LeapMotion Visualisation System (LVS)

Leap Motion Controlled Data Manipulation
Using Visualisation Toolkit

Chun Yin, Tsang
Student ID: 1467193
Supervisor: Dr. Hamid Dehghani

Submitted in conformity with the requirements
for the degree of BSc. Computer Science
School of Computer Science
University of Birmingham

Abstract

LeapMotion Visualisation System (LVS)
Leap Motion Controlled Data Manipulation
Using Visualisation Toolkit

Chun Yin, Tsang

Visualisation was becoming more important in our daily lives. For example, medical applications, academic software and computer games were often required to display information in 3-dimensions. In the meantime, gesture interface was also becoming more popular as it had been applied to various commercial products and applications. However, most of the existing visualisation software had a complex UI and did not support gesture interaction. Therefore, we developed a new visualisation software that can overcome these deficiencies. The software included three main component, a Gesture Interface, VTK and the integrated GUI. A series of tests had been conducted and the result was recorded.

Keywords: Leap Motion, Visualisation Toolkit, Gesture Recognition, Java, Human-Computer Interaction

Acknowledgments

This project would not been gone this far without the help and support from many people. First, I would like to express my sincere gratitude to my supervisor, Dr Hamid Dehghani, for guiding and support me no matter on related knowledge or equipment throughout the project. Secondly, I would like to thank the participate who spent their time to complete the functional test and recognition test. These participates including the Ph.D. Students in the Medical Image Laboratory of University of Birmingham and also my friends. A special note of thanks to Dr. Alexandre Dufour, from the ICY developing team who helps me on solving the compatibility problem of VTK with Java and Mac.

All software for this project can be found at
https://codex.cs.bham.ac.uk/svn/projects/2015/cyt493/

Contents

List of Abbreviations **7**

1 Introduction **8**

1.1 Project Aims 8
1.2 Project Overview 8
1.3 Motivation 8
1.4 Report Outline 9

2 Literature Review **10**

2.1 Gesture Analysis 10
2.2 Motion Detectors 14
2.3 Visualisation Toolkit 16

3 Problem Analysis **17**

4 Solution Design **18**

4.1 Methodology 18
4.2 Project Plan 18
4.3 System Architecture 19
4.4 GUI Design 20

5 Implementation **21**

5.1 Overview 21
5.2 Gesture Analysis 21
5.3 Visualising with Visualisation Toolkit 25
5.4 Integrated System 28

6 Testing **36**

6.1 Unit Testing and Integration Testing 36
6.2 Functional Testing 37
6.3 Usability Testing 38
6.4 Recognition Rate Testing 39

7 Evaluation **40**

7.1 Related Work 40
7.2 System Performance 41
7.3 Other Aspect 42

8 Discussion & Future Work 46
8.1 Programming Language for Implementation 46
8.2 Gesture vs Mouse 46
8.3 Gesture Management 46

9 Conclusion 47

10 Bibliography 48

11 Appendix 51
11.1 System Requirement 51
11.2 Structure of Zip 51
11.3 Installation 51

List of Abbreviations

LVS	Leap Visualisation System
VTK	Visualisation Toolkit
HCI	Human-Computer Interaction
DOF	Degrees of Freedom
FEMD	Finger-Earth Mover's Distance
API	Application Program Interface
GUI	Graphical User Interface
UI	User Interface

1 Introduction

1.1 Project Aims

The aim of the project is to create a new visualisation system with a gesture interface. In contrast to similar software, the project aims to deliver a system with a much user-friendly interface such that it is easy to learn. Also, the system should be able to maintain its own file system to eliminate duplicated work. Lastly, the system should also maintain a smooth rendering process and allow user-customisation in several areas.

1.2 Project Overview

Visualisation has become more common in various academic researches areas other than medical imaging. After reviewing available visualisation software in the market, we have found that most of them have complex UI and are difficult to learn. Meanwhile, although traditional I/O device that used by these software like mouse and keyboard, provides a reliable and accurate interaction. In some particular cases like during a surgery or a presentation, neither mouse nor keyboard, is appropriate for communicating with 3D information. Considering manipulating 3D information is the emphasis of visualisation, a more appropriate solution should be introduced. Gesture was said to be the most natural way for human to communicate. This also applies to human-machines interaction. Yet, most of the existing visualisation software that allow gesture interaction are mainly designed for surgical uses only. To overcome these deficiencies, we have developed a new visualisation system that supports gesture interaction for general uses. The system consist of two major components: 1) gesture analysing and 2) 3D model visualising. Upon implementation of the system, we have also conducted several testings in order to evaluate the system.

1.3 Motivation

Computer scientists have been working on gesture analysis throughout the past three decades. In the past five years, the attention of the public on this aspect had a significant rise. This has been partly due to some technological breakthroughs such as 3D printing and Virtual Reality. Undoubtedly, we can foresee that visualisation and gesture interface will be widely used in the society for the next few decades. Therefore, learning or getting in-touch with related technologies will be the best starting-point for me to step into the area.

In the meantime, a new motion detector named Leap Motion has been released to the market recently. This device is specifically designed for PC and it provides a simple framework that facilitates gesture analysis. Until now, the device has been released for 3 years, however, most of its related software has been focusing on gaming. Therefore, it would be interesting and challenging to develop a software using this device for non-gaming purposes.

1.4 Report Outline

In section 2, we will present some background information that is related to the project. Following the background, section 3 will be a detailed analysis of the problems like projects requirements. In section 4, we will describe our solution design that can answer the questions raised in section 3. In section 5, we will demonstrate how the solution is implemented. In section 6, we will interpret our testing results. In section 7, we will evaluate the system based on various important aspects, for instance, the system performance. Finally, in section 8, we will discuss several further extensions to the project.

2 Literature Review

2.1 Gesture Analysis

Gesture is a non-verbal communication language, it can be represented by our body parts including hands, feet or even eyes. By using this language, human can interact with machines without any tradition I/O devices such as mouse and keyboard. In recent years, there has been a considerable effort made in the application of gesture interface in various devices and applications such as mobile phones, televisions and Virtual Reality Head Sets. These researches focus on different body gesture, one of the important research was done by Vladimir et al. [22]. They claimed that, it is natural and sufficient enough to use index finger to explore virtual objects. Therefore, among all types of gestures, the following paragraphs will mainly focus on hand gesture recognition.

According to Michal and Ying [16, 26], hand gesture analysis could be separated into two stages (Figure 1). The first stage is Hand Localisation, which aims to track the movement of user's hands and to locate the hands in the images. These images could be analysis by different methods, for instance, in form of histograms. After that, the data will be used for Gesture Recognition, and classified into various gesture types.

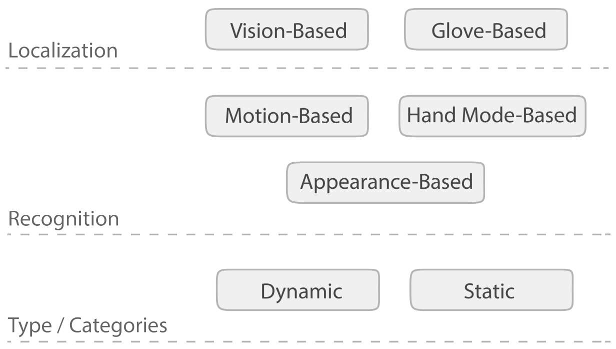


Figure 1: Levels of Gesture Analysis

2.1.1 Hand Localisation

The accuracy of gestures rely on trustworthy, real-time hand position and pose data. Accordingly, extracting correct hand features is the emphasis of this stage. In general, features like fingers and palms are usually extracted, but some devices like Leap Motion, more features like metacarpals (palm bones) and phalanges (finger bones) can also be extracted.

In order to extract those features from a complex background, throughout the past few decades, several methods had been developed and widely used in related researches. These methods can be grouped into two categories: Glove-Based method and Vision-Based method [3, 9, 10].

Glove-Based Approach Glove-Based approach usually requires users to wear a sensor device for digitising hand and finger motions into multi-parametric data. Sometimes, colour markers are used instead in order to provide colour cues. Shahram J et al. [20] demonstrated a wrist-worn IMU (Inertial Measurement Unit), which features on transmitting three-axis accelerometer and other geometrical data wirelessly as to interact with medical images for surgical uses. With the help of modern technologies, users no longer require to wear a cumbersome device which carries a load of cables. However, as Vladimir et al. [22] mentioned , Glove-Based approaches still do not completely fulfill one important requirement of HCI: Naturalness.

