `Moving Around` data set. Subfigures (a) - (c) show estimated motion along the X, Y and Z axis respectively.



(a)



(b)



(c)

Figure 7.22: Illustrates results for the `Moving Around` data set.
Subfigures (a) - (c) show the estimated rotation that occurred along the roll, pitch and yaw axis respectively.

7.2.5 Additional Experiments

An additional experiment was performed when attempting to incorporate more accurate trajectory estimation techniques. Systems like that in Whelan and Kaess (2012) moved away from the ICP-based motion estimation, instead opting for a motion estimation system based on Huang and Bachrach (2017). An attempt to mimic this was made by modifying our system to use feature-based estimation. This uses the incoming RGB image and the RGB image extracted from the ray caster, but maintaining the frame-to-model approach. This approach could potentially provide better results because the feature matcher can infer better point correspondences than the current ICP approach. This was implemented using the ICP technique by Besl and McKay (1992). The system was unable to produce adequate results due to the pixelated appearance of the ray caster's RGB image and the feature detectors not working well on the RGB image. The system was unable to produce good key points or enough matches between the incoming image and the extracted RGB image. Pixelation occurs because the colour data is stored in a 3D volume, and the ray caster image resolution is limited by that of the 3D volume. This pixelation proved to be too much for the feature detectors to function well.

7.2.6 Conclusion

This section detailed experiments performed to test various aspects of the system's ability to estimate the camera's motion. The data set `Translation Test` aimed to test the motion estimator's ability to estimate the translation components. This data set featured a camera being translated along the X, Y and Z axes in isolation. It was shown that the system has the ability to estimate the translation along each axis with a reasonable degree of accuracy. The `Translation Test` was followed by the `Rotation Test` that tested the system's ability to estimate the camera's rotation. This consisted of rotating the camera along the roll, pitch and yaw axes in isolation. The system was shown to be capable of estimating the rotation about each axis with a reasonable degree of accuracy. This enabled the next experiment, using the `Circling Table` data set. The `Circling Table` data set allowed simultaneous rotation and translation to be estimated. The system proved capable of estimating a combination of different rotations and translation simultaneously. This suggests that the system is capable of estimating combined translation and rotation. The final data set that the motion estimation was performed on was the `Moving Around` data set, representing a synthetic room. Analysis of this experiment showed that the system has the ability to estimate the camera's pose over a long sequence of frames; however, the estimated trajectory drifts from the ground truth.

In the next section, we discuss the results of various techniques used to

(a) (b)

Figure 7.23: Test data that is used to illustrate the different point cloud extraction techniques. Subfigure (a) is the depth map that shows a depth measurement of 1 meter in the outer rectangle of the image and a 2 meter measurement for the inner rectangle. Subfigure (b) illustrates the associated RGB image that uses blue to represent the area that is 1 meter away from the sensor and white to show the area that is 2 meters away from the sensor.

extract the point cloud from the TSDF volume. This includes analysis of the point clouds that are generated from the synthetic data sets as well as a selection of real-world data sets that were recorded with the Kinect.

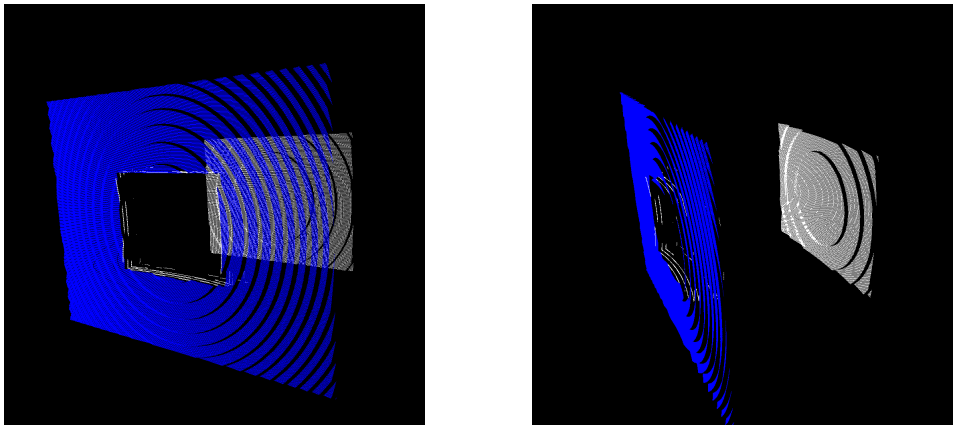
7.3 Point Cloud Analysis

This section investigates different methods of point cloud extraction and hole-filling, as presented in Chapter 5. This section analyses points clouds generated from the synthetic tests described above, as well as a few real-world data sets that were recorded with the Kinect sensor. The images presented in this section are best viewed electronically to allow zooming to observe finer detail.

7.3.1 Hole-Filling

The ray caster used for the ICP algorithm is also used to extract the point cloud in the original Kinect Fusion system. However, this is replaced with an orthogonal ray caster as described in Whelan and Kaess (2012) to provide some level of hole-filling of the extracted point cloud, the techniques were discussed in Section 5.7. A synthetic depth image and RGB image (Figure 7.23) is used to demonstrate the effects of these techniques.

The test depth and RGB frames shown in Figure 7.23 are input into the Kinect system using the ray caster as the point cloud extraction technique and then subsequently extracted as a point cloud. The results of using the original point cloud extraction technique are shown in Figure 7.24. Here it is



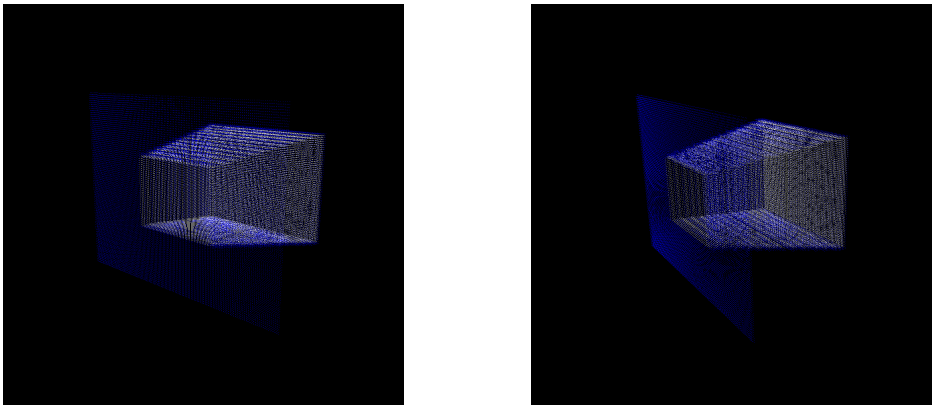
(a) (b)

Figure 7.24: Subfigures (a) and (b) show the extracted point cloud using the original ray casting technique from different angles.

observed that the points extracted illustrate the two planes of the test data. These are the points of the zero-crossings along the line of sight of each pixel from the camera. In Figure 7.25 the results of using the orthogonal ray caster are given. This results in the same two planes with four additional connecting planes. Since all the TSDF values behind the blue plane from the camera's line of sight are negative and all the TSDF values in front of the white plane in the line of sight of the camera are positive, there are zero-crossings that lie between these two volumes. These results are expected using the orthogonal ray caster and were not encountered during the original ray casting process. This illustrates the ability of the orthogonal ray caster to provide a basic level of hole-filling for the point cloud.

The orthogonal ray casting technique provides an additional set of points that fill key areas in the extracted TSDF volume. This provides additional information about the surface in the point cloud compared to the original ray casting technique.

After experimenting with this approach, an issue occurred that prevented depth data of an area from being integrated from multiple directions. This adversely affected the generation of the point cloud using the original ray caster and the orthogonal ray caster. This is demonstrated in an experiment that sets up the camera in two locations as shown in Figure 7.26. Both cameras integrate the depth and RGB image into the volume at each position. This could occur when the camera moves from one side of an object or a wall to another side of the object or wall. Figure 7.27 illustrates a cross-section of the TSDF volume and shows the result of these integrations. This illustrates that the integration of the depth data will override previous data that lies behind a surface with negative values. The figure illustrates that although *Camera*



(a) (b)

Figure 7.25: Subfigures (a) and (b) show the extracted point cloud using the orthogonal ray casting technique. An illustration of this technique's ability to provide hole-filling.

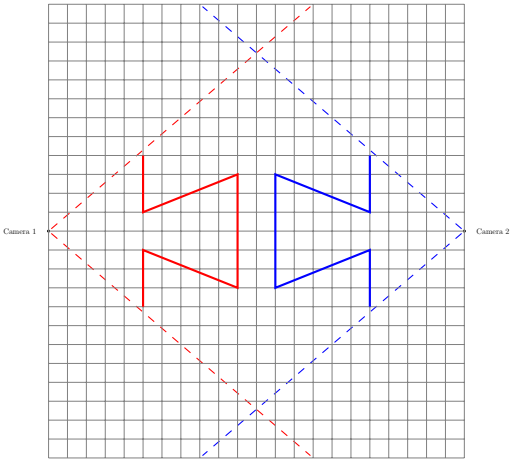
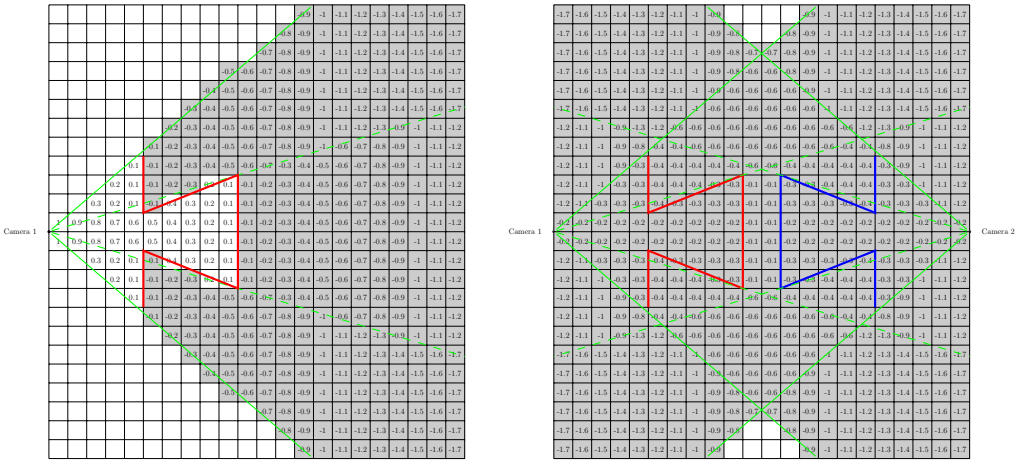


Figure 7.26: In this experiment a camera is simulated at location *Camera 1* and a second camera is simulated at *Camera 2*. The cameras are facing each other. The dashed red lines indicate the field of view of *Camera 1* and the red surface illustrates the depth data that should be integrated into the TSDF volume from *Camera 1*. Similarly, the dashed blue lines indicate the field of view at *Camera 2* and the blue surface illustrates the depth data that should be integrated into the TSDF volume from *Camera 2*.



