3D Reconstruction of Hands

May 2018

Abstract

This final year project is undertaken with the ultimate aim to design, implement and evaluate a system that reconstructs the human hand within 3D space, from a dataset of RGB images. This paper will first outline the project motivation and scope, before categorising its fundamental and secondary objectives. A discussion of recent and relevant literature will then be included, allowing me to extract useful information and methodological techniques to inform my own project design. This design will be outlined in detail, discussing any areas of difficulty and changes of design which these necessitated during the subsequent development process. Finally, the project will be evaluated using the technique which I have found to be most successful and pertinent: a comparison of the per-vertex distance between a mesh reconstructed by my system, and its corresponding ground-truth model. This evaluation will then inform the conclusion of this paper, which will also enter a discussion of hand reconstruction within the wider area of 3D reconstruction and of computer science as a whole.

Contents

1 Introduction **3**
- 1.1 Project Motivation . . . 3
- 1.2 Project Scope . . . 4
  - 1.2.1 Fundamental Objectives . . . 4
  - 1.2.2 Secondary Objectives . . . 5
- 1.3 Similar Projects and the State of the Art . . . 6
  - 1.3.1 Hands and Computer Vision . . . 6
  - 1.3.2 3D Reconstruction . . . 6

2 Specification and Design **6**
- 2.1 Specification . . . 6
  - 2.1.1 Feature Detection and Matching . . . 7
  - 2.1.2 Structure from Motion . . . 7
  - 2.1.3 Multi-View Stereo . . . 8
- 2.2 Design . . . 9
  - 2.2.1 Technologies . . . 9
  - 2.2.2 Capturing Input . . . 10
  - 2.2.3 The Pipeline . . . 10

3 Development Process **12**
- 3.1 Development Methodologies . . . 12
- 3.2 Feature Matching and Structure from Motion . . . 13
- 3.3 Multi-View Stereo . . . 14
- 3.4 Optimising for Structure from Motion . . . 14
- 3.5 Hand Keypoint Detection . . . 15
- 3.6 Hand Keypoint Triangulation . . . 17
- 3.7 Full Hand Reconstruction . . . 18
- 3.8 Mesh Reconstruction . . . 20
- 3.9 Mesh Texturing . . . 23
- 3.10 Overcoming the Problems . . . 23
  - 3.10.1 Locating Hand Keypoints in a 3D Space . . . 24
  - 3.10.2 Utilising Multi-View-Stereo Depth Maps . . . 24
  - 3.10.3 Backprojection through the 3D Mesh . . . 24
  - 3.10.4 Isolating the Model . . . 25

4 Evaluation Experiment and Results **25**
- 4.1 The Experiment . . . 25
- 4.2 Results . . . 27

5 Future Scope and Conclusions **28**
- 5.1 Future Scope . . . 29
- 5.2 Conclusions . . . 30

1 Introduction

The aim of this project is to utilise the latest 3D Computer Vision techniques in order to design and develop a 3D hand reconstruction system. The system will take a dataset containing RGB images of human hands as input, generating as output the 3D reconstructed model of the hand, ready to be used in its application area.

The main focus of the report will be on utilising the techniques discussed in the Literature Review to generate a 3D mesh of the reconstructed hand that has accurate geometry. Then, I will see if I can extend the usability of the outputted model, automatically texturing, rigging and skinning it to meet demands in the area of computer animation for games or TV.

1.1 Project Motivation

Many 3D modelling artists agree that modelling human hands is one of the most difficult parts about modelling a human. "The biggest challenge is to get the hand, to look like a hand." [5] This is due to their small size and complex shape, with creating a realistically proportioned hand being the most crucial step to success. This is the reason that a large number of cartoon characters are drawn with 4 fingers on each hand instead of the natural 5; they are simply easier to draw. [3]

Figure 1: A 4-fingered character from The Simpsons
[3]

Art historians claim that in old portraiture, you can tell how wealthy a person was based on whether their hands are visible in the painting. If the hands are hidden, then the customer could not afford to pay the amount it would cost, which was sometimes as much as the painting alone, as the artist would rarely paint such intricacy for a low price.

These examples show that it's very easy for humans to make mistakes when recreating hands (whether modelling, drawing or painting), and these mistakes, even if very small, can make the final product look very unrealistic. Conversely, it is much easier for a computer to get the crucial parts, such as the hand proportions, correct, even if there are little imperfections elsewhere. The most important objective of my project is to be able to recreate a realistic looking hand. This means it must be scaled as accurately as possible to the subject hand being modelled, with some leniency towards minor imperfections such as spikes or bumps, although minimising these would, of course, be preferable. Ideally, I would like the outputted 3D model of the hand to be accurate enough that it is distinguishable from an output that is reconstructed from a separate subject's hand.

A further part of a 3D character modeller's job is usually to rig and skin the model they have created. Rigging a model involves applying a virtual skeleton. This skeleton is made up of bones with joints binding them together, these joints usually have constraints stating where they can and can't move [9]. When skinning a rigged model, certain faces and vertices of the mesh are applied to a part of the skeleton: allowing the 3D model to move when the skeleton moves. Once this is done, an animator can take over and start working straight away to apply animations to the skeleton. Therefore if the system is able to output a hand that is, firstly isolated from its background, secondly rigged with a virtual skeleton, and finally complete, with it's mesh skinned to the skeleton, then it would drastically reduce the amount of

work a 3D modeller would have to do with the output, and the file could be passed to an animator much sooner, rendering the entire process more efficient.

1.2 Project Scope

Having discussed the objectives of the project as a whole, I must begin to define the scope for the project in order to complete these objectives. Due to a finite time limit on the project, my objectives must be categorised as either primary or secondary, with the former being fundamental to the successful use of the project system. Secondary objectives, while not crucial, will contribute to the project being a more extensive tool for its motivated application.

1.2.1 Fundamental Objectives

Most 3D reconstruction projects follow a similar generic pipeline, as follows:

1. Obtain a dataset of the scene.
2. Compute the camera rotation and translation (poses) for each view in the dataset.
3. Use the camera poses to compute a dense point cloud of the scene.
4. Reconstruct a mesh from the dense point cloud.
5. Map a texture onto the mesh using the dataset images.

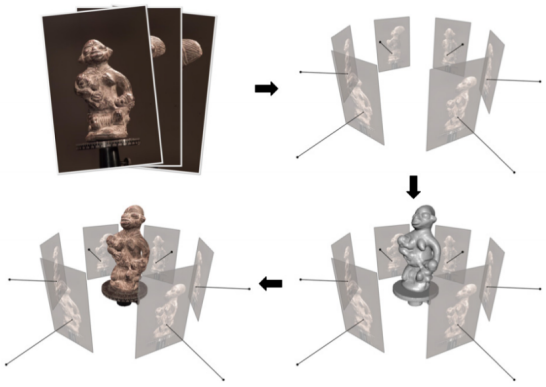


Figure 2: The typical 3D reconstruction pipeline; shows in order steps 1, 2, 4 and 5 [23]

I will be attempting to implement the same pipeline and I am considering everything up to step 4, the reconstruction of the mesh, as fundamental to the successful use of the project system. With this in mind, these are the fundamental objectives for this project:

- As is the first step in the typical pipeline, I first will need to design a user-friendly method for capturing the input dataset of the subject hand. While being the first objective, this is something that I will most likely have to return to and refine constantly as I progress through the project, evaluating as I go.
- Once an input dataset has been captured, the system must then be able to detect and match a large number of features across as many views as possible. This is the first step in the pipeline and its success may rely on the success of the method for capturing the dataset.

- Once completed, I must be able to compute the camera poses for every image in the dataset. As discussed in the Literature Review, this is a crucial step in the reconstruction process. Knowing the camera poses will also be extremely useful for accomplishing secondary objectives such as locating the bones and joints of the hand in the 3D space.

- With the camera poses calculated, the system must be able to utilise this information and generate a dense cloud of 3D vertices, which correspond to matching pixels across the numerous images. Completing this objective will provide a partial completion of the project, for the cloud of vertices already represents a recognisable reconstruction of the hand.

- However, the reconstruction of the hand in the form of a dense point cloud will most likely contain errors and be noisy. A point cloud is also not very useful for the aforementioned applications. Therefore, the next objective is for the system to generate a mesh from this point cloud. By accomplishing this fundamental objective, I will be removing a lot of the noise that was existent in the point cloud; making it more closely resemble human skin. The output now will satisfy a large area of the motivated problem, though greater functionality can still be added to the project in order for the output to be more extensively useful.

- Once the above objective is complete, the project system is able to be evaluated. Since the main focus of the system is to accurately recreate the shape, scale and proportions of the hand; this will be the main area of evaluation. This evaluation will be composed of comparisons between a ground truth 3D mesh of a hand, and its counterpart reconstructed by the project system.

1.2.2 Secondary Objectives

The typical 3D reconstruction pipeline goes beyond recreating the mesh of the scene, and will generally also map a texture onto it as seen in step 5. I will be considering this a secondary objective, since there exist many applications where it is just the shape of the hand required without the need for texture. All of the following are secondary objectives of this project for a similar reason:

- As just mentioned, the texture may not be a required output. For example, if the application of the system is for a 3D character artist, then they could want to create their own texture which matches the rest of the character.

- With the completion of all fundamental objectives, the outputted mesh of the hand may also contain parts of the background within the reconstruction. Therefore, an added objective is for the project system to be able to detect and isolate the hand model. The completion of this objective could prevent a scenario where users not experienced 3D modelling, could be forced use 3D modelling software with which they are not proficient, in order to manually isolate the hand.

- Many of the previously mentioned applications would benefit from the hand being rigged with a virtual skeleton and skinned automatically by the project system. This will also be a non-fundamental objective of the project, as there are many applications such as analysing and comparing the hands of separate individuals where the bone structure is not needed and the hand might not need to change pose.

- Since these secondary objectives involve texturing, rigging or mesh removal, they all affect the final output of the system. Therefore, as a non-fundamental objective, I will also attempt to evaluate or comment on the success of these objectives.

1.3 Similar Projects and the State of the Art

1.3.1 Hands and Computer Vision

In my Literature Review, I searched for projects which also reconstructed the hand and could find none. However, there exist many projects for tracking hands that have to go through similar stages. For example, in a paper by Mueller et al., [18] 20 sparse landmarks on the hand are used, and by means of a stereo-vision system, the location of these points within 3D space are detected. With so few landmarks, their algorithm retained a good real-time performance too. Many hand tracking projects must also use inverse kinematics in order to estimate the location of certain joints or bones in the hand if they are hidden or obstructed by an object. Like my own project, the work by Mueller et al. also necessitated the separation the hand from its surrounding background. For Mueller et al., however, this separation was needed to facilitate the tracking process, whereas my own project must separate the background for it is not wanted in the final reconstruction. Their method was to use the k-means clustering algorithm in order to distinguish the colours of the hand from the varying colours of the background. This is something my own project should definitely utilise. However, unlike most hand tracking projects, I am prioritising accuracy over performance as my application is not required to run in real-time.

In terms of human body-part construction, scanning hands involves many of the same difficulties as scanning faces: a much more popular area of research. Similarities lie in the process, as input images are taken over a period of time, in which poses are likely to change as muscles tense and relax. State-of-the-art equipment uses multiple cameras, usually in a ring or semi-dome formation. This way every camera can take an image at the exact same time and there isn't any inconsistency with the target object's pose while images from each viewpoint are obtained.

1.3.2 3D Reconstruction

The core of this project relies on recreating a dense point cloud from the input images and camera poses using Multi-View Stereo techniques. Multi-View Stereo techniques have played an extremely important role in the field of 3D reconstruction [13]. Many projects exist parallel to my own. These projects are usually more generalist in nature; they test and experiment with Multi-View Stereo algorithms to reconstruct either small objects on a table or large independent structures using just images.