Vision-Based Approach The solution to overcome the limitation imposed by Glove-Based approach bring us to the Vision-Based approach proposed by Turk and Kolsch [15]. In contrast to Glove-Based, Vision-Based approach uses computer vision in order to sense and perceive users and their actions within a HCI context. Recently, depth cameras are commonly used in this approach to provide depth cues rather than color cues from data-gloves [28]. However, this approach also raises several challenging problems which researchers have been solving over the last decade. For example, background invariant, light insensitive and person independent, these problems must be solved to achieve sufficient real-time performance. In [19] the author mentioned several common assumptions could be made including: assuming high contrast stationary backgrounds and ambient lighting conditions. Later on, we will discuss about the devices in current market that could resolve these problems.

In summary, under the assumptions mentioned above, Vision-Based approach is a better way to implement the gesture interface for any software since it provides a more natural human computer interactions.

2.1.2 Gesture Recognition

The emphasis in gesture recognition is how to make the hand gestures understood by the system. However, there are two associated questions proposed by Vladimir et al. that we have to aware: 1) How to optimise partitioning time-model parameter space? 2) How to implement the recognising procedure? To overcome this, they have introduced two types of Spatial Gesture Model(Figure 2): 3D Hand Model-Based and Appearance Based.

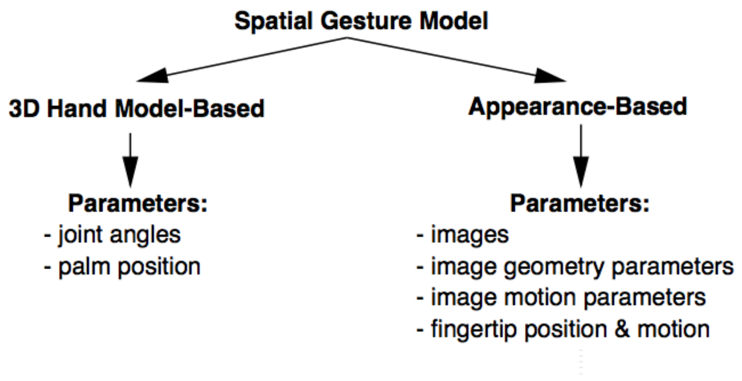


Figure 2: Spatial Gesture Models [22]

3D Hand Model-Based Approach 3D Hand Model-Based approach focus on recovering the three dimensional configuration of articulated body parts from 2D images. Considering the human hand has a complex anatomical structure consisting of 27 bones, they can be divided in three groups: carpals (wrist bones - 8), metacarpals (palm bones - 5) and phalanges (finger bones - 14). These three groups of bones are connected by joints naturally, however, some of them have very limited freedom of movement, for instance, carpal-metacarpal joints. James and Takeo [5] have demonstrated a system called DigitEyes, and proposed that by using 27 DOFs, the bones and joints could be translated to angles or even vectors. This method could eliminate many unnecessary calculations by reducing the size of the parameter space.

Appearance-Based Approach Appearance-Based approach uses two dimensional information such as gray scale image, edges or body silhouettes. For example, image features from training set are used to model the visual appearance of the hand, these parameters will then be used to compare with the extracted image features from the video input. The approach can provides better real-time performance, since easier 2D image features are employed. Zhou [28] demonstrated with the aid of FEMD, the recognition was performed by matching only fingers but not the whole hand shape. However, the detection is highly sensitive to lighting conditions, without strictly controlled working environment the robustness of the system will decrease rapidly.

In conclusion, Appearance-Based has faster real-time performance, but 3D Hand Model-Based provides a rich description that widen the class of hand gestures and its robustness. The comparison of each levels of approaches are listed below (Figure 3.1 & 3.2).

	Vision-Based	Glove-Based
Robustness	✓	✓✓
Naturalness	✓✓	✓
Easy To implement	✓✓	✓

✓ = Fair ✓✓ = Good

Figure 3.1: Localisation Level Approaches

	Hand Model-Based	Appearance-Based
Robustness	✓✓	✓
Gesture Variety	✓✓	✓
Real-Time Performance	✓	✓✓
Easy To implement	✓	✓✓

✓ = Fair ✓✓ = Good

Figure 3.2: Recognition Level Approaches

2.1.3 Types of Gestures

In HCI perspective, gestures are classified into two classes: Static (hand pose) and Dynamic (hand movement) [2]. The main difference between static and dynamic are the number of frames the system requires to identify the gestures. Generally speaking, a Static Gesture is single frame oriented where it could be recognised from particular hand configuration or pose. Whilst Dynamic Gesture involve more than one frames with continuous motions over a short time span. In all scenarios, a Dynamic Gesture will have a start state and end state, but not a specific start and end position.

2.2 Motion Detectors

Over the past 3 decades researchers have been working on developing different methodologies and devices to facilitate hand and gesture recognition [17]. Referring to previous section, naturalness is an important and recent interested aspect in HCI, therefore, the followings will only focus on depth cameras but not data-gloves.

Different from data-gloves, depth camera provides a depth map contains information relating to the distance of the surfaces of scene objects from a steady viewpoint. Although it does not provide direct hand data or color cues, it can still supply sufficient information and fulfilling naturalness for gesture recognition. The followings discuss about two of the most popular sensor that commonly used by researchers.

2.2.1 Microsoft Kinect

Microsoft Kinect is one the most popular sensor in the market. Originally, the sensor was developed for game console XBox 360. After several enhancements, due to its variety in supported platform and high performance, it has been frequently used in computer vision related research.

The sensor consists of infrared projector, infrared camera and RGB camera, nonetheless, it also accepts voice input. Although the sensor does not study particular features itself, this behaviour provides much more variation which facilitates research in many aspects, such as indoor robotics and 3D scene reconstruction. The software development toolkit allows developers to write Kinect applications in C++, C#, Visual Basic and .NET. Zhou R (2011) in [29] demonstrated a hand gesture recognition system that operates robustly in uncontrolled environments with Kinect sensor.

2.2.2 Leap Motion

In contrast to Kinect, Leap Motion explicitly targeted to hand gesture recognition. The device consists of two monochromatic Infrared cameras and three infrared LEDs, covering a roughly hemi-spherical area within 1 meter and with precision up to 0.01 millimeter. Hand position and pose data are described by a list of vectors. Different from Kinect, these information is extracted from a depth map (figure 4) and provided by the API that comes with Leap Motion controller. The device employs a right-handed Cartesian coordinate system (figure 5). The origin is centered at the top of the Leap Motion controller. After configuration, an interaction box with the same size as the primary screen can be set up. This methodology facilitates the capture of movement and hand data which can be used for further gesture recognition process.

Figure 4: A picture capture by Leap Motion

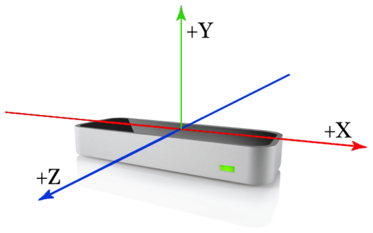


Figure 5: Leap Motion coordinate system

Tracking Model The API defines a Tracking Model - ”Frame” (figure 6) which contains lists of primary tracked objects, for instance, hands, fingers, tools and gestures. All the pose data is represented in forms of Vector. The tracking model can support multiple hands tracking, but as recommended keeping two hands only can optimal the motion tracking quality.