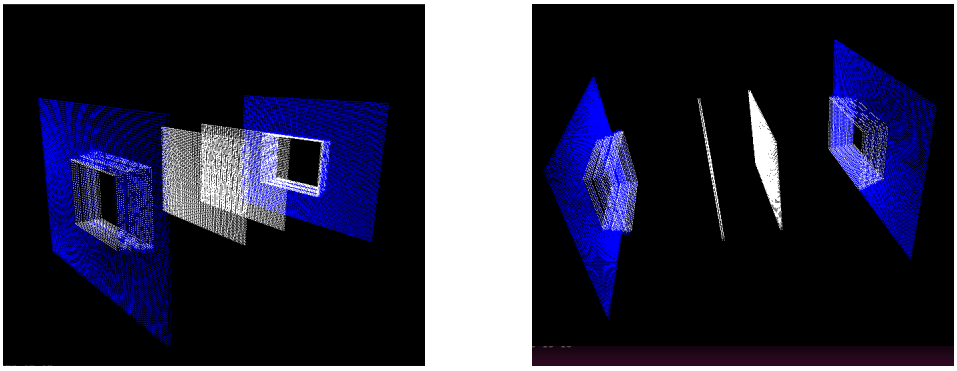
(a) (b)

Figure 7.27: An illustration of how the TSDF values change after each integration of the depth and RGB images. These figures use a background of grey to illustrate negative values, and a white background to illustrate positive values. Subfigure (a) illustrates a cross-section of the TSDF volume shown in Figure 7.25. The TSDF values after the integration of the first depth and RGB image from camera location 1. Subfigure (b) illustrates the TSDF values after the second integration of the depth and RGB image from camera location 2. The values stored in the TSDF after integrating the two depth images does not implicitly represent the actual geometry.

1 has integrated positive TSDF values into the volume, these positions lie behind the surface being integrated from the perspective of *Camera 2*, thus overriding the positive TSDF value in the voxels and replacing these with negative values. This is clearly not the desired result. The resulting point cloud shows that all of the TSDF values inside the volume have resulted in a negative value, because every voxel lies behind and far away from the surface from the perspective of either *Camera 1* or *Camera 2*. To avoid situations like this, where the camera might be facing the surface from the opposite side, a limit must be imposed on the TSDF volume voxel updates.

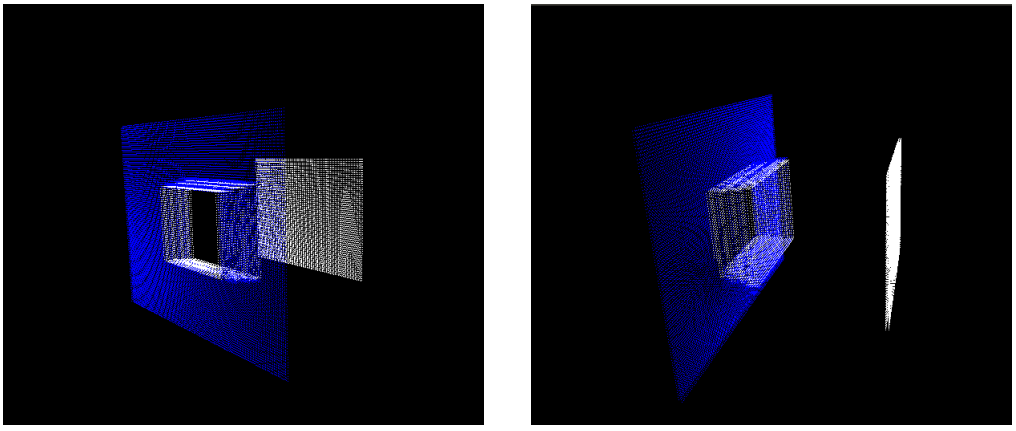
This is addressed with a simple modification to prevent the TSDF volume from updating voxels located too far from the depth measurement. This is implemented by limiting the update region to within 10cm of the surface.

This results in the desired point cloud given in Figure 7.28. Since the system no longer updates all the voxels in the TSDF volume, the hole-filling becomes less effective than that shown in Figure 7.25 when integrating the depth data from a single view point: Instead, the system is only able to fill holes within 10cm of the surface, as shown in Figure 7.29. The hole-filling properties of the three techniques, the original ray caster, the orthogonal ray



(a) (b)

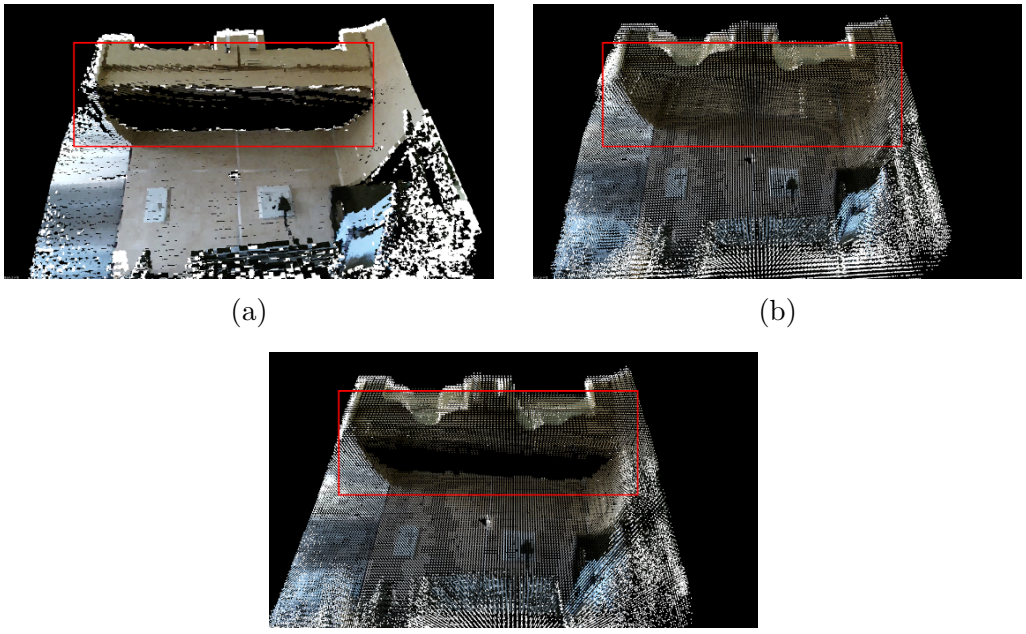
Figure 7.28: The point cloud extraction using the orthogonal ray caster limiting the voxel update region.



(a) (b)

Figure 7.29: Hole filling by updating all TSDF voxels within 10 cm of the surface from two different angles

caster without limiting the update and the orthogonal ray caster with limiting the update region, were examined for synthetic cases. It is confirmed that the ray caster with limiting the update region has the ability to fill regions within a distance of 10cm from the surface and the filled holes are present in the real world data sets. Figure 7.30 shows three point clouds using data collected from the Kinect: the original ray caster for the extraction updating all the voxels, the orthogonal ray caster updating all the voxels, and the orthogonal ray caster with limited voxel updating. The results are similar to the results obtained on the synthetic data sets shown in Figure 7.29.



(a) (b)

(c)

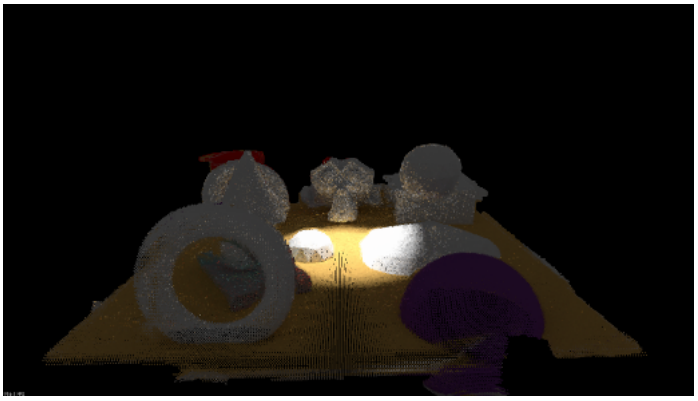
Figure 7.30: Hole-filling using the different methods on real world data set. The region of interest is shown with a red rectangle. Subfigure (a) uses the original ray caster as the point cloud extraction technique as in the original system. It can be seen that the highlighted region has not been filled. Subfigure (b) uses the orthogonal ray caster but updates all voxels in the region. The highlighted region illustrates that it has been filled. Subfigure (c) uses the orthogonal ray caster but limits the voxel update. It illustrates a partially filled region in the highlighted region.

7.3.2 Point Cloud Analysis of Synthetic Data Sets

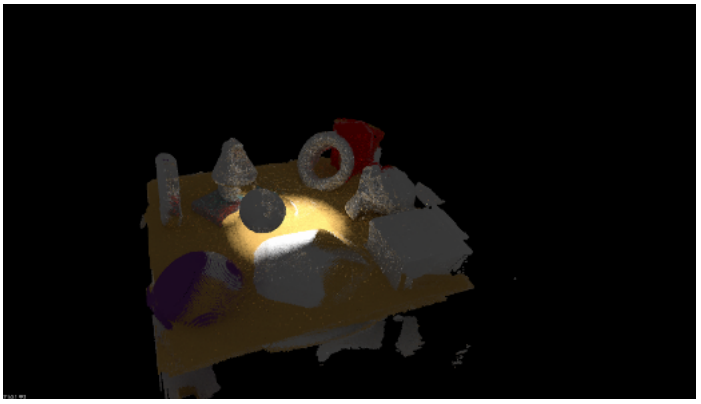
This section analyses the point clouds generated from `Table Top` scene using the `Circling Table` data set and the `Room` scene using the `Moving Around` data set, and discusses observations stemming from the results. All synthetic and real-world data sets were processed by limiting the update region of the TSDF volume and extracting the point cloud using the orthogonal ray caster.

Figure 7.31 shows the reconstructed point cloud of the `Table Top` scene using the `Circling Table` data set (see Figure 7.1) generated by the system. This demonstrates the ability of the system to generate the point cloud from the given depth images.