Many state-of-the-art multi-view stereo systems that are designed for object 3D reconstruction (instead of scene reconstruction) utilise convolutional neural networks [20] in order to integrate information about the object being reconstructed into the reconstruction pipeline (much like humans do when we see a single image of an object, we know the shape in 3D).

The state-of-the-art technology in 3D reconstruction does not use simple RGB images or video as input. I have chosen to utilise this type of data because this ensures that the project is accessible to the greatest possible amount of people, with few hardware constraints. However, projects like The Digital Michelangelo Project [15] are creating 3D reconstructions of large statues and sculptures with extreme accuracy. They're scanning the objects using high-quality hardware including laser triangulation rangefinders, laser time-of-flight rangefinders and digital still cameras. Currently, their reconstructions are occupying around 250 gigabytes of data.

2 Specification and Design

2.1 Specification

In my Literature Review, I analysed multiple techniques for reconstructing objects from RGB input. In this section of the report, I will go into detail on how some of the findings from my Literature Review will be applied to completing the objectives which I have outlined.

2.1.1 Feature Detection and Matching

Feature detection and matching is an essential step towards calculating the depth of the object features from every view. By detecting and matching features between views we are then able to use these matched features to calculate the pose of each camera view in the input dataset, using a technique called Structure from Motion.

If the algorithm used to detect features does not perform well, then there will be fewer features detected per view and therefore, a smaller probability that enough features will match between any two given views. This can result in some views being discarded because not enough features were matched with any other views and therefore the view pose could not be calculated using Structure from Motion. Every view that gets discarded diminishes the usable information for computing the dense point cloud, and therefore diminished detail in the final model.

When taking input images, variables like the distance between the camera and the object, the illumination of the object and the rotation of the camera are likely to change significantly. For this reason, the algorithm needs to be scale and rotation invariant. Scale-Invariant Feature Transform (SIFT) [16] is an algorithm that meets these criteria and yields very good results, often used for "object recognition, image stitching, visual mapping" [19]. SIFT is rather computationally intensive, however, especially when compared to other feature detection algorithms such as ORB which performs up to 40 times faster, but is not scale-invariant. For my project, which prioritises accuracy over performance, the longer computation times are a necessary sacrifice in order to ensure that every view in the input dataset is used for the dense reconstruction. However, the main focus of my project would be to capture every angle of the hand, so I shouldn't be varying the scale of input if this can be avoided. Therefore, ORB is a possibly favourable option if I need to reduce my computation time. This will, however, require a more careful approach when capturing my input dataset.

(a) ORB (b) SIFT

Figure 3: A comparison of matched features between ORB and SIFT
[2]

2.1.2 Structure from Motion

Structure from Motion is the technique that uses matched features from multiple views, in order to simultaneously calculate the pose of the camera as well as the 3D structure of the view. Executing this technique will provide me with the required camera poses, thereby completing my third fundamental objective.

The Structure from Motion techniques are usually divided into 2 classes: sequential, where the views are treated incrementally, or global, where all views are considered simultaneously. Sequential techniques for Structure from Motion usually start by computing 2 or 3 views, before sequentially adding new views into the existing reconstruction, using Bundle Adjustment to finely tune the eventual output and minimise the reprojection error. When the views are treated sequentially like this, the outputted extrinsics can be "subject to drift" [17]. An accumulation of drift could mean that, while attempting to close a loop in the camera trajectory, extracted features at the end of the trajectory are not assimilated with those at the

start of the trajectory. This can result in duplicated sections of the 3D scene. Depending on the method chosen to reconstruct the hand, this could affect the outcome of my project immensely. Alternatively, global methods distribute any error between all views, drastically reducing the probability of the same occurrence. Due to the nature of my project, I estimate that the datasets used will have a high chance of loop-closing occurring, therefore global methods of Structure from Motion are much better suited to my project. The general steps are as follows:

1. Matching view pairs are collected and the essential matrices are computed for each. For 2 views, the essential matrix relates corresponding points; defining the epipolar geometry between them [11]. A graph is constructed of all the epipolar relations, called an epipolar graph.

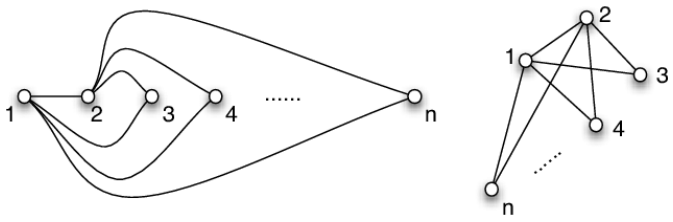


Figure 4: Illustrated epipolar graph
[6]

2. The global rotations are computed for the graph by performing a method of statistical inference called Bayesian inference.

3. The relative translations between the views are computed from the tri-focal tensors. The tri-focal tensor captures the geometric relations between 3 views, just as the fundamental matrix did for 2 views.

4. These translations are then registered in a global coordinate frame, instead of a relative frame.

5. The sparse triangulated structure of the scene is computed; this structure, the view translations, and the view rotations are all adjusted and finely tuned using bundle adjustment.

This global pipeline is by far preferable to the sequential pipeline for reasons further to that of reduced drift. The epipolar graph means that more than only 2 views are used for the computation, this results in a smaller risk of false epipolar geometries merging. Global methods are also preferable due to their not relying on the use an initial pair, whereas the success of sequential methods often depends heavily on the success of the reconstruction of this initial pair, which is refined using the other views and bundle adjustment.

2.1.3 Multi-View Stereo

I now return to another fundamental objective: taking the camera poses from Structure from Motion, and using them to generate a dense cloud of vertices. I will be doing this using Multi-View Stereo techniques. These techniques do not rely on feature detection and matching, but instead try to locate the 3D position of every pixel in each view by computing the projected geometry between multiple views (this projected geometry is known as the epipolar geometry). This process of locating the 3D position of particular points through the use of epipolar geometry, can be encompassed by the broader term, triangulation. [1].

To summarise this technique, imagine taking one of the views and selecting a point in that image, with this point being X_L (see Figure 5). Now, imagine plotting a line from the camera centre of the view (O_L in Figure 5) through X_L. If we now take a second view of the same scene and plot a line between the 2 camera centres, the plane between these two lines is the epipolar plane. We can now state that

the corresponding point in the second image must lie on the intersection of this epipolar plane and the projection of the second view, shown on the Figure as a red line. This constraint is known as the epipolar constraint, and can be described for a point X by a matrix known as the fundamental matrix.

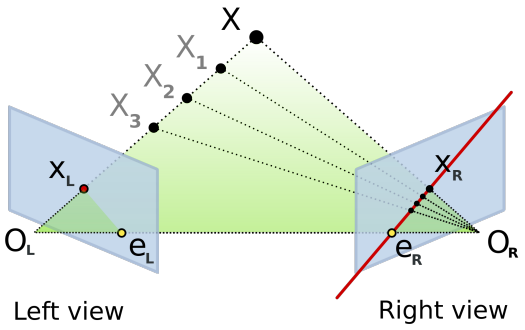


Figure 5: Illustrated epipolar geometry
[4]

When searching for a matching pixel or point between two views, this technique drastically reduces the amount of search space from the entire image of neighbouring views, to just one line. Once the corresponding point's location on the line is calculated, then we have everything needed to calculate the depth of that point. If we are to do this with all the pixels in the same view, then we can produce a depth map for the view containing the depth information for every pixel. This is the first step of general Multi-View Stereo methods, and it will normally also produce a score to measure the confidence of the decided depth so that any outliers can be removed later in the process. Once the depth maps for each view are retrieved, then the next step is to fuse these depth maps together. Fusing depth maps is not a problem linked only to Multi-View Stereo algorithms. Depth maps also need to be fused when using RGB-D sensors to reconstruct 3D objects, as their output will be very similar to that of Multi-View Stereo methods, though probably more accurate.

A method of fusing depth maps suggested by Curless et al. [12] utilises both the set of depth maps, as well as the corresponding confidence and score maps, to identify and subsequently resolve conflicts between depth maps, it can be summarised as follows:

- To start, the most central depth map is taken as the reference view.
- Other surrounding depth maps are then incorporated into the reference view.
- When multiple depth maps project their values onto a single pixel, the nearest depth is used.

2.2 Design

When it comes to the design of the project system, I will be expecting to perform a large amount of evaluative analysis, midway through the development process. This will likely result in the redesign of certain aspects of the system. In light of this evaluative methodology, it could be that my final product resembles nothing that exists in the initial design. However, should any part be redesigned, the reason for doing so will be clearly stated in the development section.

2.2.1 Technologies

I intend to program a pipeline in C++; consisting of all the techniques which I have thus far outlined, in order to get the desired final output, which should be a 3D model of the hand. I have chosen C++ for the pipeline for several reasons. Firstly, there is a bigger choice when it comes to supported Computer Vision

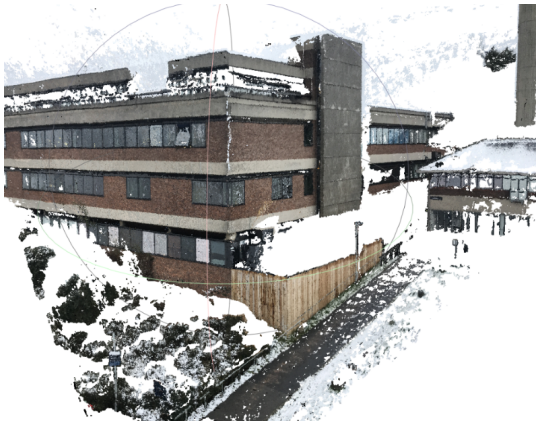


Figure 6: An example dense point cloud

libraries. Originally, I was going to use Python as it also supported many of the same Computer Vision libraries, but during some experimenting, I attempted to implement some lower level functionality and, due to the nature of python as a higher level language, I found it to be unsuitable to my project. While trying to extract intensity information from every pixel of one image, I realised that iterating through every pixel was taking far longer than it would do in C++. After some further research, I determined that Python would be inefficient in the aspects of the system which would require lower level functionality, due to the number of internal methods having to be called for at any time a pixel is read. Therefore, I deemed C++ to be a more apt choice, since many aspects of my design will need to access lower level functionality, for instance, when performing triangulation or texture mapping.

I will require software to view the output 3D meshes and point clouds. For this I have chosen MeshLab, as it provides the basic functionality I require, without the steep learning curve of another option: Blender. However, if I must eventually apply some more complex manipulation to any of my results, for instance during the evaluation, then Blender may be a good choice.

2.2.2 Capturing Input

The input of the pipeline should be the path to a directory containing two futher directories. Both directories should contain a dataset of images with the subject's hand in view, which are in a format by which the EXIF data is preserved and stored with the image, for example, JPEG. In the first directory, the subject's hand should be spread out, face down, and on a flat surface that is of a different colour to the hand. The camera should then try to capture every angle from above the hand. This can be done any way that the user likes, since the images will be treated globally, rather than sequentially. However, when capturing the input, the user needs to ensure that there are enough images so that the described features from each view will have matches by means of which they can be linked to other views. During the development phase of the project, I will attempt to come to an optimal number, however, this ultimately depends on the quality of the camera and of the images taken. The second directory should be fairly identical to the first, with the exception that the palm will be facing upwards. By capturing the same number of photos of both the up and downside of the hand, I can ensure consistent output of both sides.