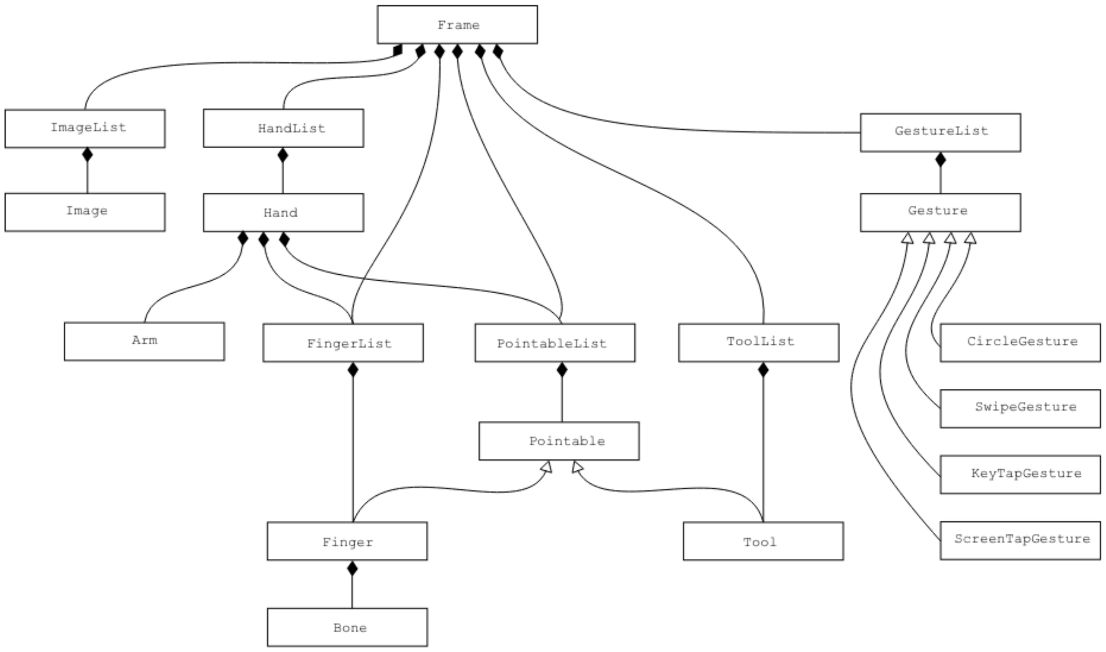


Figure 6: Tracking Model - ”Frame” [6]

2.3 Visualisation Toolkit

Visualisation Toolkit is an open-source library developed on C++ but it also supports automated wrapping of the C++ core into different languages, for example, Java and Python. The library is designed for 3D computer graphics, image processing and visualisation. The library is large and complex, which requires developer to spend time on studying its basic object-oriented design and implementation methodology before developing. Huang (2011) in [27] shows an example on using the surface and volume rendering function from the library.

2.3.1 Structure and Pipeline

Regarding the library is highly and heavily object-oriented, it has its own architecture and delivering pipeline. First, all the files must be decoded by a "vtkReader" into a "vtkDataSet" model. Various models can be produced including Poly Data, Structured Grid and Unstructured Grid, these three are the most commonly used models. The model can then be modified by different filters or functions recursively. Another way will be passing the model to a "vtkDataMapper" that being mapped to a "vtkActor". This process will convert the model into a viewable object. In order to display and interact with the object, a "vtkRenderer" and a "vtkRenderWindowInteractor" will act as the terminals of the pipeline. In the meantime, different widget and color map are also supported by the library to facilitate different visualising conditions. A simple pipeline can be found in the following diagram (figure 7).

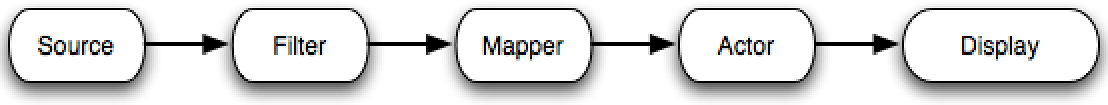


Figure 7: Example of simple VTK Pipeline

3 Problem Analysis

The product of this project is a large-scale system, and the libraries require time to study. Therefore, we assumed that the structure of some components like VTK might change from time to time. And thus, it is important to choose an adaptive software engineering approach for this project. Also, we have to decide which program language and platform is the best for implementing this project. Nonetheless, there are several problems we have to consider and to decide before developing the system.

- How to detect movement and recognise hand ?
- How to visualise the data models ?
- How to convert the output from recognition to the visualisation system ?
- How to create an effective and user-friendly GUI ?
- How to maintain a smooth user interaction

Last but not least, the product should fulfill all of the main requirements that high-lighted as follow:

Functional Requirements

1. The system should be able to accept multiple file extensions.
2. The system should be able to provide at least 3 commonly used visualisation functions.
3. The system should be able to save all user changes on the filters/functions.
4. The system should be able to recognise and provide basic gesture interaction.

Non-Functional Requirements

1. The system gesture recognition module should be robust and efficient.
2. The overall performance of the system should be smooth and effective, especially on the visualisation module.
3. The source code of the system should be written clearly and logically. It should be well commented and accompanied by appropriate document(e.g. Java Doc).
4. The system should have a significant error handling approach so as to achieve high maintainability.

4 Solution Design

4.1 Methodology

Referring to the review in Section 2, we have chosen to use Leap Motion for gesture analysis, Java as the language for implementation and Swing for GUI development.

There are plenty of reasons on choosing the above options. First, Leap Motion is one of the most popular depth camera in the market. It was the best option not only of its portable size, but also its was designed specifically for PC users. In the meantime, the API that comes with the device enhances the progress of gesture analysis. The API provides all the basic information required for developing custom gestures and motion tracking. These information included position and orientation of finger and palm, all of them are given in form of vector. Mentioned by Vladimir (1997) [22], fingertip detection was used to be non-trivial in either 3D or 2D space. Now, Leap Motion provides an easy and convenient solution. By using these information, the gesture recognition and management process could be easily implemented in short period of time.

On the other hand, we choose Java as the language for implementation because both Leap Motion and VTK have automated core wrapping which support Java development. Moreover, we could spend more time on studying the Leap Motion and VTK libraries, since we do not have to learn another programming language from scratch.

Lastly, Swing is not the best option for GUI implementation but it still provides functions that we need for designing a simple GUI. In fact, Qt will be another option, but it is not compatible (See section 8.1 for details). Hence, the whole GUI was developed by Swing with the aid of AWT only.

4.2 Project Plan

The project was large in scale consisted of different modules, therefore, we have separated it into three stages. Gesture Recognition, focusing on Leap Motion and the gesture analysis processes, except the four default gestures provided by the API we proposed to design nine more custom gestures.VTK, focusing on the VTK library, designing a dynamic pipeline and implementing at least three important functions. Integrated System, making use of the previous two modules and combine them into a single system.

Throughout each stage, a prototype was built, unit and integration testing was performed to make sure each component was working before the final integration. After the final integration, the integrated system was undergone additional testing such as functional and usability testing. A code review was carried out as well to optimise the system efficiency. A Gantt Chart can be found in the "Additional materials" which describes the details of the plan.

4.3 System Architecture

In order to increase the system's flexibility and maintainability, a software architectural pattern, Model–view–controller was being used for implementing the user interface. The architecture divides the system into three interconnected parts, which could separate internal representations of information from the interface presented to users.

The system consisted of three part, model, controller and view (Figure 8). A **Model** directly managed the data, logic and rule of the system. For instance, "VTKDataSet" was the model of all visualising object model; "CustomGesture" was the model of all gestures that we defined. A **View** could be any form of output representation of the model. For instance, "Dialog" could be used to present system preferences to users for customization (See section 5.4.5 for details). A **Controller** accepted the user input and converts them into commands for the view or model. Each component had its own controller, whilst a "SystemController" was used to connect all the other controllers. Under this architecture, whenever there was a change, it could possibly replace only a single part rather than the whole component. These approach minimized the work which would be required for any further system modification.