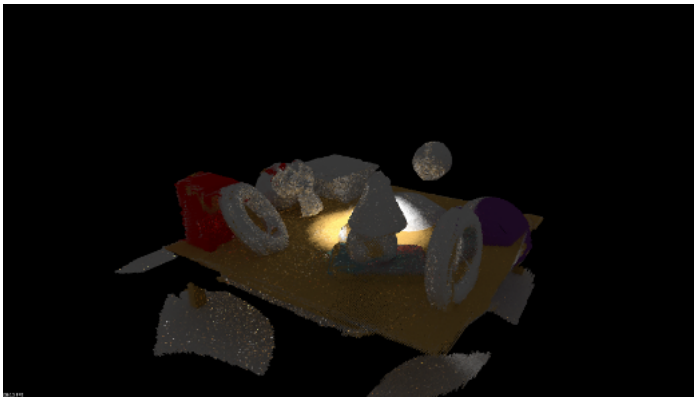
Three problems are observed with this point cloud: the inaccuracy of certain regions of the point cloud, the inaccuracy of some colours, and artefacts (noise) in the point cloud. These problems are addressed by analysing the system implementation and its effect on the results. Due to this system im-



(a)



(b)

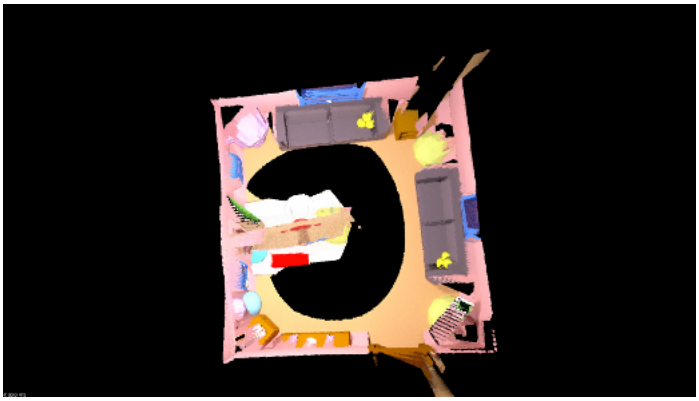


(c)

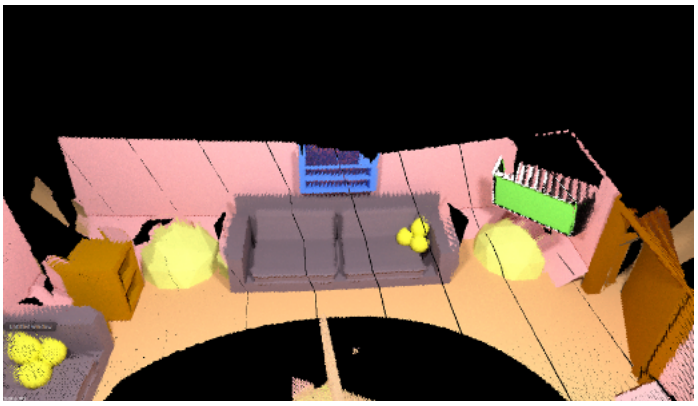
Figure 7.31: The point cloud generated from the `Circling Table` data set, viewed from three different angles.

plementing a voxel-based approach to reconstruction, it tends to produce pixelated results. This can be seen by inspecting the colour of the surface. This also contributes to the loss of detail in certain geometric objects, such as the top of the cone and the detail in the monkey's head. The colour in the reconstruction tends to have a bleeding effect, adding incorrect colour to a nearby area or projectively close area. This effect can be seen on the edges of objects on the table, such as the purple sphere and the monkey head.

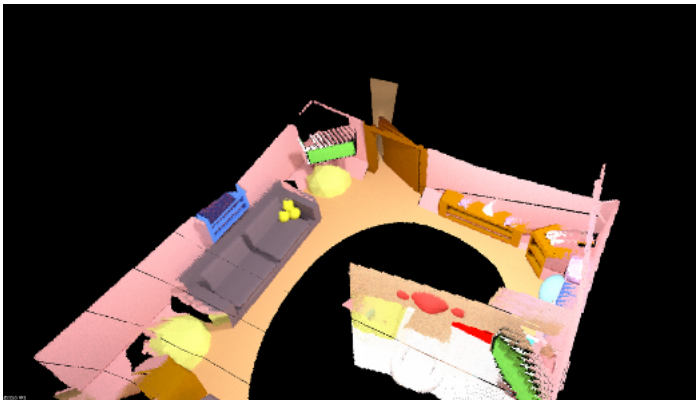
Next, the point cloud generated from the `Moving Around` data set is shown in Figure 7.32. The general shape of the point cloud appears consistent with the synthetic 3D model shown in Figure 7.2. However, the results illustrate an offset in the height from one side of the point cloud to another that can be seen in Figure 7.33. In Figure 7.33a the monkey head model can be seen separated into two sections, the rear part of the monkey head can be seen high up in the image while the jaw of it can be seen much lower. The system warped one of the sides of the point cloud to be higher than the starting area. This warping is due to the error in the camera's estimated height obtained from the ICP, as shown in Figure 7.21c. While the errors obtained from ICP are not initially large, the accumulated error over time can have a large impact on the alignment of the points. This means that even a relatively small error in the ICP algorithm can ultimately lead to large misalignments. Figure 7.34 illustrates the global axes of the scene, the Z axis facing upward, the Y axis in the same direction as the principle axis and the X axis to the right of the camera. The translation analysis shows a large drift of -0.28 meters in the estimated translation along the X and Z axis within the last 500 frames, while the estimated Y axis drifts around 0.1 meters. The drift in the X axis estimate is noticeable from the generated point cloud in Figure 7.32: The point cloud should depict a rectangular room. Instead, the trajectory drift has led to visible distortion. The drift in the Z axis estimate is visible from the warping of the height component of the point cloud. From this section we see how the effects of drift from the motion estimation impact the generated point cloud. Now the real-world data sets `Room Closed Loop`, `Desk`, and `Lab Room` are investigated.



(a)

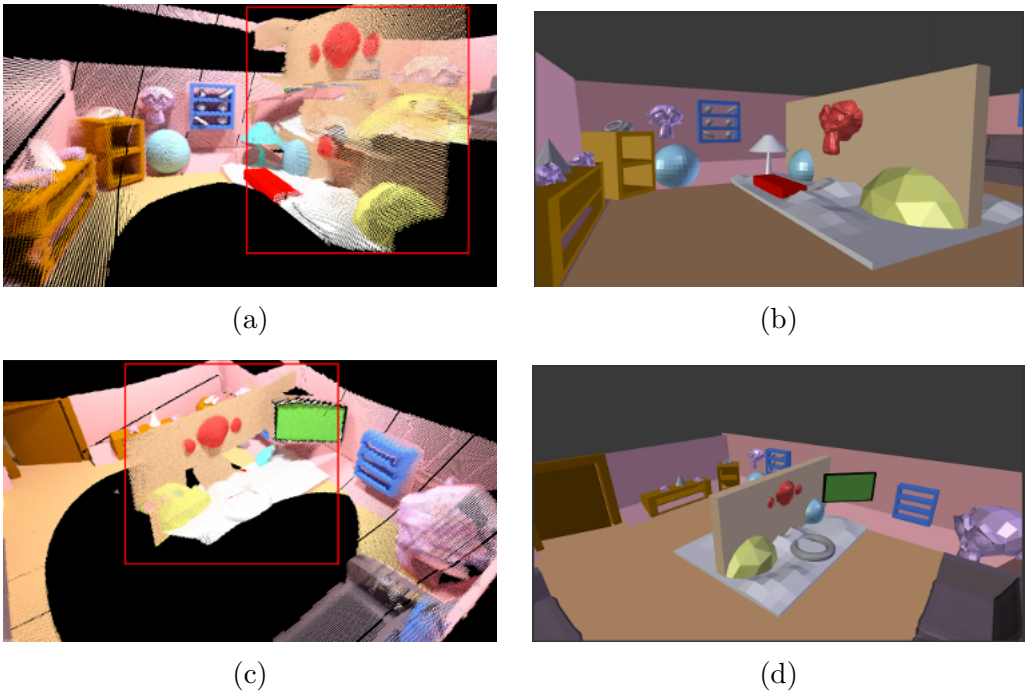


(b)



(c)

Figure 7.32: This shows three image views of the point cloud generated from the `Moving Around` data set. Subfigure (a) shows a view from the top looking down on the point cloud. Subfigure (b) shows a view looking at the left side of the point cloud, and Subfigure (c) shows a second aerial view of the point cloud.



(a) (b)

(c) (d)

Figure 7.33: An illustration of a side by side comparison of the point cloud and the synthetic model generated in Blender. The red rectangle indicates a region of inconsistency due to the motion drift. Subfigure (a) shows the front of the wall (marked in red) from the reconstruction data. Subfigure (b) shows the same view as in Subfigure (a) but from the perspective of the synthetic model. Subfigure (c) illustrates the rear section of the wall (marked in red) from the reconstruction data. Subfigure (d) illustrates the same view as Subfigure (c), but from the perspective of the synthetic model.

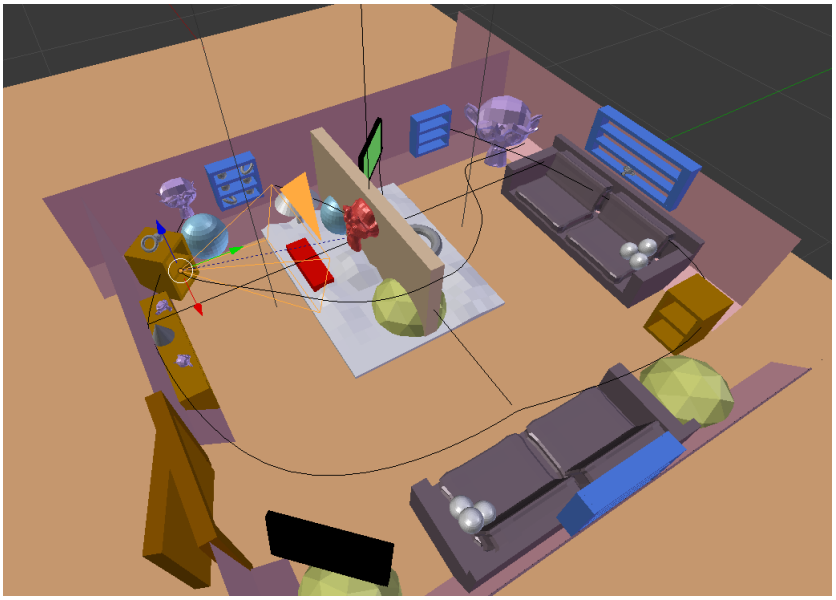
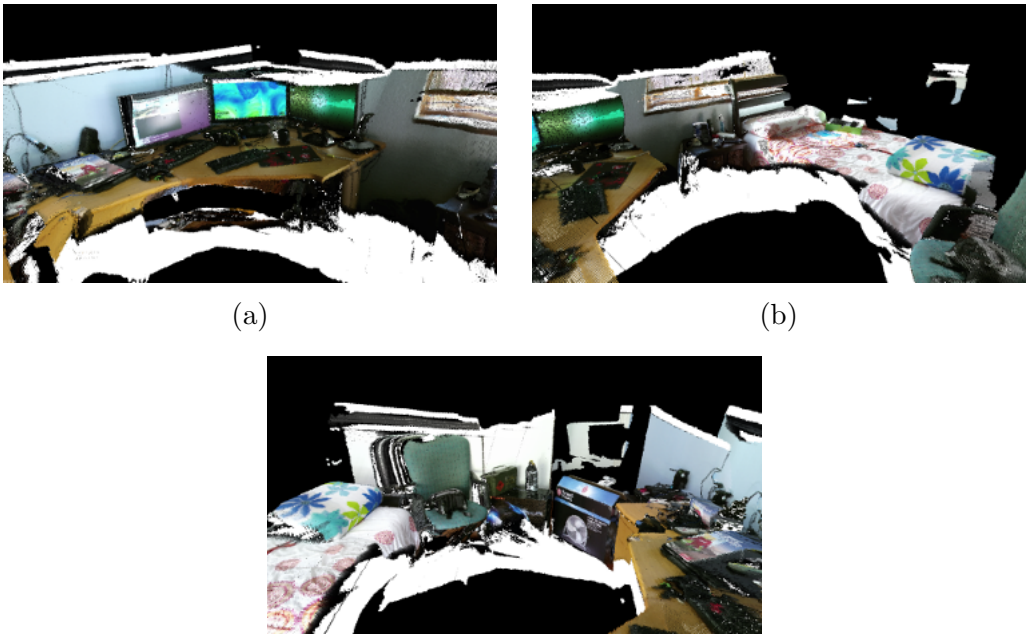


Figure 7.34: The global axes of the `Moving Around` data set in Blender: the X axis in red, the Y axis in green and the Z axis in blue. The camera initially faces the positive Y axis. The positive Z axis would correspond to the camera moving upward and the positive X axis would correspond to the camera moving to the right.