2.2.3 The Pipeline

The initial design process for the main reconstruction pipeline implemented in C++ is as follows:

+ Locate the first dataset, this can be the top or bottom half of the hand. Identification of either the top or bottom of the hand can be specified by directory name or by command line input. The distinction between the top and bottom of the hand, will be crucial when attempting to align the two halves to build a complete reconstruction.

+ Retrieve the dataset and extract the EXIF data: This data contains the intrinsics such as the focal length of the camera that captured the images. These intrinsics are needed when performing techniques like Structure from Motion or Multi-View Stereo further down the pipeline.

+ Detect and describe the features in each image using the SIFT feature detection algorithm. If using the SIFT algorithm takes far too long, or if I just want to improve the overall performance, then I will test the pipeline with ORB also. The problem that the feature detector must overcome is that the skin on the hand is generally very smooth, which can mean feature detectors struggle to find points that it can track since they look for sharp edges and corners of contrasting colours, which the typical hand does not possess.

+ Compute the matches of corresponding features between each set of feature descriptors.

+ Using a technique for global Structure from Motion, the feature matches and the camera intrinsics, calculate the camera poses in each view as well as the sparse structure of the scene. The sparse reconstruction outputted by this step should be stored for debugging purposes, but will not be needed any further. In terms of potential debugging purposes, the sparse reconstruction may be needed if the final reconstructed model contains large errors or imperfections, since, in this scenario, it could be useful to check the sparse reconstruction output from this step to determine whether the issue exists before or after this point.

+ Using a method for Multi-View Stereo with the input images, their intrinsics, and the poses computed from Structure from Motion (extrinsics), calculate depth maps for every view, before fusing the depth maps to create a dense point cloud of the specified side of the hand. The vertices in this point cloud will have a colour attribute that is taken from the pixel colour in the view images. Therefore, this dense point cloud could be considered a 3D reconstruction of one half of the hand. However, this is not the final product.

+ Having reconstructed half of the hand, the other also needs undergo an identical process. By this stage, two dense point clouds will have been obtained, each containing half of the hand in a similar pose. Ideally the poses would be identical, however, I am estimating that this is almost impossible in practice, and hence, have accepted that the hand-halves will hold slightly different poses. In the following steps, these two point clouds must be aligned into one complete reconstruction.

+ The method I am using to merge the point clouds will require first rigging and skinning both sides of the hand, then aligning them by aligning their respective skeletons. In order to do this, we need to be able to detect where the keypoints of the hand lie in the 3D space of the point cloud; these will be the joints. Hand tracking is a vast area of Computer Vision, and crucial to the success of hand tracking is the detection of the keypoints of the hands from 2D images. I will therefore test methods of detecting the keypoints in order to determine which yields the most promising results.

+ I will take all the input images that have a corresponding view with its pose calculated, and detect the hand keypoints in the image. Once detected, I will need to calculate the position of the keypoints in 3D space. If I take one keypoint of the hand, such as the index fingertip for example, across multiple views, then I can calculate the 3D location of this point by utilising the corresponding poses for these views and triangulating the point across them, using the optimal views in order to minimise error as far as possible.

+ At this stage, the 3D location of each keypoint in the hand will have been obtained. All that remains is to create the virtual skeleton by specifying connections between keypoints, which will serve the function of bones, and to specify joint constraints to assure the range of flexibility afforded where these bones meet.

+ At this point it will be necessary to separate each half of the hand from its background, this can be done by utilising the location of the hand keypoints to get a general location of the hand and each digit. The keypoints also allow us to extract an average skin colour for the subject's hand. One method of utilising this information could be to remove all vertices that are not within a certain colour range that roughly matches the colour of the user's skin while keeping all vertices around the fingernail area so that the colour variety of that area, and the detail that this brings, are not lost. In the case that this

still fails to successfully remove all the vertices of the scene around the hand, a condition could also be implemented, stating that any vertices that are not within a given radius, r, of the rigged hand, are also removed.

+ In order to align the two point clouds, the pose of one half must be adjusted to match that of its counterpart, assuming the pose of the two halves will be slightly displaced. For the vertices in the cloud to move when the rig of the hand is re-positioned, they must be skinned to the rig. This process involves assigning, or weighting each vertex to either a joint or a bone. Now if the two rigs are altered to fit each other, whether scaled, joints rotated or translated; the points or vertices will move with the rig and the point clouds should align.

+ By this stage, we should have a dense point cloud that contains vertices for both the top half and the bottom half of the reconstructed hand, with the background having been extruded. This dense point cloud should also be rigged with a final virtual skeleton and its vertices should be skinned to the rig. However, the final output of the system should be a 3D mesh as opposed to a dense point cloud. There are many techniques for this that will result in very similar outcomes. I must instead, consider certain variables that will adjust the resolution or bumpiness of the final mesh, like for example the distance between two vertices required in order for them to be considered as distinct. Increasing this distance will result in a smoother mesh because minor displacements between adjacent vertices will be less noticeable the further the vertices are from each other.

+ It is possible that after the mesh is recreated, the rig will have to be skinned to the mesh. Unless a way to incorporate the skinned vertices from the point cloud into the outputted mesh can be uncovered. If the mesh does indeed have to be skinned, then it will be done with an almost identical method to how the dense point clouds were skinned, except that faces may have to be taken into account as well as vertices.

+ The input images and the camera extrinsics for each view may now be utilised in order to texture the 3D mesh. This should work similarly to the way in which the final colour of each vertex was decided when generating the dense point cloud. For each view, rays should be cast from the camera centre through the projection image at every pixel. Where the ray meets the mesh, the colour of the pixel should be mapped to the mesh. Where rays cast from multiple views meet the mesh at a single point, the outputted colour to be mapped should be the average value between.

+ At this point, if successful then the user should be outputted a fully textured, rigged and skinned 3D mesh of the subject's hand.

3 Development Process

In this section of the report, I will outline the development process during the project. Many of the algorithms and popular techniques used in the development process have already been discussed in the Specification section. Here, changes to the design and other certain algorithms will be discussed in detail, concurrent with a discussion of the progression of results achieved.

3.1 Development Methodologies

Due to the nature of this project being mainly research-based, no outright suggested development methodology has been suggested or, consequently, followed. However, the stance which I plan to take during development would most likely resemble the Agile Scrum Methodology if one must be chosen. I am planning to develop a pipeline with many individual challenges along the way, therefore each one could be considered a separate sprint, as they would each require planning and design before, the development of the software and then the testing and evaluation of the component before moving onto the next sprint. My methodology will vary from the classic model, since I estimate that each sprint will be around a week long, instead of the typical 2 weeks. The difference between my own project and a typical Agile Development project is that I will be working alone, rather than as part of a SCRUM team. This means

there that I would be my own scrum master for the project, setting new goals and encouraging workflow in an auto-reflective manner. However, since the scrum meetings outline by the methodology are designed to synchronize the work of team members, and as a result, I am unsure as to the comparative relevance of this methodological model for my own organisation of this project. [10].

3.2 Feature Matching and Structure from Motion

After parsing the input and output directories provided as arguments, the reconstruction pipeline can begin. For the area of Feature Matching and Structure from Motion, I utilised the OpenMVG library [17]. This library provides many of the tools needed to implement Structure from Motion techniques and prevents me from having to implement already existent algorithms, hence saving a significant portion of time. This a vital consideration with a project of this scope so that I have time to complete as much as possible in a constrained time period.

The first step necessary is to extract the EXIF data from the image files, and utilise it to compute the focal length of each view in pixels, using the following equation:

$$focal_{pix} = \frac{max(w_{pix}, h_{pix}) * focal_{mm}}{ccdw_{mm}}$$

Where $ccdw_{mm}$ represents the known sensor size of the camera in millimetres. Once calculated, the camera matrix can be constructed from the available intrinsics.

At this point, I use OpenMVG's *ComputeFeatures* binary executable in order to detect and describe the features in each image before computing each of the matches before using the output to call another executable *ComputeMatches*. By default, OpenMVG uses the SIFT feature detector which is excellent as it is what I had planned to use in my project design.

At this stage of the pipeline, the plan was to compute global Structure from Motion using OpenMVG's *ComputeGlobalSfM* executable. The executed code first detects and removes false relative pairwise rotations, before continuing to compute global Structure from Motion exactly as discussed in the Specification section: computing the global rotations and the relative translations between the views, before calculating the global translation, estimating the final structure and camera motion, while using bundle adjustment to refine the result. However, this stage would repeatedly fail for all current datasets of the hand which I have obtained. The images in these datasets consisted of the hand, face down and flat, as described previously, on a smooth table of a colour which is diffferent to that of the hand itself.

While debugging, I decided to detect and match the features in two adjacent frames from the same dataset using SIFT, the results can be seen in Figure 7.

As can be seen, a very limited number of features are detected on the hand, and these frames are able to see much more of the hand compared to frames that view the side of the hand for example. To test the theory that this was the halting factor, I decided to take a pen and draw dots all over the hand, at relatively evenly spaced intervals. I knew that the feature detector would be able to pick out the corners and odd shapes of these dots much better than it could recognise bone structure and shadows in the skin. After testing the system up to this point once again, I was able to obtain many more features which could be matched between the views. As a result of this, I successfully computed the camera poses with Structure from Motion, giving me everything needed to start building the dense reconstruction section of the pipeline using Multi-View Stereo.

A further output of this, of course, is the current sparse reconstruction. I can use this sparse reconstruction to tell if the camera poses computed are adequately accurate to continue in the project process, since these poses are used to triangulate and generate the structure shown in Figure 8.

As can be seen in the figure, the structure very much resembles a human hand, with most of the drawn dots on the hand having been successfully placed into the 3D space. In light of this, the sparse point cloud is adequate and able to move on to the next implementation of the pipeline.

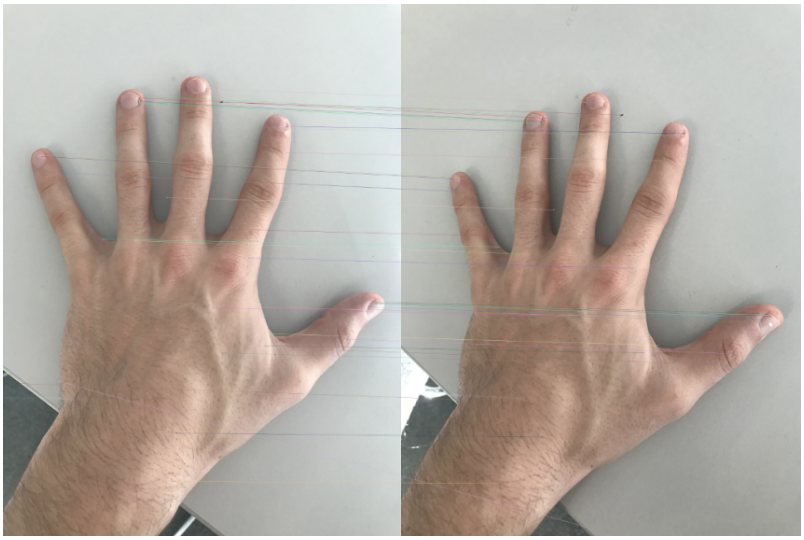


Figure 7: Matching points on the hand detected by SIFT

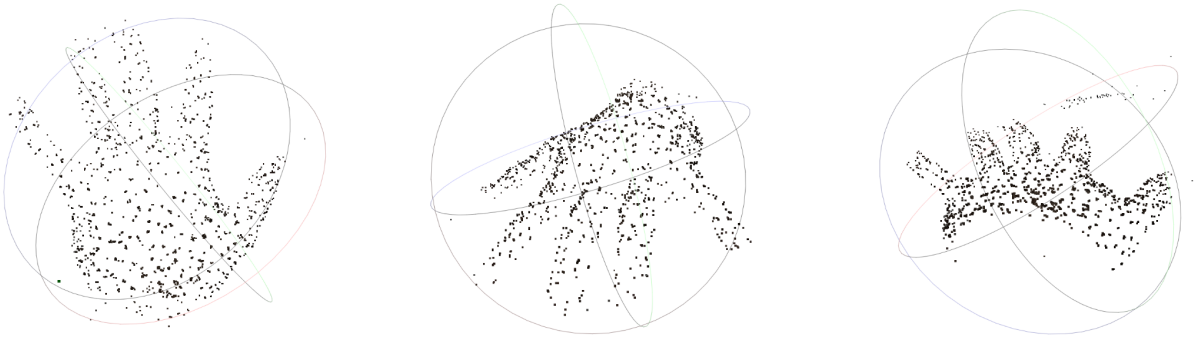


Figure 8: The sparse point cloud generated by Structure from Motion; viewed in MeshLab

3.3 Multi-View Stereo

The next step in the pipeline is to be able to utilise these camera parameters alongside multi-view stereo techniques. To aid with the implementation, I utilised the OpenMVS library for this [8].