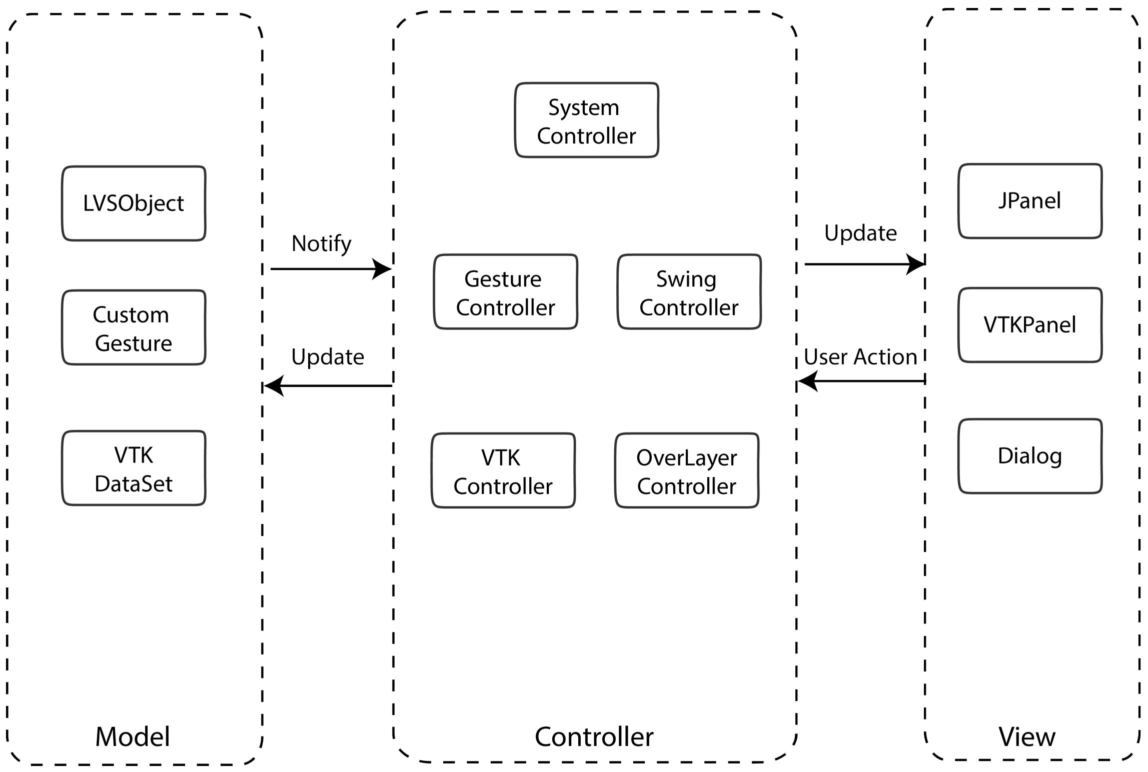
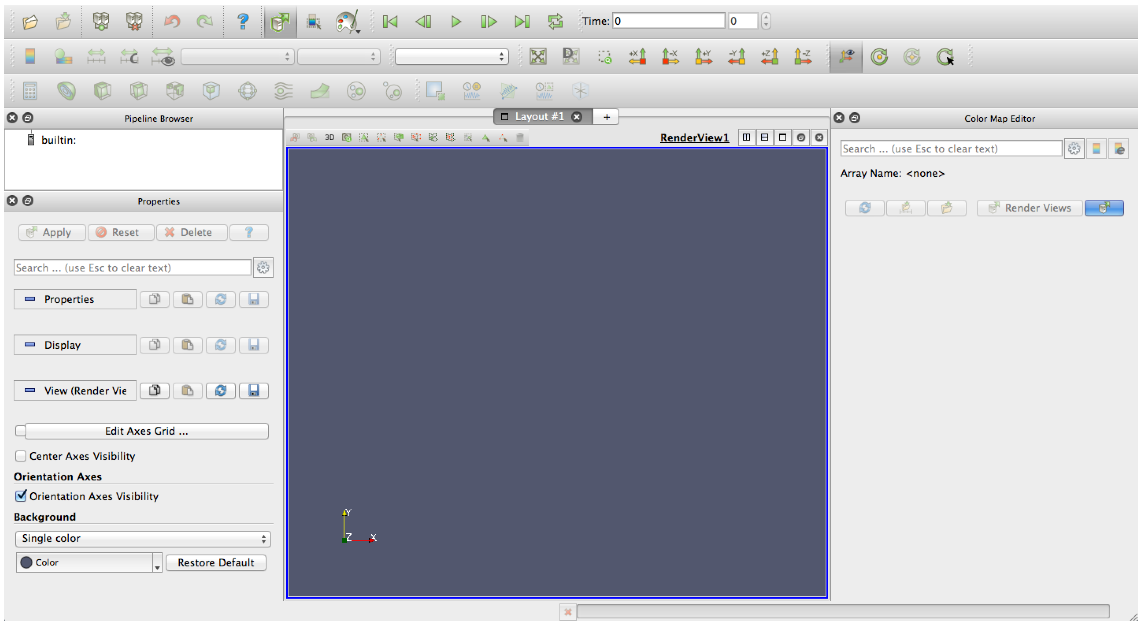


Figure 8: LVS system architecture

4.4 GUI Design

Except considering the functionality of the system, this project also aimed to provide a clean and simple GUI to users. Visualisation usually requires many algorithm and options, for instance, when a user is applying a contour filter to a model. The user might need to change the range or values of the filter in order to view different layers of the model. Sometimes, the user might even want to change the color mapping for a more clear view. Most of the existing software have implemented many buttons and panel to allow users to perform such modifications (Figure 9). Although it could provide good functionality, it was too complex to use, much less easy for new users to learn. Therefore, this project aimed to create a more simple and user-friendly GUI with the aid of gesture interface.



Time: 0 0

[ ] Center Axes Visibility
Orientation Axes
[✓] Orientation Axes Visibility
Background
Single color
Color

Array Name: <none>

Figure 9: Example of Complex GUI - ParaView

The solution could be done by only providing a number of important buttons or panels to users. Furthermore, some buttons could be merged to the Display Panel which only appear when that filter was enabled. Additionally, several frequently used functions might even mapped directly to the gesture interface. In our system, we introduced an over-layer mechanism to eliminate unnecessary graphic representation (See section 5.4.3 for details).

5 Implementation

5.1 Overview

The implementation was divided into three stages, focusing on different components of the system. A prototype was built in each stage. The followings discuss the chosen methodology in each stage.

5.2 Gesture Analysis

Gesture provide a higher level of interaction rather than directly tracking hand movement or particular hand features. With the aid of gestures, various commands can be sent by users to the system. The API that comes with Leap Motion offers 4 default gesture:

- Circle - A single finger tracing a circle
- Swipe - A long, linear movement of a finger
- Key Tap - A tapping movement by a finger as if tapping a keyboard key
- Screen Tap - A tapping movement by finger as if tapping a vertical computer screen.

Even though, these gestures provided some simple control mechanism, it was not enough for a visualisation system. To compensate this, we have implemented nine more gestures that could be easily remembered by users, including: Pointing, Two Fingers, Three Fingers, Four Fingers, Flow, Stop, Fist, Hold and Clap Gesture.

Pointing Gesture Only index finger is extended and pointing towards the screen.

Two Fingers Gesture Only index and middle finger are extended moving inwards to or outwards from the screen.

Three Fingers Gesture Only index, middle and ring finger are extended.

Four Fingers Gesture Except thumb all other fingers are extended.

Flow Gesture All fingers are extended and the palm is in horizontal position with all fingers pointing towards the screen.

Stop Gesture All fingers are extended and the palm is in vertical position with all fingers pointing upward.

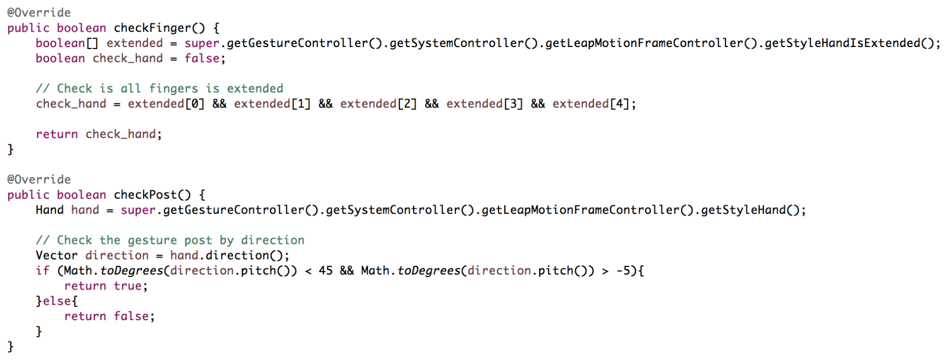
Fist Gesture All fingers are not extended, holding like a fist.

Hold Gesture Only thumb and index finger are extended.

Clap Gesture Fingers in both hand are extended and the palm moving towards each other like clapping.

Referring to section 2.2.2, although the ”Frame” object return by the API provides gesture information, the API itself is not viewable. Hence, we have no information on how the default gestures were being recognised and managed. Therefore, we have to implement those custom gestures and managing them by ourselves. There are several machine learning approaches used by researchers, for instance, applying Principal Component Analysis and Support Vector Machines for classification. These methods are frequently used and being proven throughout the last decades. However, based on the information the ”Frame” object provides, we used another way to implement the gesture analysis. Except default gestures information, the ”Frame” actually provides a ”HandList” and ”FingerList”. A ”HandList” contains a number of hands detected by the sensor, palm position and orientation could be extracted from a ”Hand” object. A ”FingerList” can be directly obtained from ”Frame” or even ”Hand”, each ”Finger” has a type indicated which finger it is, such as thumb and index. An extended status could also be acquired from it. With the aid of these information, we could design our custom gestures by setting different pose ranges and conditions for each gesture.