7.3.3 Point Cloud Analysis of Real-World Data Sets

In the real-world data sets no ground truth was obtained so the analysis of the results are qualitative. The `Room Closed Loop` point cloud can be viewed in Figures 7.35 and 7.36. Due to limitations in the RGB data, points that have depth information but no RGB information appear in white in the point cloud. This is due to differences in the camera parameters of the Kinect infrared and RGB camera. An illustration of the differences between the depth and RGB data is shown in Figure 7.37. The main areas that are missing RGB information are the top and bottom of the image this is due to the different camera parameters between the two types of cameras.

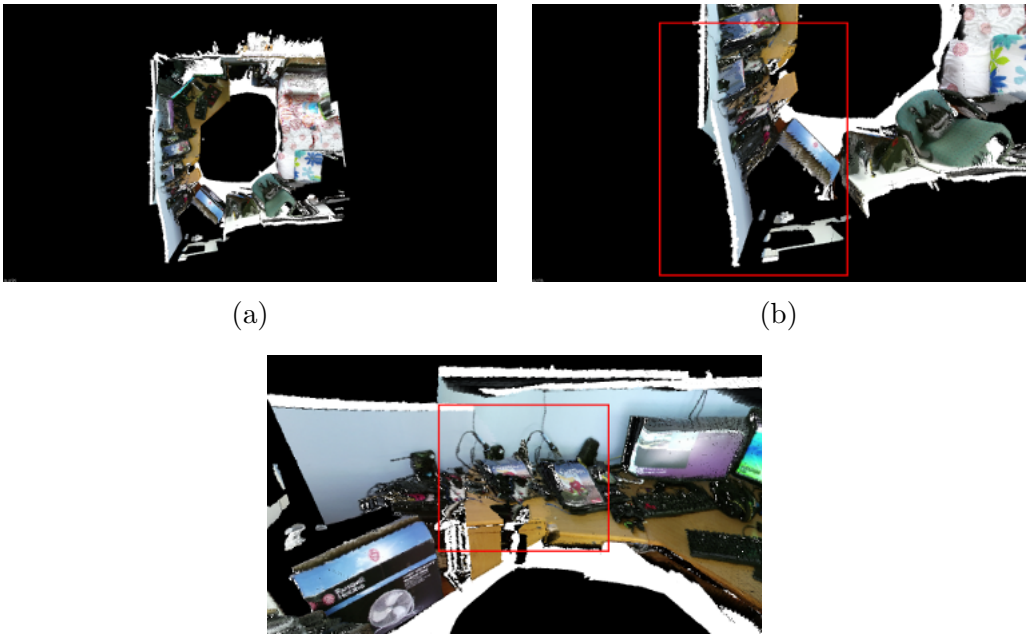
In this experiment, the Kinect sensor is simply moved around in a circle by hand to capture the complete room. In Figure 7.36 a similar effect has occurred as in the `Moving Around` data set: the accumulated error has warped the room such that the walls do not align in the region where the camera starts and ends. In this region, items have been duplicated and the wall alignment is off by a rotational and translational component. On further inspection one can see that other inconsistencies are present in the point clouds. Specifically, Figure 7.38 shows that the surface points on the box next to the cylinder appear to form an indentation. Viewing the incoming depth data in Figure 7.40, it can be seen that the depth data is incorrectly reported by the sensor



(a) (b)

(c)

Figure 7.35: These figures illustrate the reconstructed point cloud from the `Room Closed Loop` data set from three different angles.



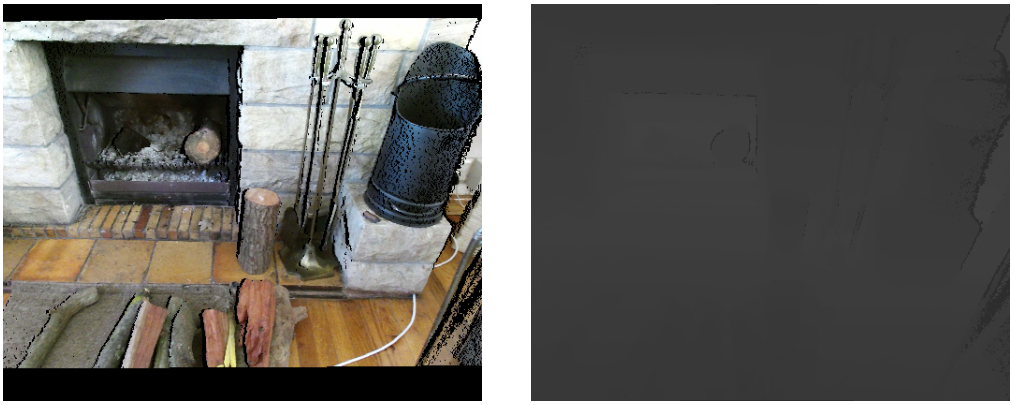
(a) (b)

(c)

Figure 7.36: An illustration of the visible warping that occurred in the `Room Closed Loop` data set. Subfigure (a) provides an aerial view of the reconstructed point cloud, showing visible warping due to the error in the trajectory. Subfigure (b) illustrates the visible warping that occurs at the position where the camera started and stopped the reconstruction (area marked in red). Subfigure (c) illustrates the duplication of a region where the camera closed the loop from the starting and ending position.

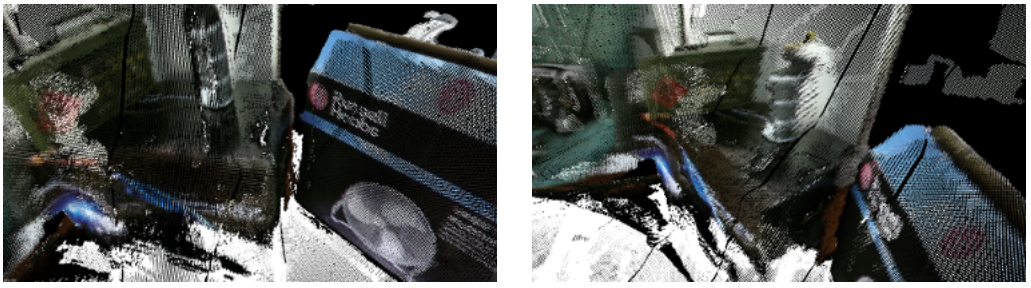
in the highlighted area. This incorrectly reported data is then integrated into the TSDF estimation of the surface. The incorrectly reported depth data can likely be attributed to the reflective nature of the surface: the IR light used by the sensor is reflected off a box and onto the nearby surface. This sensitivity to reflective surfaces is one of the limitations of the Kinect sensor as a sensing device, as discussed in Sarbolandi, Lefloch and Kolb (2015). To verify this hypothesis, a similar test was performed on a reflective granite surface, producing similar results. The reconstruction produced an area of noise at the location of the window as shown in Figure 7.39. However, since there was only a small section of the window in the scene, the issue was limited to a correspondingly small region in the final model. The reason this occurred is due to the reflection of IR light emitted from the sun which is discussed in more detail in the next data set.

The next data set, `Desk` is reconstructed in Figure 7.41. The generated point cloud produces noisy points towards the top of the point cloud. These



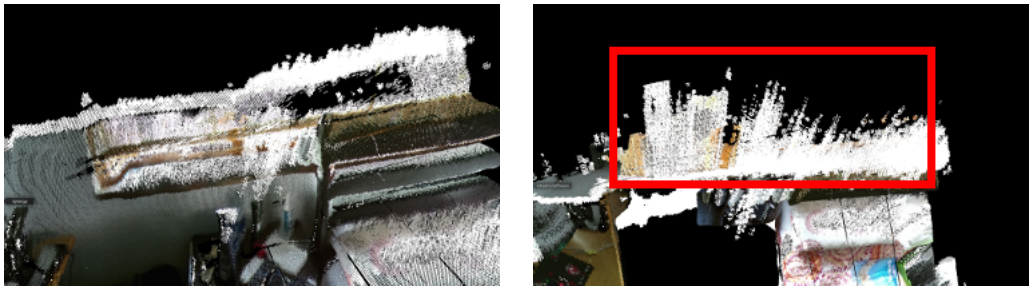
(a) (b)

Figure 7.37: Subfigure (a) illustrates the RGB image and Subfigure (b) illustrates the depth image taken at the same time. However, the RGB image was captured using a high definition wide screen camera while the depth image was captured using an infra-red camera with different camera parameters and from a different position than the RGB camera. Subfigure (a) is a corrected version of the original RGB image to match the location and camera parameters of the infra-red camera which is used for all the experiments. These images illustrate the RGB and depth image as if they were taken from the same position. The RGB image has been corrected to simulate this. Since the RGB image has not been taken from the exact same position, some information is missing due to the perspective change. This is commonly noticed along the edges of objects. It is also missing RGB information at the top and bottom of the image (black bars) while the depth image has information in this area.



(a) (b)

Figure 7.38: An illustration of the apparent indentation on the box next to the cylinder. This can be hard to see in the 2D images.



(a) (b)

Figure 7.39: These figures illustrate the noisy points generated in the `Room Closed Loop` data set in the region of the window from two different angles. The spray of white points behind the surface is the noise generated from external factors.

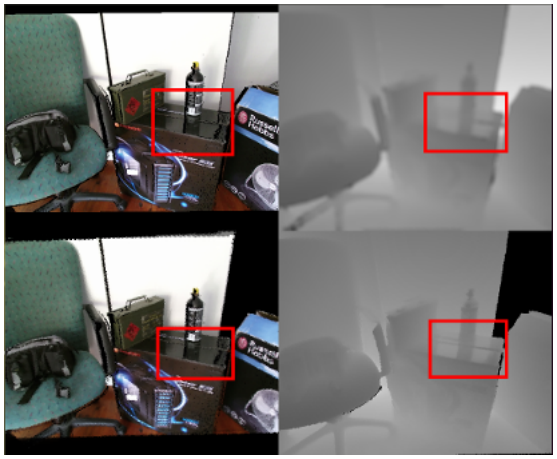
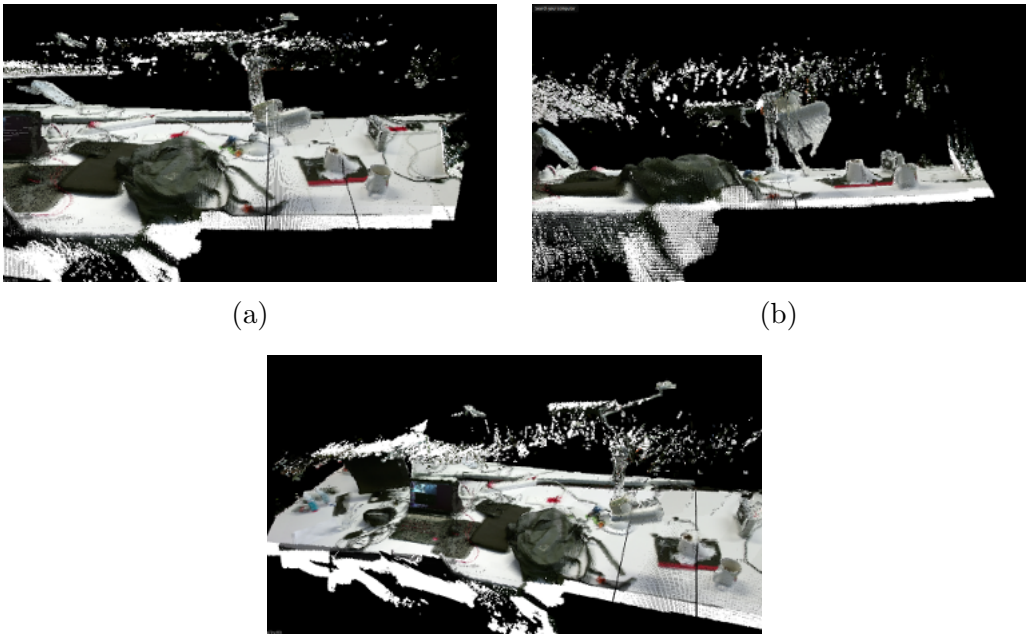


Figure 7.40: This demonstrates the Kinect's inability to handle very reflective surfaces as seen in this image. This image shows four views of the surface, the top left image is the raw RGB image, the top right image illustrates the bilaterally filtered depth image and the bottom images show the view from the RGB volume and the TSDF volume. A reflection of the surface behind the box is shown on the box. The IR light is being reflected.