OpenMVS contains an executable *DensifyPointCloud* which implements a Multi-View Stereo technique that closely follows Shen's paper on *'Accurate Multiple View 3D Reconstruction'*, [21], but also implements some original ideas. Using this method with the existing pipeline, I was able to reconstruct the dense point cloud shown in Figure 9, from the poses already computed.

The reconstruction looks almost perfect from the top down, this is mainly due to the vertices in the point cloud also having extracted the colour from the images. When we look at a cross-section of the fingers as seen in the far right image of Figure 9, or the side of the hand we can see the small imperfections that the method fails to eliminate. This, however, does not constitute a major problem and can be de-noised and smoothed when reconstructing the mesh.

In the shown figures, I have removed the background in order to get a clearer view of the hand point cloud, the background was white so I simply removed all vertices containing a certain colour range (around white) using MeshLab: a process which worked very well.

3.4 Optimising for Structure from Motion

Before moving on, I remained dissatisfied with the process, and particularly with the fact that dots had to be drawn onto the hand in order for an accurate reconstruction to be generated. My dissatisfaction

Figure 9: The dense point cloud generated by Multi-View Stereo, with the background manually removed for clarity

was an aesthetic one, since these dots affected the aesthetic outcome of the overall reconstruction. The original purpose of the dots on the hand was to ensure that enough features were matched between all the views so that no views would be discarded. This logically followed that the features would not have to be on the hand, but could instead be in the background. To test this theory, I ran the pipeline to see if Structure from Motion succeeded with no dots on the hand, instead with a sheet of paper with text behind, covering up the smooth table.

This time, much like when the hand was dotted, the camera pose was extracted for almost every view. See Figure 10 for a reference of just how many features were detected and utilised in the production of the sparse reconstruction outputted.

Figure 10: A sparse point cloud computed without dots drawn on the hand

Whilst this is promising, if I finish my implementation with some time to spare, I would like to come back and re-examine this. The time to compute Structure from Motion was significantly larger than the times seen in previous tests, due to the large number of features matched. I believe that I could get the same success by using something small and simple like a barcode somewhere in view. Or, I could detect all the features, but only utilise a limited number of them. This would be particularly promising if I could only utilise perhaps the best 100.

3.5 Hand Keypoint Detection

Now that my system is able to recreate a dense point cloud of half a hand, I must implement the rigging and skinning of this hand, as mentioned this could also be the solution to the question of removing the background since the hand, once skinned to a rig, is already separated from its background.

I will now utilise a library called OpenPose. OpenPose implements the techniques discussed in a paper written by Cao et al. [7] discussing the detection of 2D keypoints on the hand. The method proposed by

Cao et al. detects body parts using a two-branch convolutional neural network which can estimate both confidence maps and what is referred to as *Part Affinity Fields*, in order to associate body parts. In the process outlined, the algorithm then goes through a parsing stage in which it performs a set of bipartite matchings that associates body part candidates.

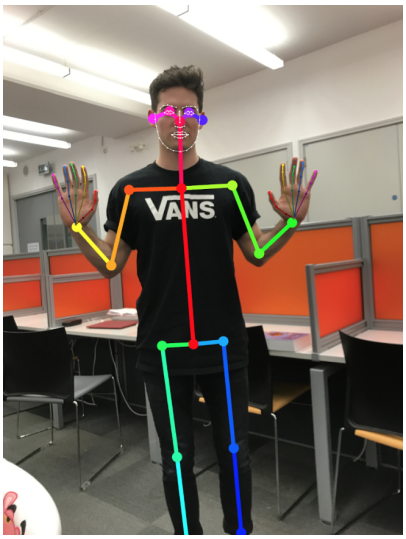


Figure 11: OpenPose performing full face, body and hand tracking (on me)

Whilst arguably very impressive, OpenPose as it was, only supported the detection of hand keypoints when also detecting the rest of the body, since it relies on the information from the rest of the detected body to more accurately detect and track the keypoints in the hand, allowing it to undertake tasks such as determining between the right or left hand. For the purposes of this project, this library may be used, but since I must be able to detect the keypoints solely in the hand, as the body also being visible is not an option for the way I want to capture my input images, its utility will be decidedly limited.

However, it is possible to write custom code for OpenPose and then rebuild the library in order to build the written code and make use of the functionality available in the library. The binary executable generated by the library can then be run from the library root. As far as I was able to determine, this is the only way to run custom code within OpenPose.

I managed to utilise this custom code implementation process to create a binary executable that detects just the keypoints in the hand with reasonable accuracy. Now I will utilise this in order to detect the keypoints in the hand for every view, an example of the results received are seen in Figure 12.

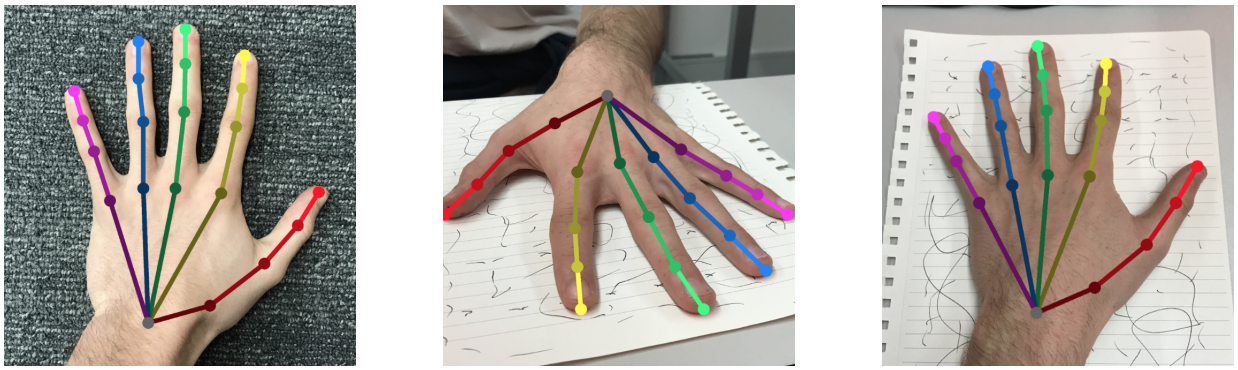


Figure 12: OpenPose tracking only the hand

3.6 Hand Keypoint Triangulation

With the keypoints detected in the 2D images, I now must compute their location in the 3D space relative to the rest of the reconstruction. As well as their locations in (x, y) coordinates, OpenPose returns a score for each keypoint, ranging from 0 to 1.

Now, based on the score and the locations of each keypoint, I should selectively choose the best views, before utilising them in an attempt to triangulate each keypoint. In order to determine the best views, I would first eliminate any instances of views with scores below 0.8.

Then in the 3D space, the remaining views would project a line from the camera centre through the 2D projection of the keypoint currently being located. It is extremely unlikely that any of the lines from each view will actually intersect as neither the camera poses the detected keypoints in the hand will be reliably accurate. For this reason, I will have to find the point in 3D space that is closest to the majority of projected lines (shown as a red dot in Figure 13).

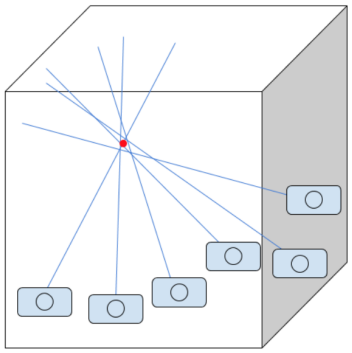


Figure 13: N view triangulation illustrated

For this problem, I can utilise the least squares solution that minimises the "sum of perpendicular distances" from the point in 3D space to every line in the set [22].

This is something for which I am able to utilise the OpenMVG library, as several triangulation methods are already implemented, one of which uses the least squares solution.

Before using the OpenMVG method, I must create a new *sfm_data* instance, the class of which is defined by OpenMVG. I then need to fill the instance with the required data for triangulation, this first requires extracting the camera extrinsics and intrinsics from the Structure from Motion step earlier, before parsing the JSON data and filling the new *sfm_data* object with this data.

The OpenMVG library offers methods for loading and saving data from files, however, they cause errors which I have traced back to the source code. This is a repeated issue with some of the libraries used in this project, since they are all written and updated either by one individual or by a small team, meaning that the necessary bug fixes are scarce. Due to the scope of this project, I was forced to halt my reparations of the libraries' source code, lest I should succeed, but the expense of my desired final product.

Once the *sfm_data* instance is filled with the existing intrinsics and extrinsics of each view, I can then add the 2D hand keypoints for each view as detected features. OpenMVG's triangulation methods can take this object and calculate the 3D structure from these features using the chosen method.

Shown in Figure 14 are the triangulated landmarks (large dark green boxes) placed with the corresponding sparse reconstruction for reference.

When compared to the level of accuracy I was expecting; this doesn't seem to have been entirely successful. The keypoints have instead clustered around the area of the hand, but it doesn't seem like they have achieved much more than this, having failed to land even close to their intended position. Through numerous tests on different datasets, I repeatedly, and disappointingly, achieved the same results.

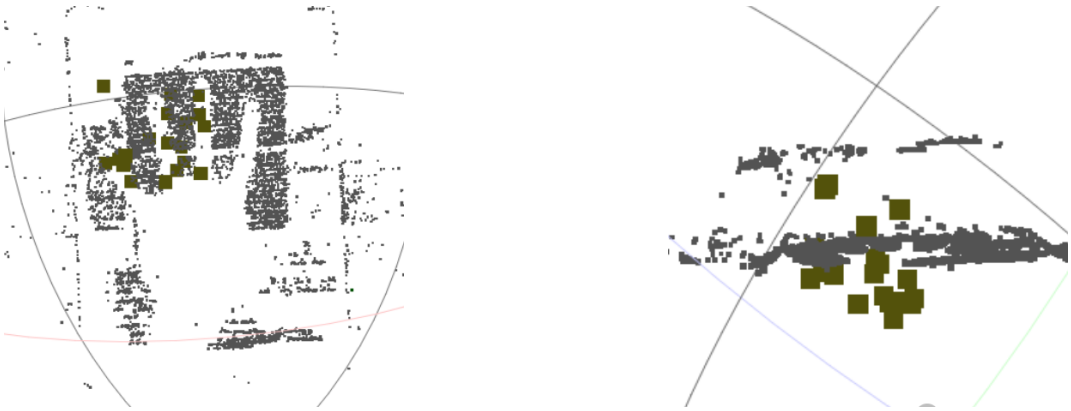


Figure 14: The triangulated hand keypoints

I now investigate what exactly is the cause of the lack of success which this triangulation method has brought, and if I am unable to improve it then I may be required to redesign the rest of the project system, since it relies so heavily on the 3D location of the hand landmarks. Without them, I am not able to obtain a dense point cloud, and subsequent reconstruction of the entire hand, as I would be unable to rig the two halves and line them up automatically. Thus, I would also lose the ability to automatically separate the hand from its background and rig and skin it. Essentially, I would be at a dead end.