Lee and Kunii (1993) [8] demonstrated that 3D locations of five fingertips, and palm define a hand pose. Hence, when implementing each custom gestures we considered in two aspects, finger status and hand pose. In finger status, we check the extended status of each recognised finger; while in hand pose, we check the position and orientation of detected palm. Giving Flow Gesture as an example (Figure 10), the system would first check whether all fingers extended, following by a hand pose checking on the orientation of detected hand.



```
@Override
public boolean checkFinger() {
    boolean[] extended = super.getGestureController().getSystemController().getLeapMotionFrameController().getStyleHandIsExtended();
    boolean check_hand = false;

    // Check is all fingers is extended
    check_hand = extended[0] && extended[1] && extended[2] && extended[3] && extended[4];

    return check_hand;
}

@Override
public boolean checkPost() {
    Hand hand = super.getGestureController().getSystemController().getLeapMotionFrameController().getStyleHand();

    // Check the gesture post by direction
    Vector direction = hand.direction();
    if (Math.toDegrees(direction.pitch()) < 45 && Math.toDegrees(direction.pitch()) > -5){
        return true;
    }else{
        return false;
    }
}
```

Figure 10: Example Codes of Custom Gesture

All the implemented gestures are held by a list, which the list would then be used for the recognition process, therefore, the sequences of the list represent the priority of gesture being recognised. Same as the Leap Motion's API, whenever a gesture was being recognised, a gesture type in format of Enum would be returned. The return value would then be used by other part of the system, such as displaying in the GUI. In the meantime, a command would be sent to the controllers, if there was any function mapped to the gesture. If we wished to change the mapped function, we could just change the command of that model easily. An activity diagram (Figure 11) was used to show the process of the gesture recognition.

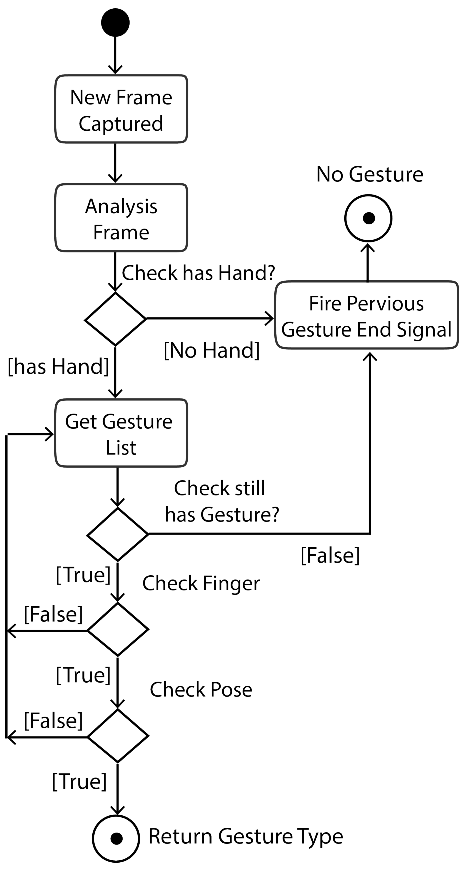


Figure 11: Activity Diagram of Gesture Analysis Process

The implemented gestures was actually divided into two categories: Static Gesture and Dynamic Gesture. It was important to mention the implementation of Static Gesture and Dynamic Gesture were actually opposite to their name. Where Static Gesture was recognised per frame while Dynamic Gesture involves multiple frame having a start status and end status. Due to this characteristic, Static Gesture was normally mapped to movement functions, for instance, a user performed Pointing Gesture and moved along the screen in order to move the cursor. On the other hand, Dynamic

Gesture was recognised by a sequences of movement which makes it unfavourable for movement function. Hence, it was usually mapped to single operation. For example, Clap Gesture was recognised when two hands moving towards each other, users could use this gesture to terminate the software.

There was one more thing, the system had its own recognition mechanism to assist left-handed or right-handed user. Most of the Static Gesture were single handed, which means the system had to choose the correct hand to analysis the gesture. To solve this, the system allowed users to set the hand preference (See 5.4.5 for details). If a user was a right-handed person, the "Style-hand" would be representing their right hand, whilst "Sub-hand" would be their left hand. Based on the user settings, the system would automatically choose the "Style-hand" to perform analysis for Static Gesture. Differently, some Dynamic Gesture would use both "Style-hand" and "Sub-hand" like Clap Gesture.

Although we had considered to include more two handed Static Gesture in our system, its performance was unsatisfying. During the development stage, we had tried to implement a two handed gesture for movement functions. The gesture consisted of both hand, which the "Sub-hand" would be used to decide the type of operation while the "Style-hand" would be used to capture the targeted movement. However, we found that under this approach, when user's hands overlap each other the sensor would lost track to both hands. These was due to the limitation of the sensor. Normally, the sensor was placed on top of the desktop and this causing the sensor having trouble to detect the hand when it was overlapping. Therefore, we decided to implement single-handed Static Gesture only. The following table showed the implemented gestures and mapped functions.

Gesture	**Hand**	**Category**	**Function**
Pointing	Single	Static	Moving Cursor
Two Finger	Single	Static	Zoom
Three Finger	Single	Static	Slice Translate
Four Finger	Single	Static	N/A
Flow	Single	Static	Rotation
Stop	Single	Static	Show Gesture Cheat List
Fist	Single	Static	Repositioning
Hold	Single	Static	N/A
Clap	Both	Dynamic	Terminate System
Click(D)	Single	Dynamic	Interact with Buttons
Swipe(D)	Single	Dynamic	Switch displaying Actor

D - Default Gestures provided by Leap Motion API

5.3 Visualising with Visualisation Toolkit

VTK is a huge library that consists of many visualising algorithms. In the meantime, it has its own object-oriented design and implementation methodology, hence, a clear pipeline has to be constructed to optimise the efficiency. The subsection 5.3.1 will focus on the pipeline we have constructed for this project while subsection 5.3.2 - 5.3.4 will focus on the algorithm we choose to apply .

5.3.1 LVS's VTK Pipeline

In section 2.3.1, we had introduced the simple pipeline of VTK, but in fact, each level of the pipeline had various implementations. Therefore, among all the classes we had chosen several emphases to implement along with the pipeline (Figure 12) constructed specifically for LVS. There were couples of levels we would like to highlight.