(a) (b)

(c)

Figure 7.41: Various artefacts near the top of the point cloud generated from noisy sensor data in the `Desk` data set.

points clearly do not represent a surface due to their position. To examine the source of these points, a view of the incoming depth image is shown in Figure 7.42. Further examination of the depth image reveals that the artefacts in the point cloud were created where there is no depth data in the depth image. This effect results from the interaction of two factors, the first being that the IR light is being sent too far and unable to return to the sensor resulting in most of the region returning a 0 measurement (black). The second interaction is that the data set was recorded in an environment featuring substantial reflected sunlight. A known issue of the Kinect sensor and its basis of operation is its sensitivity to IR light: this can cause the direct or reflected IR light from the sun or other sources to overpower the IR light sent from the sensor. This can cause incorrect depth measurements in the data if the reflected IR light from another source is stronger in a region where the sensor's IR light is weaker or too weak to report the correct values (i.e. the surface is out of the sensor's IR light range or the IR light is weaker than the alternate source). In this experiment, areas where the reflected IR light from the sun was more intense than the IR light reflected to the Kinect reported incorrect depth data.

Similar results are obtained using the `Lab Room` data set, in which incorrect depth data produces a noisy set of points in the reconstruction. Figure 7.43 shows the reconstructed point cloud with the noise being produced from the

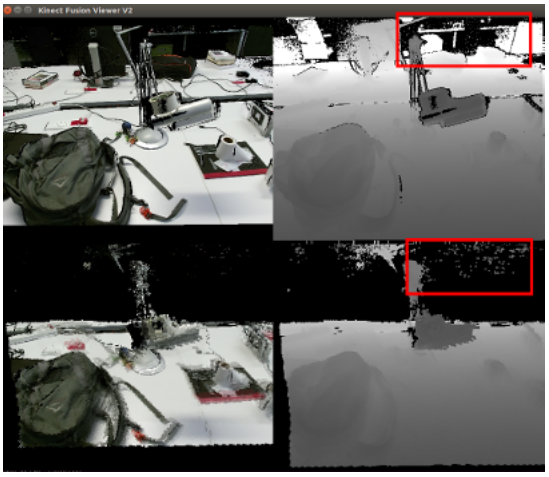
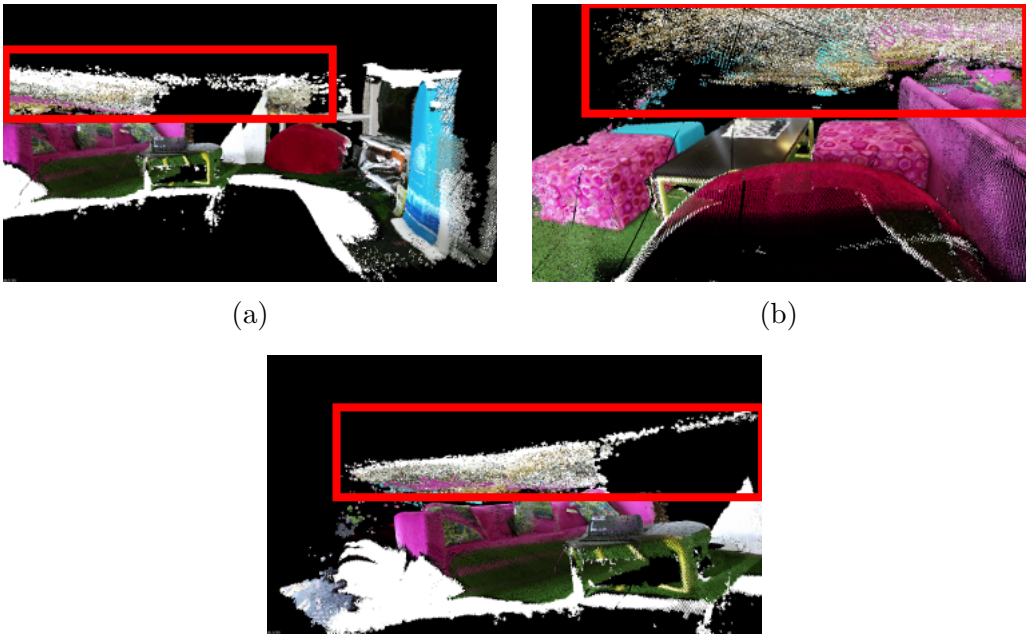


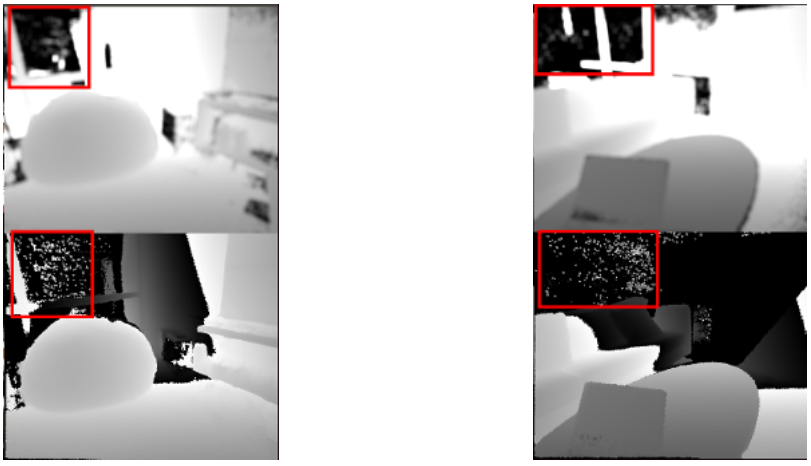
Figure 7.42: This image shows four views of the surface, the top left image is the raw RGB image, the top right image illustrates the bilaterally filtered depth image and the bottom images show the view from the RGB volume and the TSDF volume. The input RGB and depth images from the `Desk` data set with the red rectangle highlighting the sensor noise obtained from interference due to reflected sunlight.



(a) (b)

(c)

Figure 7.43: Point cloud reconstruction from the `Lab Room` data set. As in Figure 7.41, artefacts generated from reflected IR light are present.



(a) (b)

Figure 7.44: The input RGB and depth images showing the noise generated from the reflected IR light causing problems in Figure 7.43.

raw depth images shown in Figure 7.44.

7.3.4 Summary

This section examined point cloud extraction techniques for extracting implicit information about the surfaces contained within the TSDF volume. The orthogonal ray caster method was capable of extracting additional surfaces from the TSDF volume. However, to avoid overwriting TSDF values in situations where the camera views an object from opposite sides, only voxels that lie within a certain distance of the depth measurement are allowed to be updated. This allows the system to integrate depth data into the TSDF volume from multiple angles without overwriting TSDF values that have been integrated from behind the surface.

In experiments on synthetic data sets, a clear warping of the generated point clouds was observed. Comparisons to the trajectory analysis indicated that the warping is related to the drift from the ICP algorithm. In the `Moving Around` data set, a high error on the final Z axis position indicated that the camera height estimate was higher than its starting height. This is observed in the point cloud as the point cloud do not align with respect to the height at the start and end region of the point cloud. A similar issue occurred for the X axis.

Experiments were also performed to a lesser extent on some data sets that were recorded using the Kinect in two environments namely; a bedroom and a lab. Using the data set generated from the bedroom, `Room Closed Loop`, the system produced a point cloud that suffered from the same effect apparent

in the synthetic point clouds, a warping of the points in the point cloud with respect to the start and end frames. This point cloud had two additional issues that were not present in the synthetic experiments. In certain areas, there were sets of noisy points generated, while in other areas, indentations in the point cloud had formed that did not correspond to the real-world surfaces. These problems were analysed and it was found that the noisy points generated were due to incorrect depth measurements from IR light reflected by other sources into the IR sensor. This led to inaccuracies in the depth maps returned from the sensor. Therefore, spurious IR sources should be avoided at all times when working with the Kinect sensor and other IR based sensors. The issue of spurious indentations appearing in certain areas was discovered to be due to the reflectivity of the surface the sensor was viewing. The IR light had reflected off these surfaces and reported the depth from the sensor to the reflective surface plus the additional distance to the next less reflective surface the IR light hit. This was verified by testing the sensor data on a large reflective surface.

The next section, presents an analysis of the mesh reconstruction from the generated point clouds.

7.4 Mesh Generation

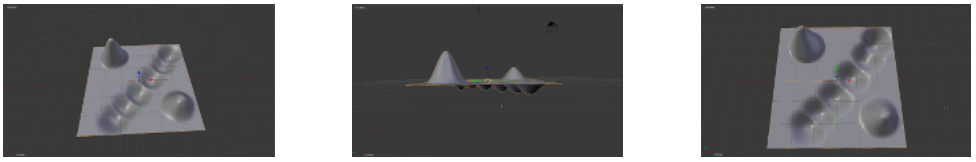
A selection of synthetic point clouds are used to evaluate the mesh generation algorithm. Following this, a selection of point clouds from the previous data sets are used to evaluate the system. The mesh generation component of the system was tested by initially creating a 3D model in Blender and using Blender's Python interface to extract the model's vertices and their associated normal vectors to use as ground truth. Initially, a few basic synthetic tests were performed to verify that the mesh generation works as expected. Following this, the mesh generation algorithm is applied to the synthetic and real-world data sets mentioned in Section 7.1. The results of the real-world data sets are analysed qualitatively due to there being no ground truth data available.

7.4.1 Synthetic Data Sets

This section details various synthetic data sets that were initially used to test the mesh reconstruction algorithm in order to verify that it can reconstruct the faces of point clouds.

Terrain Reconstruction

The `Terrain reconstruction` test is a modified plane with a series of hills and valleys that simulates a basic terrain surface, as is illustrated in Figure 7.45. A simple surface with multiple dips and bumps simulating a basic terrain



(a) (b) (c)

Figure 7.45: Blender views of the `Terrain reconstruction` data set.

surface. This is the initial test that will be used to ensure that the BPA works correctly. This data set was self-designed as a basic test for the BPA.