In order to narrow down what is causing such error, I will attempt to triangulate one point with optimal conditions. Firstly, I will hand pick the two views to triangulate the point. These views must: have a score greater than 0.9 for the selected keypoint, and be viewing the keypoint from 2 angles which have at least a 45-degree difference. This should help to reduce the error and can give me an idea of how well I could locate the points on the hand if I were to improve the view selection.

Figure 15 shows the two views chosen, as well as the result of triangulating the tip of the middle finger.

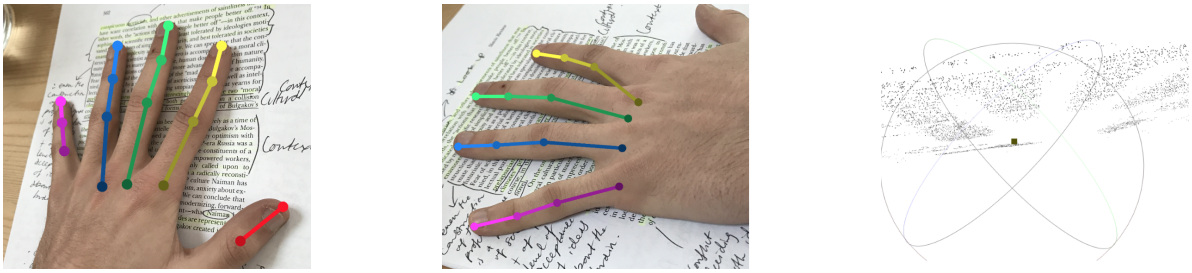


Figure 15: The manually selected views for triangulation and the triangulated point

The result is equally unsuccessful as earlier attempts, meaning that a redesign of the rest of the project system will be necessary to accommodate this unforeseen difficulty.

3.7 Full Hand Reconstruction

Due to the lack of success so far when triangulating the hand keypoints, and the time constraints of a project of this relatively small scope, I was obliged to find a way of reconstructing the full hand without the need to rig the two halves of the hand. One solution is to manually place the 2 point clouds together in 3D software (such as MeshLab, as I have been using), however, I would ideally like to avoid this, as it relies far too heavily on user involvement and input, and it also requires the user to be proficient with the 3D software, which is not reflective of the accessible system which I set out to make.

In light of this, I will experiment with a possible alternative, by attempting to get a full reconstruction

of the hand from 1 dataset alone. This originally was dismissed as an option, as the movement of the hand was too large of a problem. Originally the subject would rest their elbow on a table with their hand facing upwards and the camera would move around the hand, taking images of it. While the elbow remained stationary it would still allow the arm to pivot around it which let the hand move around the scene too much. The camera used to take the images was focused on the hand, with the background being out of focus and blurred. This combined with the movement of the hand meant that there were not enough consistently stationary features in the scene that could be detected and used for Structure from Motion.

I will attempt to solve this by gathering images using a different methodological approach. Firstly, to tackle the problem of movement of the hand, the hand will be positioned with the arm and wrist resting on a flat surface (most likely a table) of any kind. The hand should be held straight out and facing downwards, it should remain as still as possible, usually being required to settle for several minutes to ensure optimal stillness. The camera will be positioned in front of the hand, where the fingertips are pointing, facing the hand, and should then translate 360 degrees so that every angle of the hand is captured, shown in the diagram and sample input image of Figure 16.

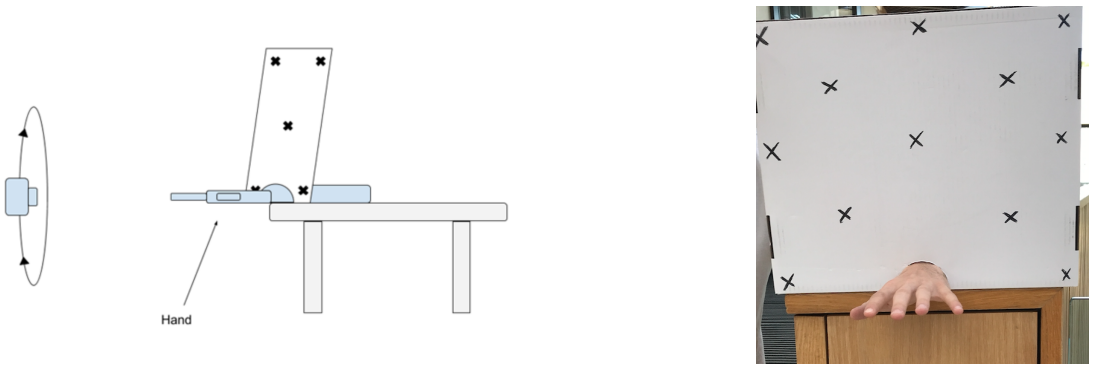


Figure 16: The ideal camera motion illustrated, and a sample input image

Behind the hand is a background with manually applied landmarks drawn on to it, and a hole for the hand to go through. This should give the camera landmarks to detect no matter what angle the hand is being viewed from. The overall scene now resembles the scanning of the front of an object with a protruding section, instead of rotating around an entire object.

After running the currently implemented pipeline with this setup, I was indeed able to perform Structure from Motion as the background landmarks were successfully detected from nearly every view. The output from performing Multi-View Stereo can be seen in Figure 17.

Figure 17: The dense point cloud computed from Multi-View Stereo

Certain pixels of each view were successfully triangulated in multi-view stereo. It seems the most success came from the wrist of the hand, and the least reconstructed parts are the fingers. I estimated

that this would be to do with the wrist being more supported and stationary, whereas the fingers were significantly more subject to their own movement, and therefore could not be consistently triangulated between views, with the points being counted as outliers.

I attempted this same process again, this time adding more landmarks on the hand in the form of dots, as I had done previously, in order to ensure that no views were discarded due to lack of matched features. The dense point cloud shown in Figure 18 was computed from a dataset of 50 images of approximately 50 images.

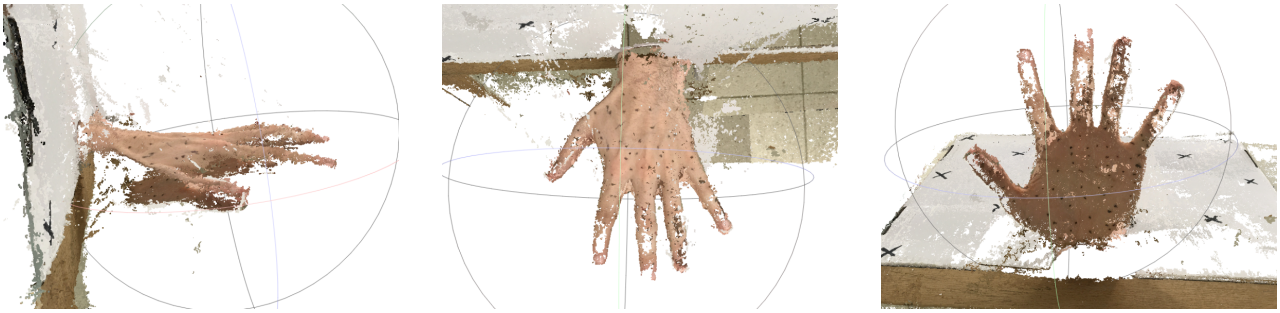


Figure 18: The dense point cloud using a dataset of 50 images

This time the reconstruction resulted in a much denser point cloud which was very promising.

I attempted yet another dataset, this time, in order to utilise the largest amount of views and recreate the hands with no landmarks on the skin, I increased the size of the dataset from 50 to approximately 200. This should allow for more vertices to be triangulated and also account for the loss of views. The results are shown in Figure 19, with the background having been manually removed for clarity.

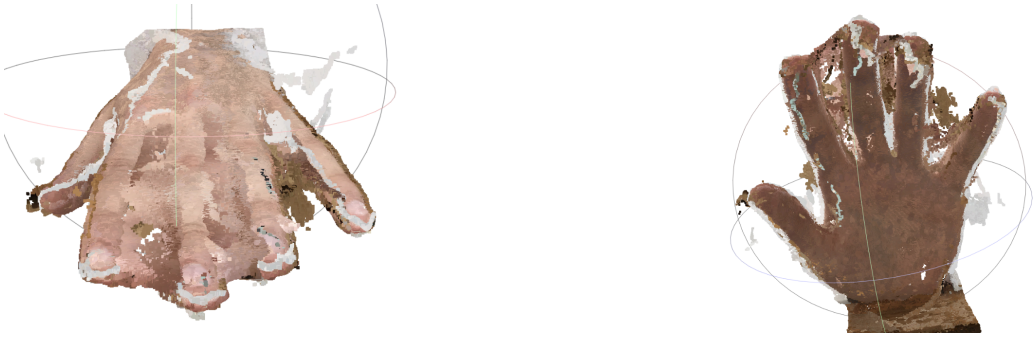


Figure 19: The dense point cloud using a dataset of 200 images

As can be seen, the point cloud is significantly denser. However, the movement of the hand has been the cause of a lot of noise surrounding the point cloud. When reconstructing the mesh of the hand, I expect this noise to be minimised since the cloud is much denser where the hand was stationary for longest.

Considered in its entirety, this was a successful redesign, as it, paradoxically, short-cutted several future challenges, such as aligning the two separate halves of the hand, which would most likely have produced problems that would have taken time to overcome.

3.8 Mesh Reconstruction

Now that the system is able to reconstruct a dense point cloud of the full hand, the reconstruction of the mesh from this point cloud, must be added to the pipeline. For this task, OpenMVS provides a binary executable *ReconstructMesh*. The executable aims to estimate the surface mesh that best fits the point cloud given to it, with various parameters available to tweak in order to get the best results. The

algorithm used is based on a paper by Jancosek and Pajdla, entitled, 'Exploiting Visibility Information in Surface Reconstruction to Preserve Weakly Supported Surfaces' [14].

The algorithm avoids reconstructing the surface mesh using the vertices in the point cloud, instead using them to model free space. Then, what is not free space is declared as full, and the boundary between them becomes the surface. As the title would suggest, the algorithm developed excels at reconstructing weakly supported surfaces and is more accurate than other available methods.