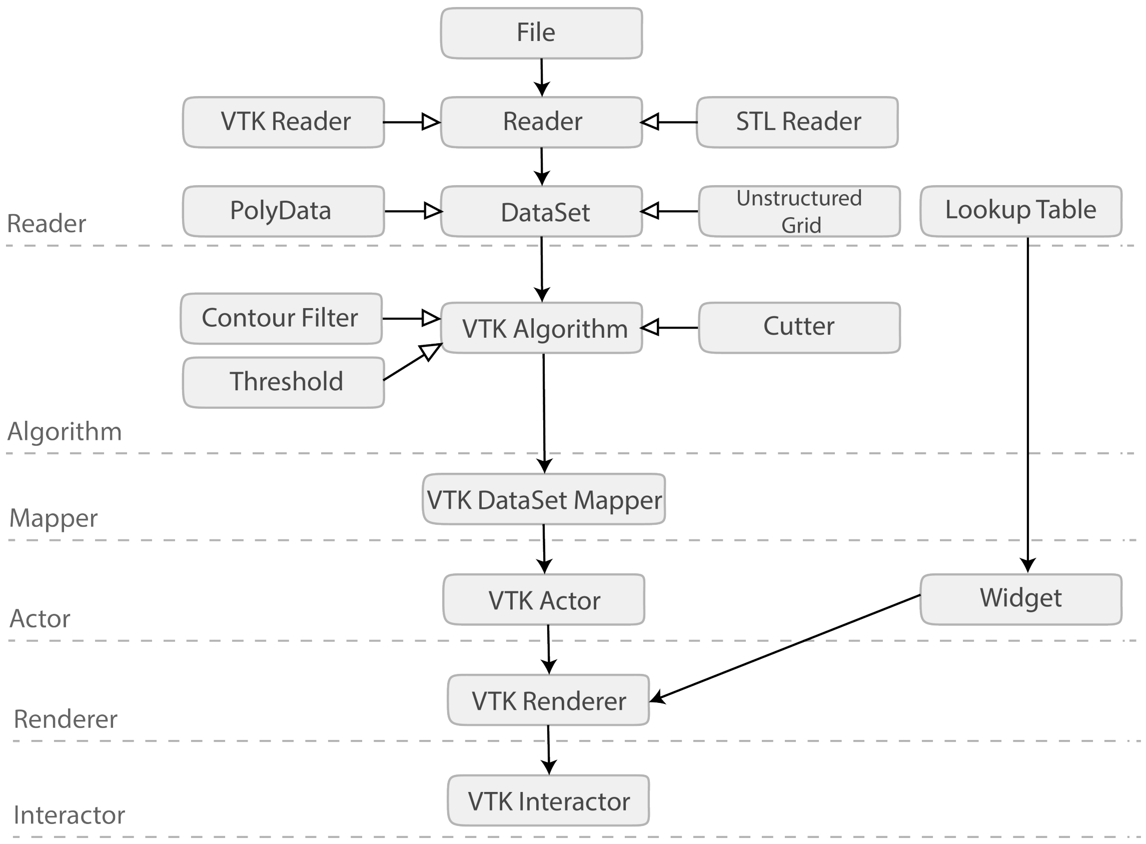


Figure 12: VTK pipeline of LVS

Reader VTK supports more than fifteen data file extensions, including VTK, Bitmap, JPEG and etc. Among all the extensions, we have chosen to implement two of them: VTK and STL. VTK was a Simple Legacy Format commonly used for medical images. The file mainly consisted of five basic parts: 1) file version and identifier 2) header 3) file format 4) dataset structure 5) dataset attributes. Another type STL was commonly used in 3D printing. These two file types were easy to read and write either by hand or program. Each of them required an individual reader to convert the file to a dataset object which then could be used to apply various algorithms. Indeed, if the system required to support more extensions, the current pipeline could be easily modified by adding few lines in the class VTKController. Other than extension, each data model had its own data structure, including: Poly Data, Unstructured Grid and Structured Grid. Sometimes, a specific data structure reader was required instead.

Widget VTK had numbers of widget that could enhance user experience. Each widget was the same as an actor, but some of them required support from other levels. For instance, scalar bar widget required a look-up table to map the scalar range with appropriate color from color map. In our system, we had implemented scalar bar and orientation widget to improve our system's user experience.

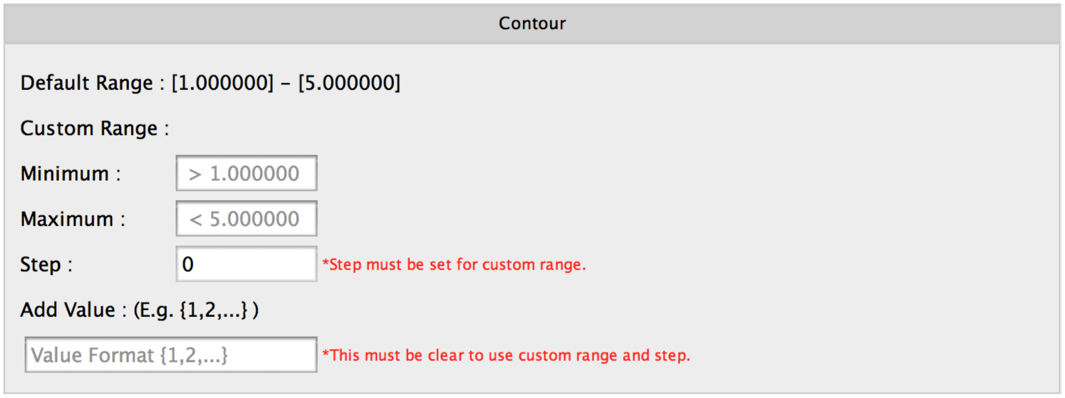
Algorithm The library provided different algorithm, expedited the development and enhanced the variety of functions. However, we could not implement all of them, therefore, we had to select three of them which were commonly used: Threshold, Contour and Slice. In some other software's pipeline, users could apply the algorithm recursively. Since the recursive pipeline was too complicated and time consuming, thus, our system had chosen to implement a straight forward pipeline at the moment. Still, the system could allow to display multiple algorithm result of the same file at once.

5.3.2 Threshold

Threshold showed all values in particular range. Users could set the minimum and maximum range to filter out unnecessary data. It was a simple function with not much parameter could be applied, but still an important algorithm in visualising.

5.3.3 Contour

Contour was similar to threshold but showing lines or surfaces of constant scalar value rather than all the value within the range. In VTK, users usually apply a scalar range or exact value to view particular layer of the model. In our implementation, users could edit the values in the edit dialog (Figure 13). In addition, when a user was setting a scalar range, a step which indicated numbers of exact value was also required. When valid range and step had been applied, the system would calculate the exact values automatically.



Contour

Default Range : [1.000000] – [5.000000]

Custom Range :

Minimum : > 1.000000

Maximum : < 5.000000

Step : 0 *Step must be set for custom range.

Add Value : (E.g. {1,2,...})

Value Format {1,2,...} *This must be clear to use custom range and step.

Figure 13: Contour Editing

5.3.4 Slice

Slice was a function which helped users to investigate the cross-section of a 3D model. It required a plane and a cutter function. The plane was used to set the position and orientation of the cross section, while cutter was used to extract the data lies on the plane.

Whenever a user wanted to change the cross section, apart from traditional way by editing the preference and moving by mouse, our system allowed users to interact with the panel by gestures. When the advance slice translate function was enabled, users could use Three Fingers gesture to translate the slice (Figure 14).

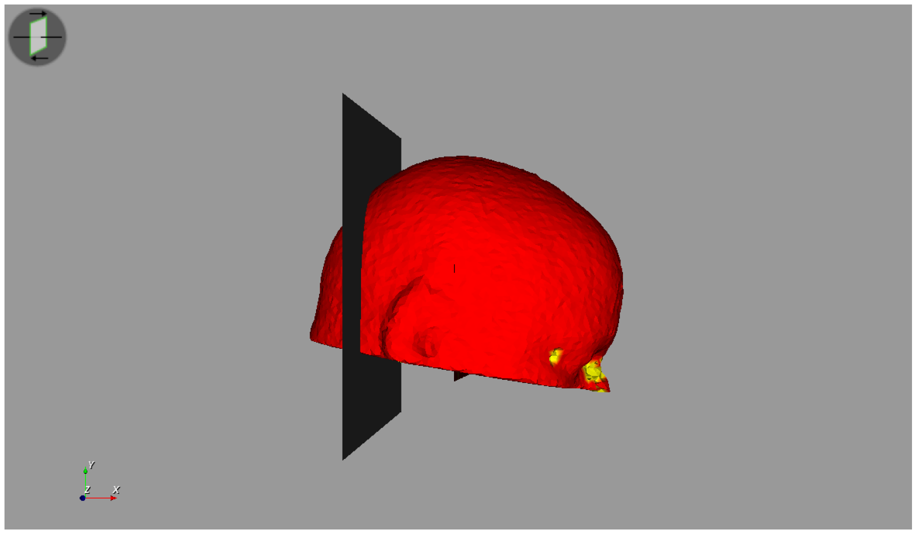


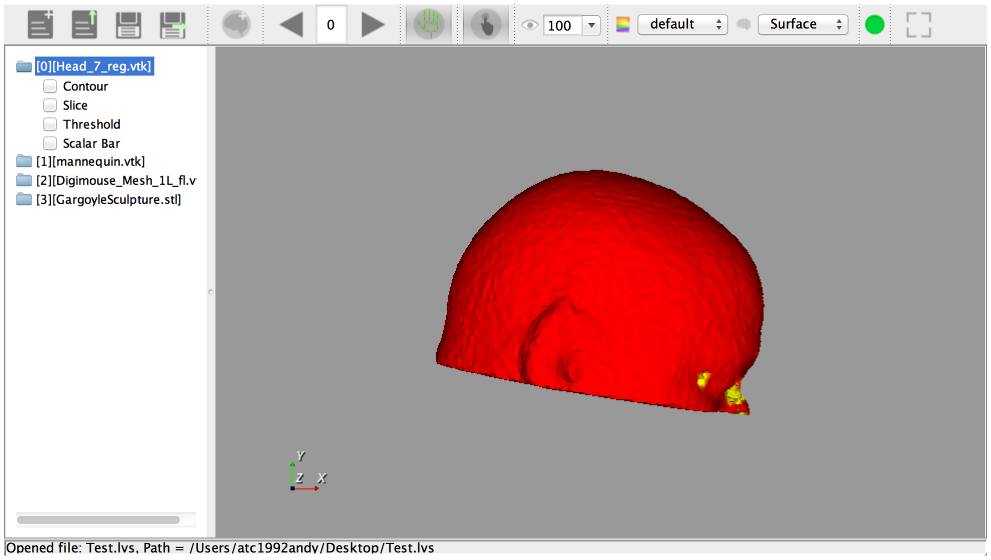
Figure 14: Screen Shot of Slice Translate Function

5.4 Integrated System

Making use of the prototype of previous stages, an integrated system name Leap Visualisation System (LVS) was built. Due to compatibility reason, Swing and AWT were used to develop the GUI rather than another popular library Qt. The system consisted of three main modules, gesture analysis, VTK and GUI. The core of first two modules had been explained in the above section. Thus, the following subsection would first focus on the Main Frame and then on several highlighted features.