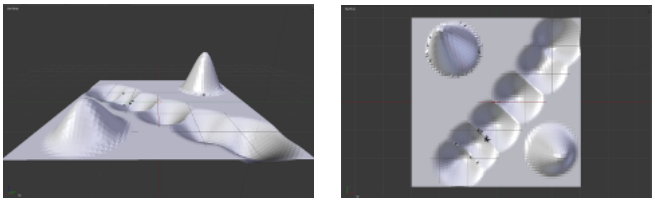
Stanford Bunny

We also use the Stanford bunny model, a widely used point cloud from the Stanford 3D scanning repository (Curless and Levoy, 1996*b*), as the basis for testing. Due to the post-processing required for the Stanford point clouds, only the `Stanford Bunny` data set was used. The Stanford point clouds provided the vertices of each point without the surface normal information required for the BPA. The surface normal information for each point needs to be generated manually. With point clouds of tens of thousands of points, this was infeasible for all other Stanford data sets.

7.4.2 Experiments

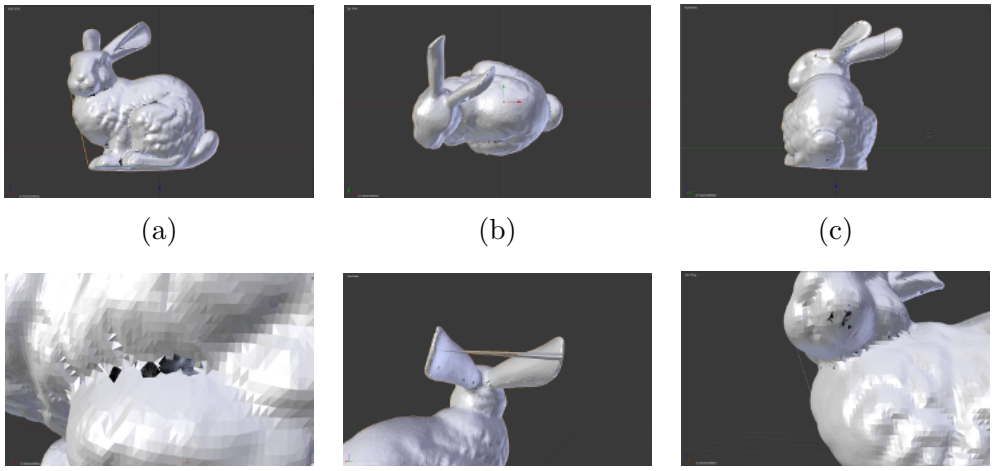
The initial tests of the BPA are show in Figure 7.46. It illustrates the mesh generated from the `Terrain reconstruction`. The algorithm produced a generally well-structured mesh, capable of easily reconstructing the surface of the mesh in areas that did not have a sharp change in gradient. However, in most areas where steep gradient changes occurred the mesh generation algorithm was unable to fill some of these areas with the appropriate triangles. The reconstruction produced accurate results in areas of small gradient changes.

Once the system produced adequate reconstruction results that resemble the original mesh, the system was tested using the Stanford bunny model. Figure 7.47 shows the generated mesh from the `Stanford Bunny` point cloud. It can be seen that there are two issues that occur in the mesh. Some areas of the mesh have missing triangles as seen in Figures 7.47c, 7.47d and 7.47f. In certain situations when processing fine detail, the BPA is unable to create a p-ball that only touches three points without including any additional points. This case occurs when multiple p-balls are used in areas with large changes in gradient. The missing triangles can be seen in Figure 7.46 and Figure 7.47d . The large triangles created between the ears shown in Figures 7.47a and 7.47e, are created by choosing a p-ball that is too large for the density of the given



(a) (b)

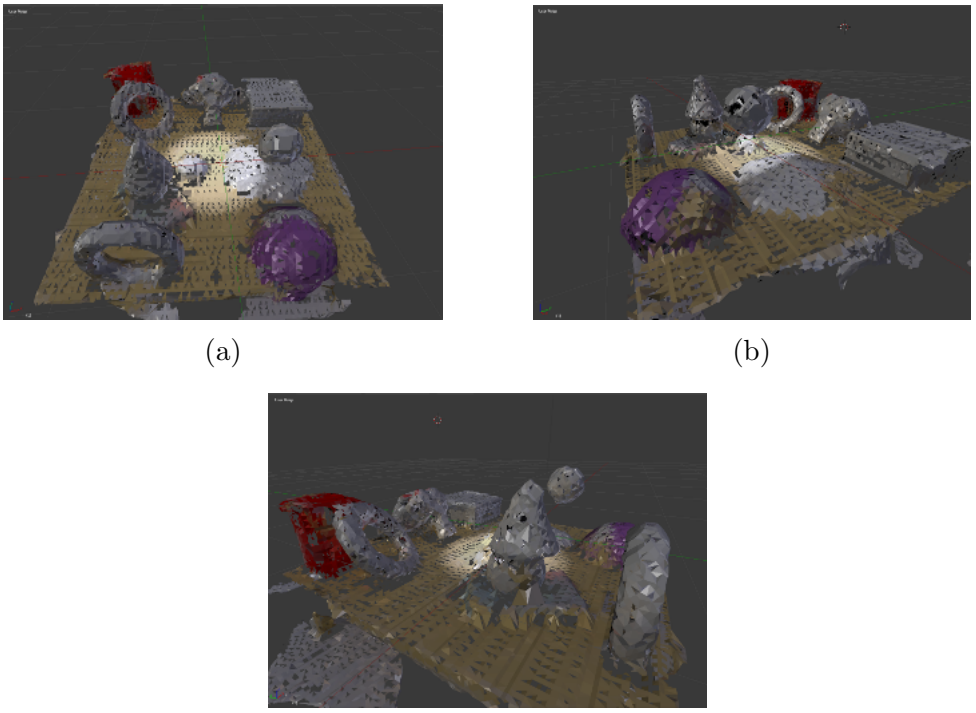
Figure 7.46: This illustrates the reconstruction of the `Terrain reconstruction` point cloud.



(a) (b) (c)

(d) (e) (f)

Figure 7.47: An illustration of the generated mesh of the `Stanford Bunny` data set. Subfigures (a) - (c) illustrate an overview of the generated mesh from two side views and a top view. Subfigure (d) and (f) illustrates sections of missing triangles in the mesh. Subfigure (e) illustrates an area where a triangle is incorrectly generated.



(a) (b)

(c)

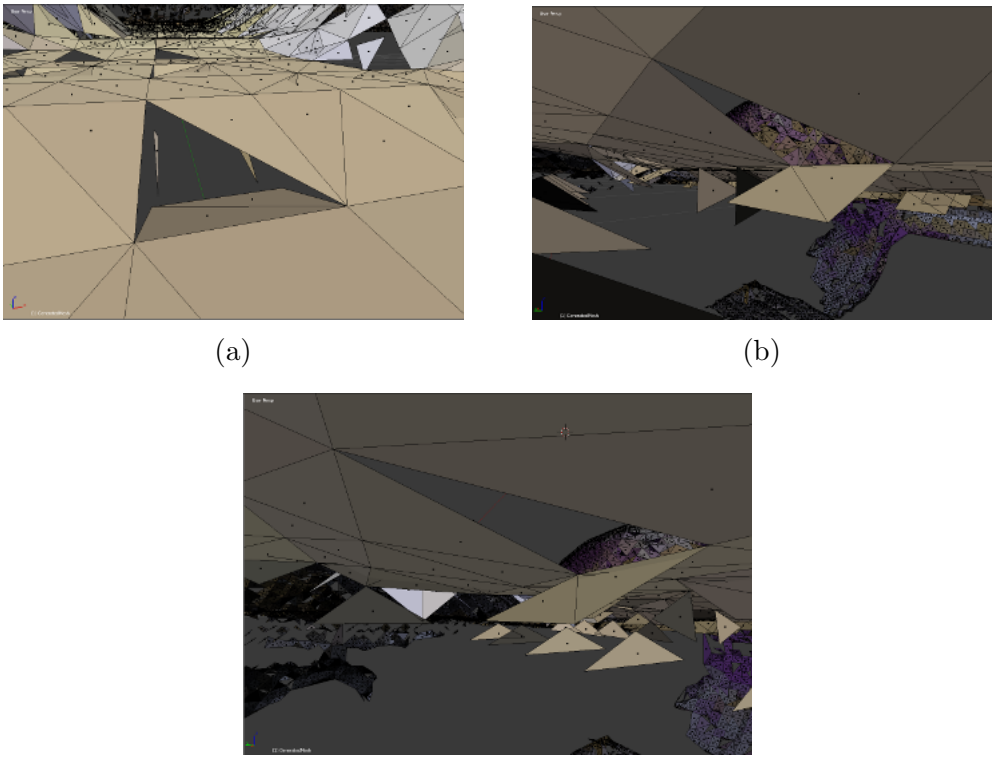
Figure 7.48: Various views of the mesh reconstruction for the `Circling Table` data sets.

point cloud. This produces incorrect triangles for the model.

During the mesh assembly, building the model in Blender using the results from the BPA, Blender has a limit on the maximum number of materials it can use in a model. Due to this, some of these generations do not include colour data. Since the extracted point cloud is too dense, a voxel grid filter is used to replace all points in a $0.025\,\mathrm{m}^3$ volume with a single point that represents the centroid of the points in the volume. The original Kinect Fusion system did not have this implemented as the reconstruction area was limited due to the system being unable to move the TSDF volume. This is performed due to the number of points created during the point cloud extraction. This reduces the number of points from millions to tens of thousands, which decreases the runtime of the mesh generation. Next, the reconstruction results from the data sets `Circling Table`, `Moving Around` and `Room Closed Loop` are considered.

Mesh Construction

This section analyses the meshes constructed from the data sets `Circling Table`, `Moving Around` and `Room Closed Loop` presented in Section 7.1. Look-



(a) (b)

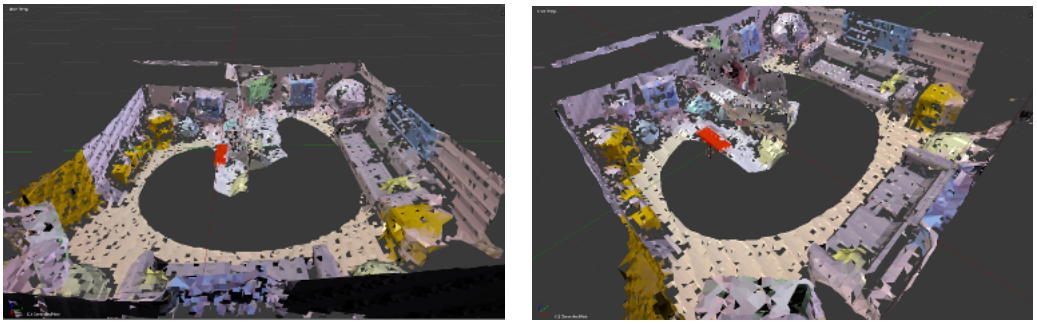
(c)

Figure 7.49: An illustration of the `Circling Table` data set's mesh reconstruction close up.