Due to the first point cloud in Figure 17 being too incomplete, I will not be attempting to reconstruct the 3D mesh for it. For the second full-hand result in Figure 18 however, I reconstructed the mesh with default parameters. Here are the results:

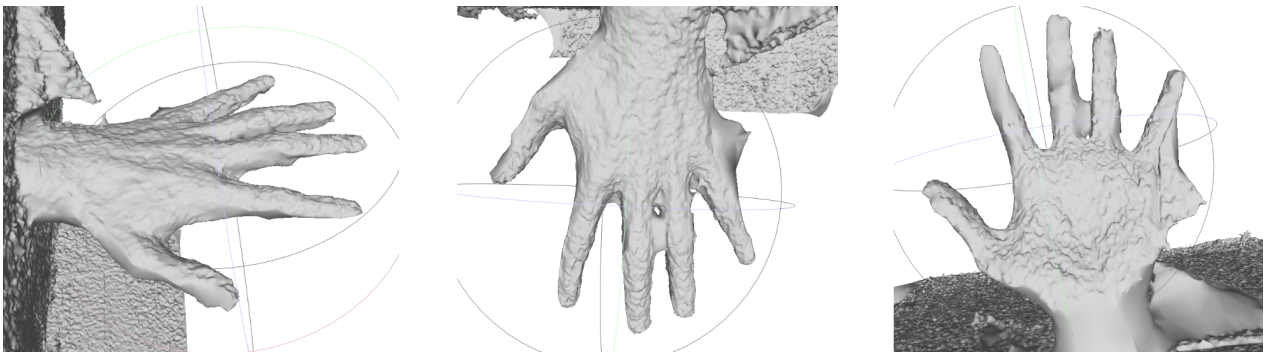


Figure 20: The reconstructed mesh for the point cloud in Figure 18

I consider this a fairly successful start. All the digits are distinguishable and the scale and proportions of the hand did not seem to deviated drastically from those of the point cloud. There are some obvious issues, however; to start, from the vertex shading I can see that the surface texture of the hand appears rough and bumpy. This is a problem since the actual texture of the skin should be locally smooth (ignoring imperfections or wrinkles) in order to appear realistic. In light of this, the parameters of the ReconstructMesh binary must be altered in order to remove the local sharp small bumps in the mesh, but keep the global gradients so that attributes of the hand like protruding knuckles or bone structure in the back of the hand remain clear. Here is a simple diagram to explain:

Figure 21: Local gradient changes are smoothed while the global structure remains

The executable also has a smoothness parameter, this controls the number of iterations performed in order to smooth the reconstructed mesh. Each iteration does not reduce the resolution of the final mesh, it just flattens bumps in the mesh. This parameter is set to 2 by default.

I attempted to change the parameter to 4, but this failed to produce a drastically different effect. After changing it to 20, however, the result was vastly improved (see Figure 22).

There is a clear improvement here as it was successful in smoothing out the hand whilst still succeeding in retaining some of the underlying bone structure.

One rather eye-catching issue is the large protrusion on the side of the little finger as well as the indentation on the side of the thumb, which are especially clear on the smoothed hand, having been somewhat amplified. I believe these are simply caused by the point cloud not being dense enough in

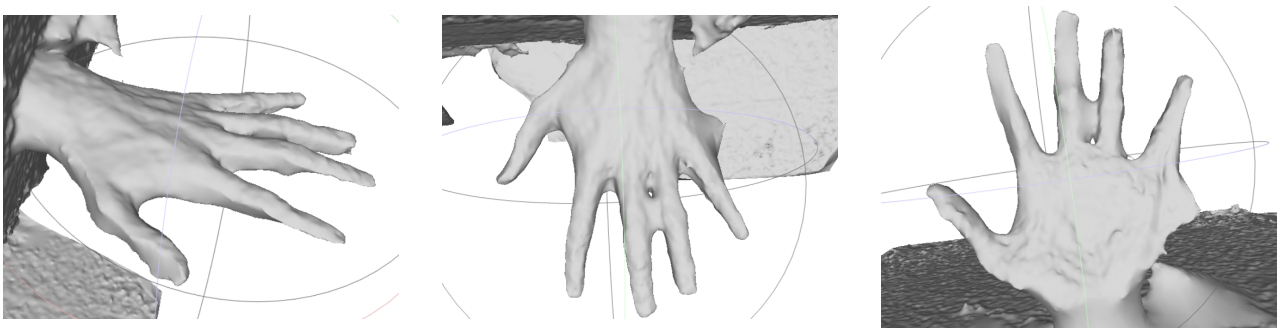


Figure 22: The reconstructed mesh after increasing the number of smoothing iterations to 20

these areas. Re-examining Figure 18, this hypothesis is strongly corroborated. Fortunately, subsequent datasets were able to obtain the location of vertices where the current dataset was unable.

Now, when attempting to reconstruct the mesh of the second much denser point cloud, the results can be seen in Figure 23.

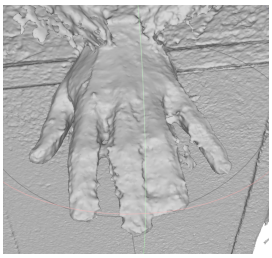
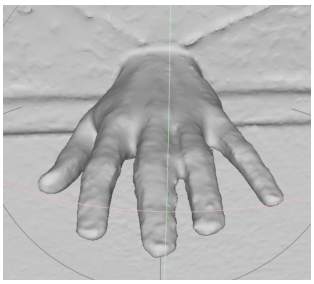
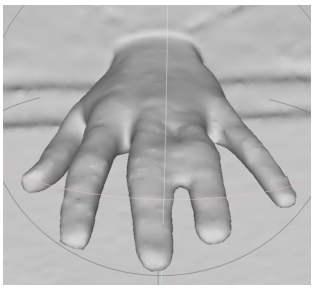


Figure 23: The reconstructed mesh for the point cloud in Figure 19

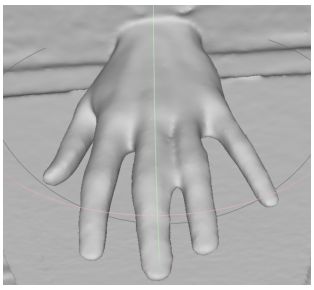
It's clear that due to the number of vertices in the point cloud, the outputted mesh has a lot of noise. To tackle this, I altered the parameter d. This parameter controls the distance between two vertices needed in order for them to be considered as distinct, by increasing this I should be able to reduce the resolution of the hand and obtain a clearer mesh. The following figures show how altering d and smoothness can obtain an optimal mesh from the point cloud.



(a) d=8



(b) d=16



(c) d=8, smoothing iterations=10

Figure 24: The refinement of the mesh using available parameters

With the final parameters, I consider this reconstructed mesh a good result that completes the fundamental objective of accurately reconstructing the mesh of the hand. I would now like to attempt to apply a texture to the mesh, as a secondary objective of this project which was outlined at the outset.

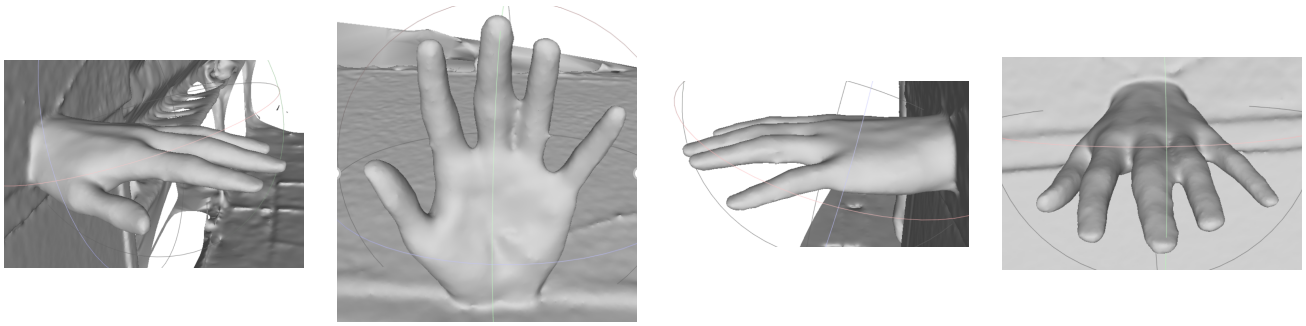


Figure 25: The final result of the mesh reconstruction

3.9 Mesh Texturing

The texturing of the 3D mesh, for simplicity reasons, utilises OpenMVS's *TextureMesh* binary executable. Using this means that a reimplementation of the algorithm is unnecessary, especially since it's designed to be used with the output of the previous *ReconstructMesh* executable. The algorithm used is relatively straightforward and similar to the method discussed in the design section of this report, based on the work of Vu et Al. [24]. The more complex area, making it more valuable, is the way in which it is able to cope with the errors accumulated from calculating the pose of each view.

Figure 26 shows the results of texturing this mesh, after scaling the image resolution down by 2 to ease computation:

(a) Without vertex shading active

(b) With vertex shading active

Figure 26: The result of texturing the reconstructed mesh

3.10 Overcoming the Problems

Unfortunately, the inability to successfully triangulate the detected landmarks of the hand, set the remaining project back significantly. Despite my discovery of a way in which the system could be redesigned, the triangulation of the landmarks is vital to isolating the hand model from its background, rigging and skinning the model. After no improvement in the results of triangulation, I decided that it was necessary either to locate the keypoints in a 3D space without the use of triangulation, using another method, or to not locate the hand keypoints at all. By not locating the keypoints on the hand, I will be sacrificing the ability to both rig and skin the model, a major disadvantage. However, there are still methods for isolating the hand model that do not rely on the keypoints being located, so those could certainly be attempted in a future project with a larger scope.

I have thought of numerous designs which would allow for the location of the hand keypoints in a 3D space using the information I already have, and whilst I selected one to implement, it is one which will have to be implemented in a future project. Part of this is due to the ambition of the project, given the time available, and the unpredictable practice of any attempt to estimate how long certain parts of any given project in this area might take. Many of the libraries used in this project are small and maintained

by individuals or small groups, therefore I have run into numerous errors in their source code which require working around, and have had to swap between libraries in some cases, also costing significant time. In short, the popularity of other similar, but none the less differing research areas, such as that of face reconstruction for instance, have meant that hand reconstruction is an area for significant growth, in which resources and research are perhaps less readily available. However, I will still discuss the various methods for locating the hands in a 3D space as well as the methods for isolating the model without knowing the location for the hand keypoints.

3.10.1 Locating Hand Keypoints in a 3D Space

There are two potential designs for re-implementing this step of the system pipeline, both of them involve further use of the data gained from using OpenMVS to perform Multi-View Stereo techniques on the hand.

Both methods should be able to obtain a location for any hand keypoint, from just one view. This differs from the triangulation methods, which required the selection of multiple views, and was an imposing challenge in itself.

Since we require just one view, then both methods should start like so:

First, for every view successfully used in the Multi-View-Stereo reconstruction, the image belonging to the view should be used to detect the hand's 2D landmarks. Then, for each keypoint the system should track which view has detected that keypoint with the highest confidence score. That view will then be the one used to locate that keypoint. To take an earlier example, in either of the views displayed in Figure 15, the fingertips of the hand look to be positioned correctly and could therefore be used to locate their position in 3D space. However, the views should perhaps not be used to track the thumb or the palm of the hand, since these cannot be clearly seen in the image.

Now the 2 techniques, taking the keypoint and corresponding view as input, are separated.

3.10.2 Utilising Multi-View-Stereo Depth Maps

The first technique will extract the depth map for the input view, that was used for multi-view-stereo. Then, the depth of the keypoint is extracted from the depth map using the 2D coordinate of the keypoint. However, this depth, while related to the scene in the image, may not translate to the world coordinates that the reconstructed hand is in, especially if the depths for every point change when the depth maps are fused together so that the relative depth becomes global depth. I believe it may still be possible to translate the depth at that 2D coordinate, to the 3D coordinate in world space if I make use of the OpenMVS interface.

Figure 27: A view of the hand with keypoints detected, and its corresponding depth map

3.10.3 Backprojection through the 3D Mesh

In the second technique, the pose of the input view is first extracted in world coordinates. Then, a ray is cast from the camera centre which passes through the projected keypoint, much like in the triangulation

of the keypoint.

I then see where the ray intersects with the 3D reconstructed mesh of the hand the first and second time it does so. Figure 28 depicts the ray entering and leaving the mesh where the keypoint is to be situated.

Assuming the skeleton of the hand should lie as centrally as possible inside the hand, I can then assume that the mid-point between the entrance and exit of the mesh is where the keypoint will be located in a 3D space.

The disadvantage of this is that problems could occur with views depicting the side-view of then hand, as the ray will no longer be entering and exiting the hand through the located keypoint (for example when locating the palm or knuckles). However, there are ways of preventing this. One way is to simply avoid using views that are from the side; done by either manually selecting usable views, or alternatively, to implement functionality that will discard a view's ray if the second intersection with the mesh (upon exit) is a certain distance away from the first (upon entrance).