5.4.1 Main Frame

The integrated GUI (Figure 15) aimed to be simple and clean. The main frame contains four components: Toolbar, File Tree, VTK Panel and Status bar. The Toolbar provided basic function, such as open or save a LVSFile, import a data model file and changing color map. The bigger size of the buttons were designed for gesture interaction. The File Tree on the left was a realizable panel, which showed all the imported data model files. Every file had a list of node showing the filter/algorithm options. Each file also had an index number which was used for referencing in the toolbar for switching actors. The VTK Panel was used to display the visualised object. On the left bottom corner was an orientation widget indicating the current orientation of the 3d space. Finally, the status bar showed the system status, for example, after a file being successfully loaded, it would show the file name and path, or else an error message would be shown.

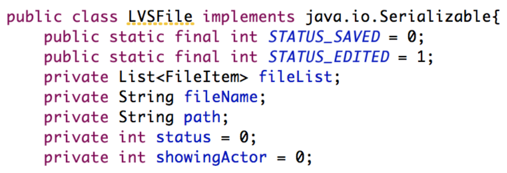


Opened file: Test.lvs, Path = /Users/atc1992andy/Desktop/Test.lvs

Figure 15: Screen Shot of LVS GUI

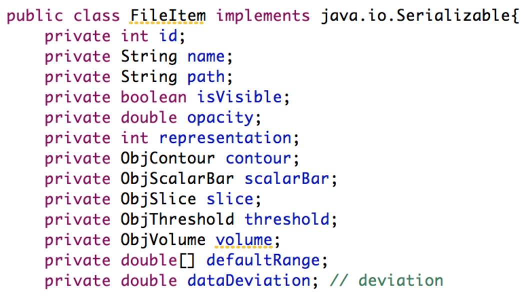
5.4.2 File System

Throughout a visualising experience, users might need to import a number of data model files. Additionally, users might have to apply or configure various filter for each file. Such that, to facilitate users from repeated work, it was important to save the substantial controls or operations that the user had carried out. To overcome this, we had introduced a unique file system in our software. The file object called "LVSFile" (Figure 16) containing the file name, path and also a list of "FileItem" (Figure 17). Each file item represented a single imported data model file. Except basic file information, the "FileItem" contained visualising parameters such as visibility and opacity as well. Nevertheless, there were different filter/algorithm objects too, for example, a threshold object stored the user's configuration on minimum and maximum range.



```
public class LVSFile implements java.io.Serializable{
    public static final int STATUS_SAVED = 0;
    public static final int STATUS_EDITED = 1;
    private List<FileItem> fileList;
    private String fileName;
    private String path;
    private int status = 0;
    private int showingActor = 0;
```

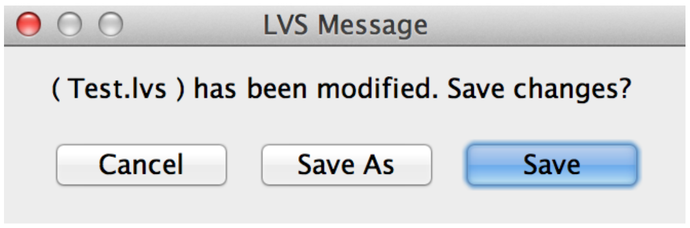
Figure 16: Fragment of "LVSFile"



```
public class FileItem implements java.io.Serializable{
    private int id;
    private String name;
    private String path;
    private boolean isVisible;
    private double opacity;
    private int representation;
    private ObjContour contour;
    private ObjScalarBar scalarBar;
    private ObjSlice slice;
    private ObjThreshold threshold;
    private ObjVolume volume;
    private double[] defaultRange;
    private double dataDeviation; // deviation
```

Figure 17: Fragment of "FileItem"

When a user created a new project, the user could open an existing "LVSFile" or created a new one by clicking the "New File" button in either toolbar or menu-bar or even using hot keys. Alternatively, a user could also directly import a data model file, if there was no opened "LVSFile" the system would automatically created a new one. However, if a user created a new one without saving the current copy, the system would alert the user and remind them to save it. Finally, after every open or import operation, the system would automatically save the path that the user has accessed recently.



LVS Message

(Test.lvs) has been modified. Save changes?

Figure 18: System Message - Saving Alert

5.4.3 Over-Layer

To reduce the complexity of the GUI and eliminate unnecessary buttons appearing, an over-layer had been designed for the system. There were many ways to implement the layer, but we had chosen to use a transparent JFrame placing on top of the main frame. The over-layer was actually a canvas which the system could draw different GUI presentation, like points or buttons, based on the required functionality. The main reason of choosing this approach was due to the different drawing sequence of the graphic components. VTK Panel used AWT while the rest of the components used Swing instead. Such that, if we included the canvas on the main frame, part of it would be overlapped and covered by the VTK Panel.

Currently, the over-layer had three main features: Movement Tracking, Gesture Tracking and over-layer buttons.

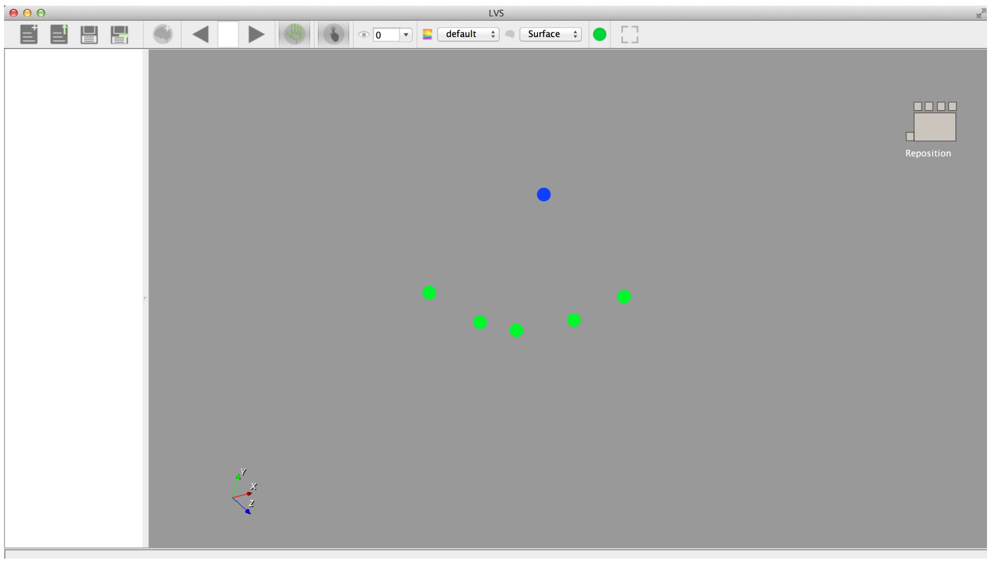


Figure 19: Screen Shot of Demonstrating Over-Layer

From the above figure, we could see that when the movement tracking and gesture tracking function was activated, the green dots would appear indicating the position of user's fingers, whilst the blue dot representing the center of user's palm. Users might change the color from the edit dialog (See section 5.4.5 for details). Furthermore, on the right-side of the screen a small diagram would appear when a gesture was being recognised. Meanwhile, a short text under the diagram showed the operation that the recognised gesture could perform. These feature allowed users to know the current interacting situation which helped the users to communicate with the system. Apart from that, the over-layer button would appear when certain function is activated, for instance, slice translate function (See section 5.3.4 for details).