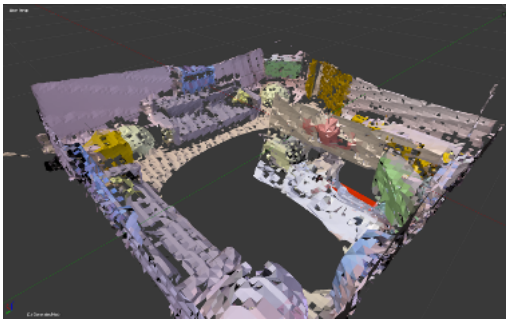
ing at the mesh generation in Figure 7.48, many examples of missing triangles are observed, far more than the number observed for the `Stanford Bunny` data set. Looking closer at the constructed mesh, sudden spikes along the surface throughout the mesh are shown. The BPA created triangles that are initially under the surface as shown in Figure 7.49. This situation can occur when the small ball size is insufficient to complete the surface, a larger ball was used that interpolated over the previous meshed surface. This results in triangles that are connected on one side and completely disconnected on the other, as shown in the figure. This behaviour could be attributed to multiple factors such as noise, the order in which the points are selected for generated triangles, the accuracy of the point cloud extraction and the voxel grid filter changing the centroid of the points.

Similar results are observed in Figures 7.50 and 7.51 for a synthetic and real world data set respectively. These observations are visible on all results of the synthetic and real world data sets.

Figures 7.46 and 7.47 clearly show that the algorithm can successfully construct the meshes. However, the point clouds from the system generate a mesh



(a) (b)



(c)

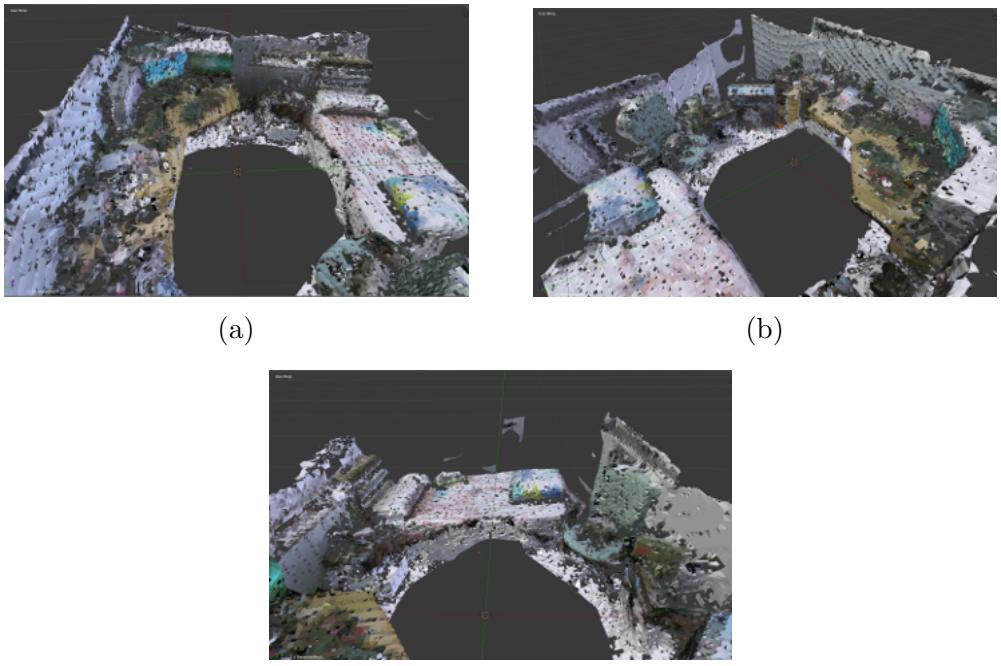
Figure 7.50: Three views of the mesh reconstruction, of the `Moving Around` data set.

that has noisy surfaces. This could be due to the noise generated when integrating the TSDF data into the volume or the adjustments the voxel grid filter introduces. This contrast in meshing accuracy suggests that this method of mesh generation is more practical for meshing smaller scale and less noisy data sets such as objects as opposed to environment.

7.5 Summary

The trajectory reconstructions generated by the system provided a reasonable estimate of the initial camera path with a relatively small error as shown in Section 7.2. However, it was also observed that the accumulated error over many frames can distort the final model significantly.

In Section 7.2.5, an attempt at using a kind of feature-based motion estimation technique with the frame-to-model implementation failed due to the low resolution output from the ray caster's colour volume. This resulted in insufficient features being detected in the ray cast image, leading to insufficient matching pairs. Ultimately this prevented feature-based motion estimation from further implementation.



(a) (b)

(c)

Figure 7.51: Three views of the mesh reconstruction of the `Room Closed Loop` data set.

From the analysis of point clouds performed in Section 7.3, it can be concluded that the original ray casting method provides a dense point cloud that can be sufficient for general use. Using the orthogonal ray casting technique, extraction of additional information about the surfaces represented by the TSDF was obtained. If it is assumed that the system would not have to view the same area again from the opposite side as illustrated in Figure 7.27, the TSDF volume could update all the voxels in the reconstruction region. This would allow the system to fill any section where there is a sudden change in depth information from the sensor. However, this assumption is not made in this system because certain situations can occur where this would negatively impact the generated point cloud. Instead, we limit the voxel region update, updating voxels within 10cm of the surface measurement.

Some important limitations of the Kinect sensor were highlighted in these experiments, namely the inability of the sensor to handle reflective surfaces or spurious IR light reflecting onto the sensor. These limitations constrain the system to use in indoor environments or night-time scenarios with artificial light where there is minimal IR light present.

While the system can produce high resolution point clouds, which accurately represent the 3D world these do suffer from accumulated trajectory error. Although not implemented in this work, such drift can in principle be

corrected using loop closure techniques.

Overall, the mesh generation experiments in Section 7.4 showed a reasonable measure of success, but there are clear issues when using the extracted point cloud data. The BPA was unable to completely rebuild most of these data sets. The mesh generation experiments that were performed on clean synthetic point clouds worked well, however there are still limitations.

An issue with using the BPA is that the points in the cloud require the associated normals which, in other reconstruction methods, are typically difficult to obtain. This technique seems better adapted to cleaner point clouds than the ones produced by the proposed system. This might be addressed by preprocessing the candidate point cloud using some form of function fitting to smooth out the noise in the point cloud. The concept of inputting a sequence of ball sizes also limits the reconstruction: if the system is given balls that are too big or too small, the reconstruction will perform poorly or entirely fail. Although an ideal place to start is using the average distance between the points as the smallest ball size, this is a factor that will significantly impact the runtime of this algorithm. The next chapter concludes the thesis and discusses future work that can improve this system as a whole.

Chapter 8

Conclusion

This chapter gives an overview of the thesis, reviews its aims and objectives, evaluates whether they have been achieved, and discusses potential improvements and extensions to the current implementation.

8.1 Summary

This thesis focused on researching and implementing an approach allowing non-specialists to easily perform fast environment reconstruction using consumer-grade products. Specifically, this work considered the low-cost Microsoft Kinect sensor, which allows consumers to record depth and colour data from the environment. A system was proposed for 3D reconstruction that is capable of high throughput motion estimation on a large scale reconstruction, and can be operated without much user intervention.

This thesis provided a review of fundamental concepts in Chapter 2 and presented a summary of various techniques and approaches that have been considered by other researchers in Chapter 3. Chapter 4 discussed a methodology that established the approach employed in this thesis and provided detail on the testing procedure employed when evaluating the proposed system. This thesis then detailed the motion estimation technique and the volumetric method used to construct the 3D model in Chapter 5, while Chapter 6 detailed the mesh generation algorithm. The detailed techniques were presented with tests and experiments. The tests and experiments were performed to evaluate the system in Chapter 7. Next, we review the aims and objects for the thesis and evaluate them.

8.1.1 Aims and Objectives

This section reviews the aims and objectives of this thesis, before evaluating how the thesis contributes towards their fulfilment.

Collecting Data

In this thesis we used the Microsoft Kinect to acquire depth and colour data from the environment that can be used to reconstruct the environment. This is a low-cost consumer grade device that is widely accessible.

Estimating the Motion of the Depth Sensor at Online Speeds

An iterative closest point technique based on scan-matching was used to compute the motion of the Kinect sensor. However, this motion estimation technique suffers from drift over the image sequence. Over a large area, it is observed that the system begins to warp the model such that it is consistent with the drift in the motion estimate. This source of error was reduced by using a frame-to-model method as opposed to a frame-to-frame method of motion estimation. The iterative closest point technique allows the system to estimate motion at a frame rate of 30Hz using readily available GPU hardware and the CUDA framework, thus allowing real-time reconstruction of the environment. An alternative motion estimation technique was investigated by adding feature-based motion estimation using a frame-to-model approach. However, this produced poor features from the model view. Since it was unable to create enough feature matches between points, this alternative method was thus unused.

Pre-processing Sensor Data to Reduce Noise

To further improve the quality of the reconstructed point cloud, the sensor data was preprocessed by applying a bilateral filter, to help reduce noise in the sensor data and improve the motion estimation. This is as discussed in Section 5.3

Fuse Successive Sensory Data Sets Together

A volumetric technique using a truncated signed distance function was used to fuse successive depth images together. The TSDF allows the system to compute an average surface location for each surface measurement obtained by the depth sensor. This helps to reduce noise and adjust for inconsistencies within the environment such as noise in the sensor readings.

Enable Generation of Large Scale Environments

The system was able to produce larger environments than the original system (Newcombe *et al.*, 2011*a*) by introducing a moving volume, using a wraparound indexing system for the memory allocated to the TSDF volume. The original system allowed the capture of a global volume of $512 \times 512 \times 512$ voxels.

This system removes this limit and allows the system to integrate new areas until memory limitations are reached by the computer. The `Moving Around` experiment covered three $512 \times 512 \times 512$ volumes.

Extraction of Implicit and Explicit Structure

Investigation into extracting the inferred structure from the volumetric model using hole-filling techniques proved fruitful. This allows the system to extract a surface even where there are no direct observations. This is achieved by using an orthogonal ray caster that searches for the point where TSDF values change from positive to negative along each axis of the volume.

Generate a Mesh from the Point Cloud and Integrate Colour

The technique of point cloud extraction provides a set of points that are separated by a minimum distance of the voxel size, due to each voxel representing a point in world space. The resulting point cloud with its associated colours are converted to a colourized mesh by applying a surface reconstruction algorithm. In this system, the ball-pivoting algorithm (Bernardini, Mittleman, Rushmeier, Silva and Taubin, 1999) was chosen due to its simplicity. The results of this algorithm proved mixed. The initial testing stages on the Stanford `Bunny` and `Terrain reconstruction` data sets, the results indicated that it was able to reconstruct the surfaces with a reasonable degree of accuracy. However, in the synthetic and real-world data sets that were generated from the Kinect Fusion system the generated mesh showed inconsistent surfaces that did not depict the true structure of the environment. Planar surfaces had noise attached to the surface and produce a rough plane with sections of missing triangles. It was possible to see the general structure of the environment, but this is not well represented in the generated mesh.

Exporting the Mesh to a Universal Format

The mesh is finally converted into the `blend` file format used in Blender, the mesh is converted to the `FBX` file format using Blender's internal API.

Summary

Most of the aims and objectives of this thesis have been met, the proposed system is capable of estimating the motion of the Microsoft Kinect sensor at 30Hz and integrating the depth and colour information into a 3D volume that builds a point cloud representing the model. The data represented in the 3D volume is extracted using a technique that extracts implicit and explicit points from the 3D volume. This is then passed to a mesh generation algorithm that is capable of generating a mesh that largely represents the environment. The

mesh generation provided a rough structure of the point cloud; however, it did not closely match the true geometry of the real-world environment.