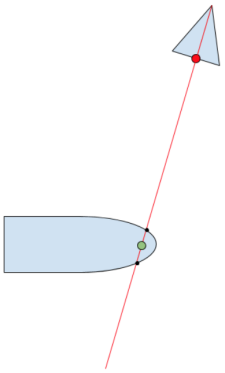


Figure 28: The location of the midpoint illustrated

3.10.4 Isolating the Model

Due to the difficulties discussed in triangulating the hand landmarks in the 3D space, isolating the hand model from its background would remain an important task, using a different design which would not depend on the localisation of the triangulated landmarks.

One possible method would be to utilise detectable markers (such as marker codes or bar codes), in order to obtain the plane that models the background behind the primary subject of the hand. Using this plane as a guillotine, it would be possible to slice the hand, before removing all faces and vertices not associated with the now free-floating, isolated hand.

Another method is to use the 2D detected keypoints to extract the average colour of the hand. Any vertices that do not lie within a certain colour range, built from this average colour, are removed. As discussed in the design section of this report, this may also remove shadowed areas as well as the fingernails, an obvious disadvantage in the ultimate goal of realistic and recognisable hand reconstruction.

4 Evaluation Experiment and Results

4.1 The Experiment

In this section of the report, I will describe the evaluation process chosen and why I regard it as the correct approach to have taken for my particular project. As discussed earlier, the main area of evaluation shall focus on how accurate the proportions and overall geometry of the reconstructions are, given that my

main goal was to create realistic hand reconstructions. Therefore, I will be omitting the mapped texture and focusing solely on the reconstructed mesh.

The experiment involves using a pre-existent 3D model of a hand, attempting to reconstruct it, then evaluating the reconstruction. The steps for the evaluation experiment will be as follows:

A 3D model of a hand online should be downloaded. This model will act as the ground truth in the comparison.

Now, I must use the developed system to recreate this model, using a similar environment to that which I would use when recreating a real-life hand, as opposed to a ground-truth model. In order to create this environment, I I have taken the backdrop from one of the other reconstructions and removed the reconstructed hand from the scene. I have then translated, rotated and scaled the ground truth model so that it fits in front of the now hand-less backdrop, in a similar pose to that which would be adopted by the subject during the capturing of a real-life dataset. For the purpose of the evaluation, it is not problematic if there is some overlap or slight mismatch in scale, as long as the backdrop is visible from most angles. The resulting scene is shown in Figure 29.

Figure 29: The scene containing the ground truth model to be reconstructed

I will now attempt to emulate the capturing process of a real-life dataset by taking screen-shots of the current scene, before rotating the camera around a point on the hand slightly and taking another screenshot. I will repeat this, attempting to capture as many angles of the hand as I can, until I have a dataset of N different views.

Having done this, I will extract the intrinsics of the virtual camera simulated inside of MeshLab, and input this with the dataset into the project pipeline in order to reconstruct a dense point cloud from the dataset.

Multiple meshes will be computed from the obtained dense point cloud, each while altering the parameter that sets the distance between 2 vertices necessary for the to be classed as distinct from one another: d.

This process will be repeated while changing the value of N, this should provide information about what value of N will give the best reconstruction possible, as well as which mesh reconstruction parameters provide the best reconstruction for that value of N.

The planned values of N will be 25, 50, 75 and 100. For each value of N, the mesh reconstructed parameters will be d = 2.5 (default), 5, 7.5 and 10. I am purposely not changing the number of smoothing iterations when reconstructing the mesh, as I don't believe it will have a drastic effect on the reconstructed meshes accuracy.

Once a reconstructed mesh has been computed and outputted, a method is required to be able to analyse the differences and similarities of the reconstructed mesh next to its ground truth and compare the measurements with the other outputs.

Before a comparison can be made, the reconstructed mesh must be aligned with the ground truth mesh in world coordinates. To do this, I will first import the two meshes into Blender, a 3D modelling

N=25		
d	Hausdorff distance	Mean error
2.5	0.073117	0.005281
5.0	0.068701	0.006636
7.5	0.070619	0.006999
10.0	0.065297	0.008732

N=50		
d	Hausdorff distance	Mean error
2.5	0.038087	0.005051
5.0	0.0.038866	0.004846
7.5	0.041939	0.004871
10.0	0.046009	0.005299

software, before using an alignment method called manual point alignment. This method requires me to select 4 or more matching points on both meshes, which will then be used to transform one mesh, so that the sum of the distance between all the points is as small as possible. Once aligned this way, there may still be some prevailing misalignment, since there exists only a finite number of places on the hand where I can accurately match points, which, if not spread out far enough will produce some error. I will now use the Iterative Closest Point algorithm to refine the existing alignment. Then I will export the aligned meshes and import them in to MeshLab for analysis.

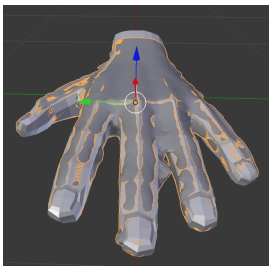


Figure 30: The reconstructed mesh aligned with the ground true mesh in Blender

In MeshLab, the reconstructed mesh will be analysed against the target ground truth. The method of doing so is as follows. First, I must calculate for every vertex in the reconstructed mesh, the minimum geometric distances between it and any vertex belonging to the ground truth mesh. These distances can be mapped directly onto the reconstructed hand, visualising the error in the form of a heat map. I will also record from these distances, both the mean error and the maximum distance, with this latter being known as the Hausdorff distance.

I predict that the mean error will be the most valuable of these results when comparing the accuracy of the meshes, as it will not be affected by large unexpected protrusions as the Hausdorff distance may be.

By evaluating various reconstructions of the same model like this, I am able to get an excellent overview of the range of accuracy that can be achieved using this system pipeline. It will reveal where the system pipeline can be improved, should there be a larger scope or future project of similar nature.

4.2 Results

The following data: table 1 and Figure 31, show the results, as well as the error map for each optimal reconstruction (minimum mean error) with each value of N.

From the error maps shown in Figure 31, we are able to see the areas of the reconstructed mesh with the largest displacement from the ground truth mesh. Every reconstruction struggled to separate the fingers in places, especially the middle and ring finger which are positioned particularly close together in the pose of the ground truth model. However, from the progress made between the datasets, I suspect that with more views this error would be eliminated. Looking at the error map for the reconstructed mesh where N=25, it is clear that the generated point cloud was too sparse due to the small number of views, and a digit of the hand is severed as a result.

N=75		
d	Hausdorff distance	Mean error
2.5	0.042943	0.4322
5.0	0.041719	0.004040
7.5	0.045493	0.004464
10.0	0.052701	0.005176

N=100		
d	Hausdorff distance	Mean error
2.5	0.036062	0.003697
5.0	0.036008	0.003667
7.5	0.035812	0.003881
10.0	0.065297	0.008732

Table 1: Tables showing the results collected

The following figures, 32 and 33, plot a point for each mesh that was reconstructed from a dataset, using a separate line for each dataset of varying size.

The graph in Figure 33 measures the mean error for each mesh constructed. We can see a separation between the lines of different N values, with N=25 (the lowest value) claiming the largest mean error, and N=100 (the highest value) having the least mean error. Therefore, we can say that increasing N will lower the overall error present in the reconstruction, at least up to a value of N which is 100. I believe that, in the ideal conditions of the virtual simulation, the error could be reduced for much greater values of N than 100. However, in practice the human hand cannot possibly stay as steady as the ground truth model, which was, of course, entirely still, and I would predict that, in real-life conditions, once N exceeds a limit (which would depends on the steadiness of the hand), the point cloud calculated will become too noisy and the reconstruction will start to deteriorate as the hand begins to move, even subtly.

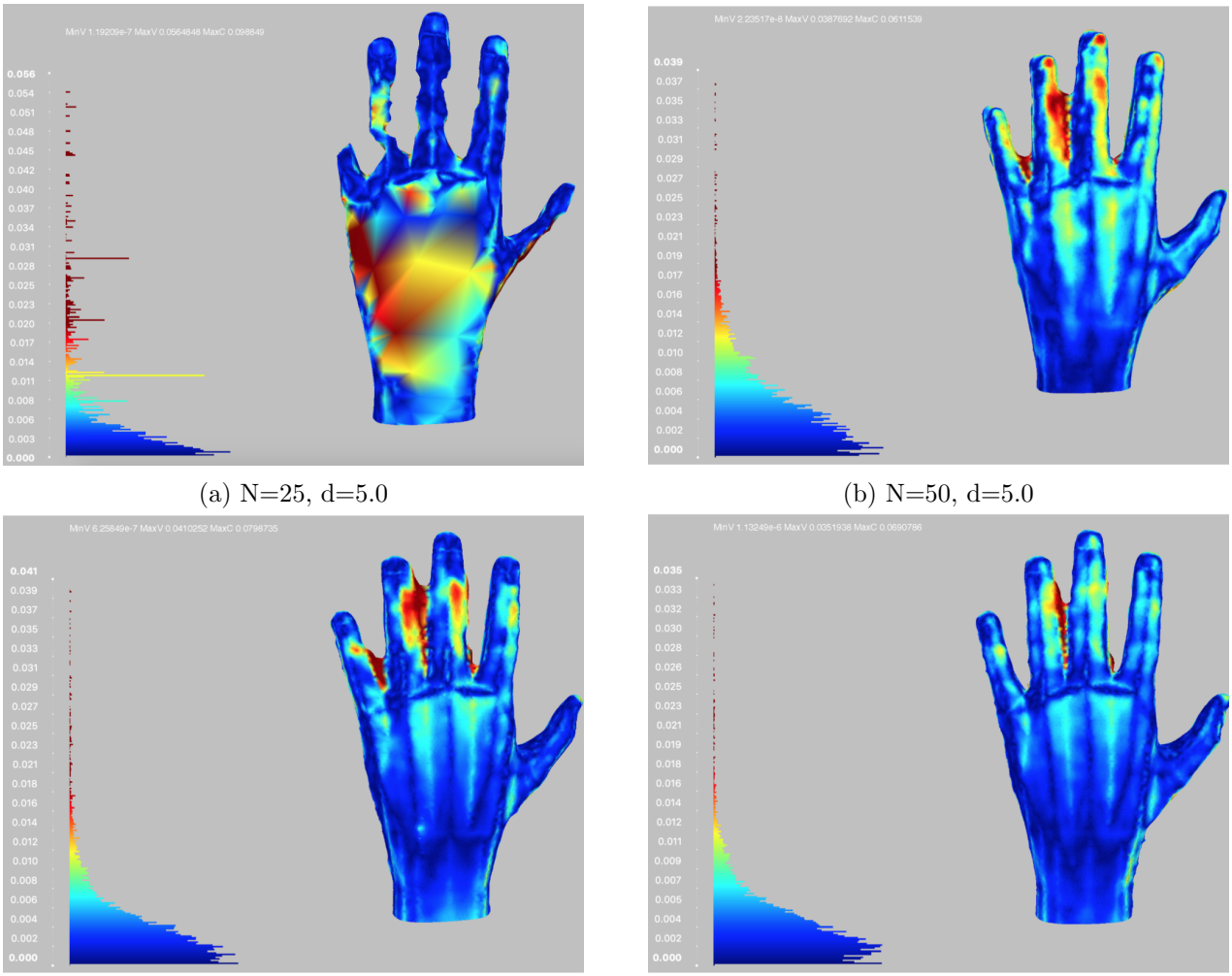
As parameter d is increased from the default 2.5 to 10, the behaviour of the reconstructions depends on how many views are in its dataset (N). For example, for datasets with ¿25 views, the mean error does not change too drastically. Referring back to what changing 'd' achieves, it controls the distance between two vertices to actually consider them as distinct from one another. Logically, if we increase 'd', we are decreasing the number of points in the generated point cloud, and, perhaps, the resolution. So, for the datasets where N¿25, this might result in a smoothing of the bumpiness of outputted mesh (indeed, this smoothing effect of 'd' can be seen during the development stage), which in turn could refine the reconstruction, potentially decreasing the mean error, as can be seen in Figure 33, when 'd' has a value of 5. However, as we can see from the graph if the value of 'd' becomes too large then it will start to have a negative impact, and may miss out crucial parts of the hand's shape by discarding vertices. This is also visible in the graph, as 'd' increases from 5 to 10. The dataset where N=25 behaves differently as 'd' increases. Due to the lack of views, the point cloud generated was only just dense enough to accurately depict the shape of the hand. By increasing 'd', we further remove vertices from the point cloud, which has a drastically negative impact on the overall product.

The graph in Figure 33 measures the Hausdorff distance, a term described earlier in this paper. Again, there is a large gap between the datasets where N has a value of 25 or greater. This is most likely because the reconstructed mesh might have large protrusions or indentations in areas where the point cloud is not dense enough, these will result in the mesh having a high Hausdorff distance.

5 Future Scope and Conclusions

As I had originally anticipated, this project was a constant journey of 1 step back, followed by 2 steps forward. A big factor in this is the reliance the project has on smaller scale libraries, which are worked on by individuals or small groups. Any bugs within these libraries' source code would often require a redesign of my own system, as even monthly bug fixes on the part of the library maintainers, would be rather optimistic.

Having navigated through significant challenges, the end result is a system pipeline that can be of great use not only in its intended application area, but in another specific area that was brought to light during the evaluation process. This process highlighted the possibility of using the techniques outlined in



(a) N=25, d=5.0

(b) N=50, d=5.0

(c) N=75, d=5.0

(d) N=100, d=5.0

Figure 31: The best hand reconstruction for each value of N, with the error mapped to the mesh

this report, in order to analyse the difference between two separate reconstructed hands.

5.1 Future Scope

It's exciting to think about what a similar project can accomplish with a larger scope and I hope to be able to continue to research this area and refine the techniques outlined in this project accordingly.

From the evaluation results, it is clear that the most difficult part of the hand to reconstruct are the fingers. In the evaluation, there was no movement in the subject hand, so any error was due to their intricacy. However, in real-life reconstructions, there is the added difficulty of movement, as is clearly shown in the development process. I believe the problem posed by the intricacy of the fingers can be solved by using a larger dataset, and this is shown to work to some extent in the results of the evaluation, as N=100 models the mesh of the fingers with the greatest accuracy. However, by increasing the size of the dataset when reconstructing a real hand, the increased time that this takes to obtain means that the fingers have significantly more time to move, thereby counteracting any advantage gained through the use of a larger dataset, since many of the views will likely be discarded.

There exist recent techniques that expand further on the Multi-View Stereo algorithm instead of simply redesigning or improving it. In the very recent paper 'NRMVS: Non-Rigid Multi-View Stereo', by Innmann et al. [13], the focus is placed on the 3D reconstruction of objects that are more dynamic than

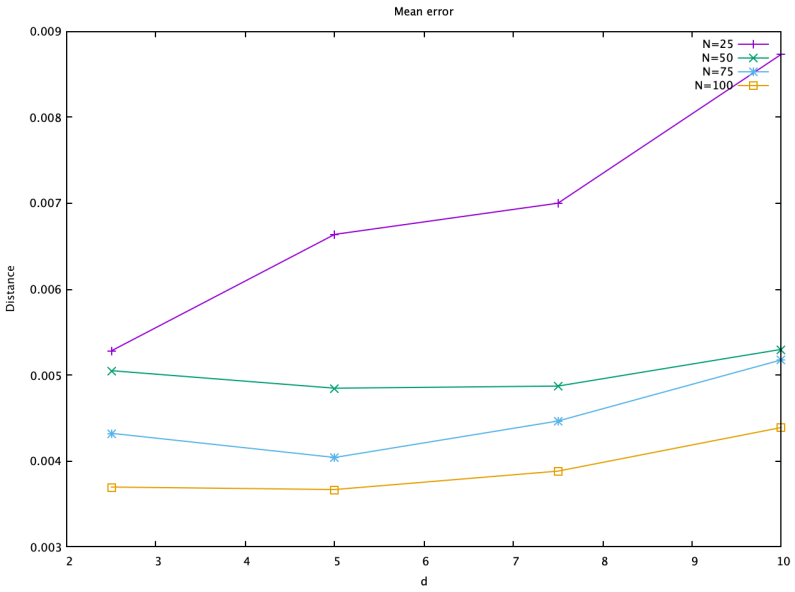


Figure 32: A graph of the mean errors

structures and models. One example cited in the paper, is that of a human face where the expressions change slightly in-between each image. This technique would be an excellent solution to the problem just mentioned and would allow for more images in the dataset without hindering the result.

In the work of Innmann et al., an initial point cloud is constructed using only views that have minimal deformation, these views are dubbed canonical views. For the remaining views, the depth and deformation values are estimated and this information is used to interpolate the deformation between views.

5.2 Conclusions

Comparing the end result to the original project scope, I have successfully completed all of the existing fundamental objectives (which were defined as crucial to the successful outcome of my project) to a standard that solves the original motivated problem. These fundamental objectives were:

- Firstly, designing and refining the optimal conditions for capturing the input dataset, with a significant and visible improvement in the change in output.
- Secondly, computing matching features between the views.
- Thirdly, executing Structure from Motion techniques to calculate and refine the camera poses accurately.
- Fourthly, generating a dense point cloud using Multi-View Stereo techniques, and learning how to improve it.
- Finally, reconstructing the mesh from the dense point cloud, and hence providing the user with an output that is ready to be used in the application area required.

Furthermore, I was able to complete the secondary objective of texturing the outputted mesh. This adds a lot of value to the project system as a tool for its application area. The rest of the secondary objectives, while having not been implemented in this project, were researched thoroughly and have been utilised heavily in a consideration of redesigns which could be implemented in a future project of greater scope.

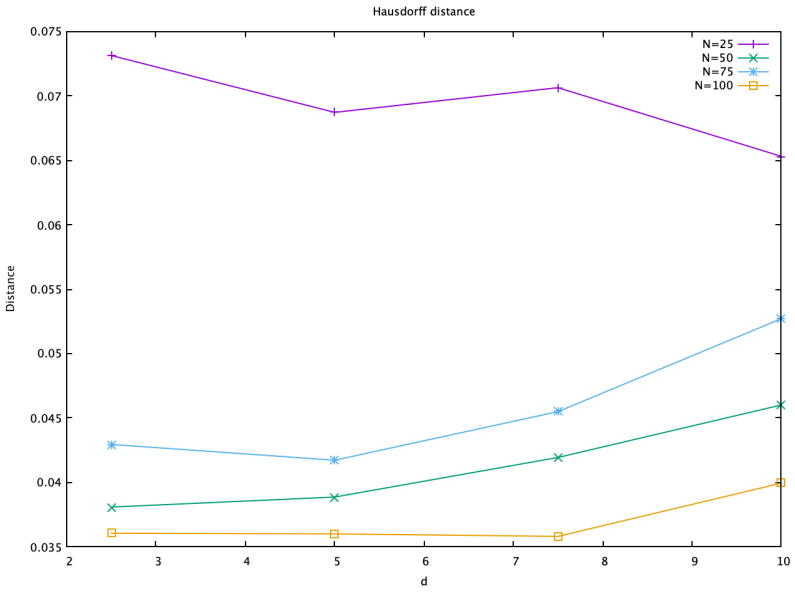


Figure 33: A graph of the Hausdorff distances

Given the ambitious nature of this project and its relatively small scope when compared with the few other projects which have been undertaken in this area, I am satisfied that I have been able to do justice to the methodology, succeeding in reconstructing a geometrically accurate, realistic and well-proportioned hand model. The partial failure of secondary objectives, with particular reference to the rigging and skinning processes, is one which I had set out to overcome. However, I am confident that I would be able to solve the issues of triangulation which this faced, in a future project of larger scope. I greatly look forward to continuing research in this area and adding to the relatively sparse number of programmers working on the reconstruction of hands.

When compared with other areas of 3D reconstruction, such as the face, buildings and landscapes, hand reconstruction is an area which has received little attention. I believe that, in choosing to undertake a project in this relatively neglected area, I have been able to tackle problems that would never have arisen in areas where there exists more research. Overall, this has meant that I have, I believe, been able to truly challenge myself during the implementation of this project.

Evidently, the area of hand reconstruction is one which is still in its infancy and will no doubt, with the involvement of more researchers, and the development of suitable methodologies, experience significant growth in future years. This project has been one which has greatly broadened my horizons of interest, and I have relished the opportunity to be able to undertake work which is not far behind the current frontiers of computer science. I am eager to see how far these frontiers will extend in the near future.

References

[1] Epipolar geometry and the fundamental matrix.

[2] Feature Matching — OpenCV 3.0.0-dev documentation.

[3] Four-Fingered Hands.

[4] Multiple View Geometry | Real Time 3d Reconstruction from Monocular Video.

[5] The key steps to modeling a hand in the fastest amount of time, Nov 2014.

[6] Federica Arrigoni, Beatrice Rossi, and Andrea Fusiello. On computing the translations norm in the epipolar graph. 10 2015.

[7] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1611.08050, 2016.

[8] cDc. open Multi-View Stereo reconstruction library. Contribute to cdcseacave/openMVS development by creating an account on GitHub, April 2019. original-date: 2015-05-18T15:21:44Z.

[9] Justin Slick A. technical writer who specializes in 3D character and Environment Creation. How Are 3d Models Prepared for Animation?

[10] Mike Cohn. Scrum Methodology and Project Management.

[11] Robert Collins and Penn State. Lecture 19: Essential and Fundamental Matrices. page 29.

[12] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

[13] Matthias Innmann, Kihwan Kim, Jinwei Gu, Matthias Nießner, Charles T. Loop, Marc Stamminger, and Jan Kautz. NRMVS: non-rigid multi-view stereo. *CoRR*, abs/1901.03910, 2019.

[14] Michal Jancosek and Tomas Pajdla. Exploiting Visibility Information in Surface Reconstruction to Preserve Weakly Supported Surfaces, 2014.

[15] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[16] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[17] Pierre Moulon, Pascal Monasse, and Renaud Marlet. Global Fusion of Relative Motions for Robust, Accurate and Scalable Structure from Motion. In *2013 IEEE International Conference on Computer Vision*, pages 3248–3255, Sydney, Australia, December 2013. IEEE.

[18] Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, and Christian Theobalt. Ganerated hands for real-time 3d hand tracking from monocular RGB. *CoRR*, abs/1712.01057, 2017.

[19] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[20] Daniel Seita. Learning a Multi-View Stereo Machine.

[21] S. Shen. Accurate Multiple View 3d Reconstruction Using Patch-Based Stereo for Large-Scale Scenes. *IEEE Transactions on Image Processing*, 22(5):1901–1914, May 2013.

[22] Johannes Traa. Least-squares intersection of lines. 2013.

[23] George Vogiatzis and Carlos Hernández. Video-based, real-time multi-view stereo. *Image and Vision Computing*, 29(7):434 – 441, 2011.

[24] H. Vu, P. Labatut, J. Pons, and R. Keriven. High Accuracy and Visibility-Consistent Dense Multiview Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):889–901, May 2012.