8.2 Future Work

This section identifies a few promising extensions that can be incorporated into the system to improve its ability to reconstruct the environment.

- Correcting the drift can be improved by replacing the current motion estimation technique with a more reliable motion estimation technique such as a feature-based motion estimator or including sensor data from an inertial motion unit (Huang and Bachrach, 2017).
- Correcting drift, can also be performed by incorporating loop closure techniques that allow the system to correct itself after it views an area that has already been seen (Ho and Newman, 2007).
- When the sensor revisits an area that has previously been extracted as a point cloud, the point cloud could be integrated back into the TSDF volume. This would allow old data to be updated and allow the system to perform corrections on its position. If old data does not align with the recently acquired sensor information, it could then be corrected.
- A preprocessing method to detect and correct reflections and spurious IR light that might affect the sensor data, could improve point cloud quality.
- To improve the mesh reconstruction, eliminating the need for the voxel grid filter and adding post-processing of the point clouds to remove noise could help to improve the results.
- Instead of using an RGB volume to associate a point with a colour, the RGB images can be projected onto the generated mesh from the different camera locations as a texture for the model. This should produce a higher resolution on the RGB mesh for the colour.

8.3 Reproducibility

To facilitate reproducibility and future extensions, this system is provided as an open-source C++ implementation of the Kinect Fusion algorithm, including the hole-filling and moving volume extensions. It uses the CUDA framework and is available at `https://gitlab.com/pleased/3d-reconstruction` under the MIT license.

List of References

Amenta, N., Bern, M. and Kamvysselis, M. (1998). A new Voronoi-based surface reconstruction algorithm. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '98*, pp. 415–421.

Amenta, N., Choi, S. and Kolluri, R.K. (2001). The power crust. In: *Proceedings of the sixth ACM symposium on Solid Modelling and Applications*, pp. 249–266. ACM.

Arulampalam, M.S., Maskell, S. and Gordon, N. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics and Automation Magazine*, vol. 13, pp. 108–117.

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C. and Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359.

Besl, P. and McKay, N. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256.

Bium, H. (1964). A transformation for extracting new descriptions of shape. In: *Symposium on Models for the Perception of Speech and Visual Form.*

Blackwell, D. (1947). Conditional expectation and unbiased sequential estimation. *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 105–110.

Blender Online Community (2018). *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam. Accessed: 2018-04-16.
Available at: `http://www.blender.org`

Boissonnat, J.-D. (1984). Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics (TOG)*, vol. 3, no. 4, pp. 266–286.

Bolitho, M., Kazhdan, M., Burns, R. and Hoppe, H. (2009). Parallel Poisson surface reconstruction. *Advances in Visual Computing*, vol. 5875, pp. 678–689.

Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155.

Curless, B. and Levoy, M. (1996*a*). A volumetric method for building complex models from range images. *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '96*, pp. 303–312.

Curless, B. and Levoy, M. (1996*b*). Stanford 3D scanning repository. Accessed: 2018-01-08.
Available at: `http://graphics.stanford.edu/data/3Dscanrep/`

Cyberware Incorporated (2018). Cyberware 3030. Accessed: 2018-04-16.
Available at: `http://cyberware.com/products/scanners/3030.html`

Davison, A.J., Reid, I.D., Molton, N.D. and Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 1052–1067.

Delaunay, B. (1934). Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, pp. 793–800.

Dellaert, F., Fox, D., Burgard, W. and Thrun, S. (1999). Monte Carlo localization for mobile robots. *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1322–1328.

Dey, T.K. and Giesen, J. (2001). Detecting undersampling in surface reconstruction. In: *Proceedings of the seventeenth annual symposium on Computational geometry - SCG '01*, pp. 257–263.

Dey, T.K. and Goswami, S. (2003). Tight Cocone: a water-tight surface reconstructor. *Journal of Computing and Information Science in Engineering*, vol. 3, pp. 302 – 307.

Doucet, A., de Freitas, N., Murphy, K. and Russell, S. (2000). Rao-Blackwellised particle filtering for dynamic Bayesian networks. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 176–183.

Durrant-Whyte, H. (1988). Uncertain geometry in robotics. *IEEE Journal on Robotics and Automation*, vol. 4, pp. 23 – 31.

Gamini Dissanayake, M.W.M., Newman, P., Clark, S., Durrant-Whyte, H.F. and Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 229–241.

Gopi, M. and Krishnan, S. (2000). A Fast and Efficient Projection-Based Approach for Surface Reconstruction. *High Performance Computer Graphics, Multimedia and Visualisation*, vol. 1, no. 1, pp. 1–12.

Gotsman, C. and Keren, D. (1998). Tight fitting of convex polyhedral shapes. *International Journal of Shape Modeling*, vol. 4, pp. 111–126.

Grisetti, G., Kummerle, R., Stachniss, C. and Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43.

Hamilton, W.R. (1844). II. On quaternions; or on a new system of imaginaries in algebra. *Philosophical Magazine Series 3*, vol. 25, no. 163, pp. 10–13.

Harris, C.G. and Pike, J.M. (1988). 3D positional integration from image sequences. *Image and Vision Computing*, vol. 6, no. 2, pp. 87–90.

Hartley, R.I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. 2nd edn. Cambridge University Press, ISBN: 0521540518.

Ho, K.L. and Newman, P. (2007). Detecting loop closure with scene sequences. *International Journal of Computer Vision*, vol. 74, no. 3, pp. 261–286.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W. (1992). Surface reconstruction from unorganized points. *ACM SIGGRAPH Computer Graphics*, vol. 26, pp. 71–78.

Horn, B.K. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, vol. 4, no. 4, pp. 629–642.

Howard, A. (2008). Real-time stereo visual odometry for autonomous ground vehicles. *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008.*, pp. 3946–3952.

Huang, A. and Bachrach, A. (2017). Visual odometry and mapping for autonomous flight using an RGB-D camera. In: *Robotics Research*, pp. 235–252. Springer.

IIPImage (2018). Accessed: 2018-04-16. Available at: `http://iipimage.sourceforge.net/documentation/images/`

Isselhard, F., Brunnett, G. and Schreiber, T. (1997). Polyhedral reconstruction of 3D objects by tetrahedra removal.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D. and Davison, A. (2011). KinectFusion: Real-Time 3D Reconstruction and Interaction using a Moving Depth Camera. *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 559–568.

Julier, S.J. and Uhlmann, J.K. (1997). New extension of the Kalman filter to nonlinear systems. In: *Signal processing, sensor fusion, and target recognition VI*, vol. 3068, pp. 182–194. International society for optics and photonics.

Kaess, M., Ranganathan, A. and Dellaert, F. (2008). ISAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378.

Kalman, R.E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45.

Kazhdan, M., Bolitho, M. and Hoppe, H. (2006). Poisson Surface Reconstruction. *Proceedings of the Symposium on Geometry Processing*, pp. 61–70.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In: *Mixed and Augmented Reality*, pp. 225–234. IEEE.

Lorensen, W.E. and Cline, H.E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. In: *ACM SIGGRAPH computer graphics*, vol. 21, pp. 163–169. ACM.

Low, K. (2004). Linear Least-squares Optimization for Point-to-plane ICP Surface Registration. *Chapel Hill, University of North Carolina*, vol. 4, pp. 2–4.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous robots*, vol. 4, no. 4, pp. 333–349.

Martin, J. (2017). 3D Reconstruction using the Kinect Fusion. Available at: `https://gitlab.com/pleased/3d-reconstruction`

Maybeck, P.S. (1982). Stochastic models, estimation, and control. vol. 3.

Mei, C., Benhimane, S., Malis, E. and Rives, P. (2008). Efficient homography-based tracking and 3D reconstruction for single-viewpoint sensors. *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1352–1364.

Microsoft (2018). Microsoft Kinect. Accessed: 2018-04-16. Available at: `https://developer.microsoft.com/en-us/windows/kinect`

Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 593–598.

Murphy, K. (2000). Bayesian map learning in dynamic environments. In: *Advances in Neural Information Processing Systems*, vol. 12, pp. 1015–1021. ISBN 10495258.

Nettleton, E.W., Gibbens, P.W. and Durrant-Whyte, H.F. (2000). Closed form solutions to the multiple platform simultaneous localisation and map building (SLAM) problem. *Proceedings of the International Society for Optical Engineering*, vol. 4051, pp. 428–437.

Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohli, P., Shotton, J., Hodges, S. and Fitzgibbon, A. (2011*a*). KinectFusion: Real-time dense surface mapping and tracking. *10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011*, pp. 127–136.

Newcombe, R.A., Lovegrove, S.J. and Davison, A.J. (2011*b*). DTAM: Dense tracking and mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2320–2327.

Rao, C.R. (1945). Information and the accuracy attainable in the estimation of statistical parameters. *Bulletin of the Calcutta Mathematical Society*, pp. 81–91.

Remondino, F. (2003). From point cloud to surface: the modeling and visualization problem. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXIV, pp. 24–28.

Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. *European conference on computer vision*, pp. 430–443.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. *Proceedings of the International Conference on 3-D Digital Imaging and Modeling, 3DIM*, pp. 145–152.

Sarbolandi, H., Lefloch, D. and Kolb, A. (2015). Kinect range sensing: Structured-light versus time-of-flight Kinect. *Computer Vision and Image Understanding*, vol. 139, pp. 1–20.

Schoenberg, I.J. (1964). Spline functions and the problem of graduation. *Proceedings of the National Academy of Sciences*, vol. 52, no. 4, pp. 947–950.

Smith, R., Self, M. and Cheeseman, P. (1987). Estimating uncertain spatial relationships in robotics. *IEEE International Conference on Robotics and Automation*, vol. 4.

Smith, R.C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68.

Steinbrücker, F., Sturm, J. and Cremers, D. (2011). Real-time visual odometry from dense RGB-D images. *Proceedings of the IEEE International Conference on Computer Vision*, pp. 719–722.

Stewart, J. (2007). *Calculus*. Cengage Learning. ISBN 9780495011606.

Strang, G. (2016). *Linear Algebra*. 5th edn. Wellesley-Cambridge Press.

Sturm, J., Engelhard, N., Endres, F., Burgard, W. and Cremers, D. (2012). A benchmark for the evaluation of RGB-D SLAM systems. *IEEE International Conference on Intelligent Robots and Systems*, pp. 573–580.

Terzopoulos, D. (1988). The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 417–438.

Thrun, S. (2005). *Probabilistic Robotics*. MIT Press, Cambridge, Mass. ISBN 9780262201629.

Thrun, S., Liu, Y., Koller, D., Ng, A.Y., Ghahramani, Z. and Durrant-Whyte, H. (2004). Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716.

Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. *Sixth International Conference on Computer Vision*, pp. 839–846.

Triggs, B., McLauchlan, P.F., Hartley, R.I. and Fitzgibbon, A.W. (1999). Bundle adjustment modern synthesis. In: *International workshop on vision algorithms*, pp. 298–372. Springer.

Weil, W., Hug, D., Baddeley, A., Capasso, V., Bárány, I., Villa, E. and Schneider, R. (2006). *Stochastic Geometry: Lectures given at the C.I.M.E. Summer School held in Martina Franca, Italy, September 13-18, 2004*. Lecture Notes in Mathematics. Springer Berlin Heidelberg. ISBN 9783540381754.

Whelan, T. and Kaess, M. (2012). Kintinuous: Spatially extended Kinectfusion. *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*.