5.4.4 Color Map

During different visualising situations, users might require various color representation. In other existing visualising software, they had implemented their own set of color maps. Moreover in software like Paraview, those color map could be exported into a JSON file. The system allowed users to import those color map in form of JSON file. After a user placed the JSON file in the rsc/colorMapping directory, the system would parse the file into color map option and viewable in the toolbar dynamically.

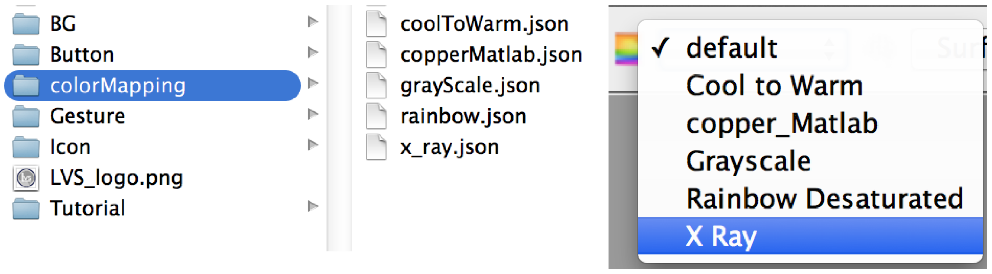
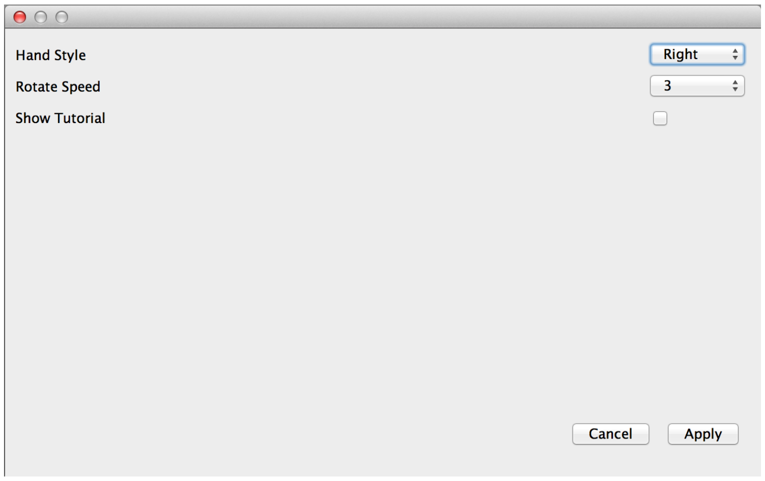


Figure 20: Color Map

5.4.5 System Preference

The system provided plenty of options to users, they could modify these preference through specific edit dialog. These edit dialog could be found in the menu-bar.

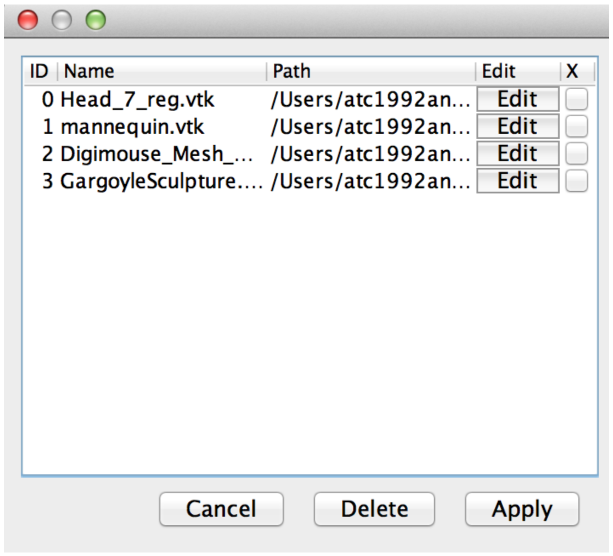


Hand Style	Right
Rotate Speed	3
Show Tutorial	☐

Figure 21: System Preference Edit Dialog

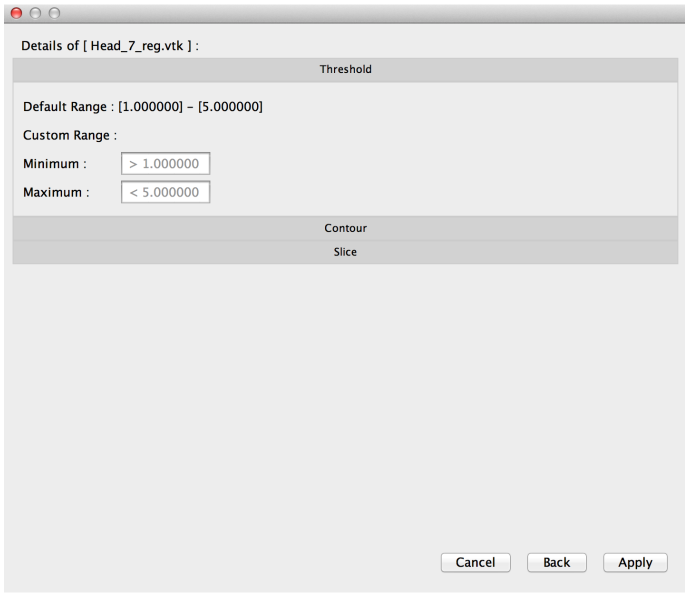
Each Edit Dialog were designed for users to modify specific information of the system. The System Preference Edit Dialog (Figure 21) allowed users to edit general information of the system.

The LVSFile edit dialog (Figure 22) was linked with FileItem edit dialog (Figure 23). Users could view all the imported file along with its ID and path. Users would delete multiple file at once but only updating one per time. By clicking the "Edit" button, the dialog would link to the FileItem edit dialog which could apply filter configurations.



ID	Name	Path	Edit	X
0	Head_7_reg.vtk	/Users/atc1992an...	Edit	
1	mannequin.vtk	/Users/atc1992an...	Edit	
2	Digimouse_Mesh_...	/Users/atc1992an...	Edit	
3	GargoyleSculpture....	/Users/atc1992an...	Edit	

Figure 22: LVSFile Edit Dialog



Details of [Head_7_reg.vtk] :

Threshold

Default Range : [1.000000] – [5.000000]

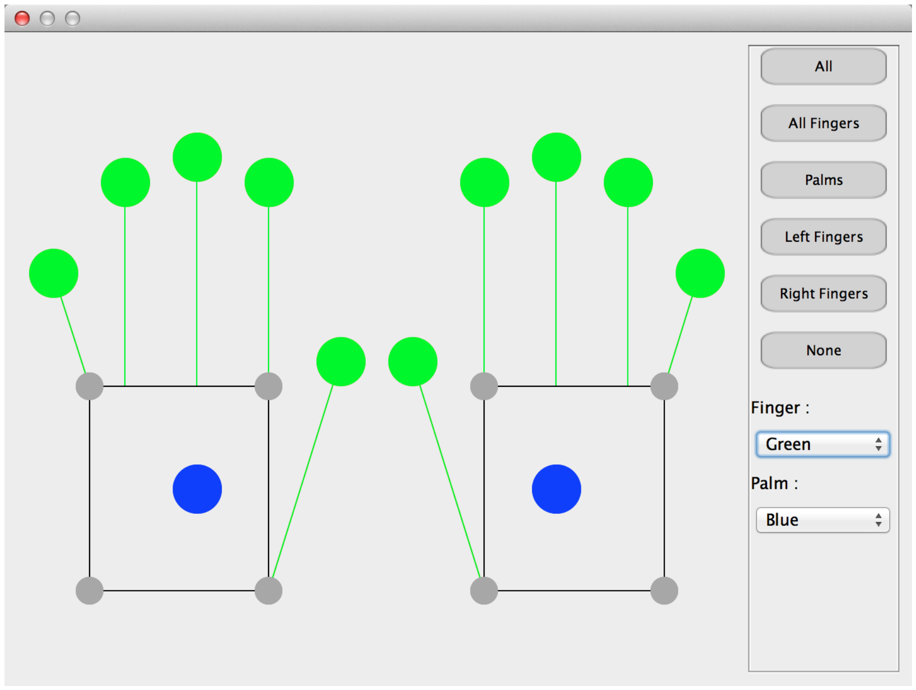
Custom Range :

Minimum : > 1.000000

Maximum : < 5.000000

Figure 23: FileItem Edit Dialog

Lastly, the Movement Tracking Edit dialog allowed users to change the presented information of movement tracking function (See section 5.4.3 for details). Users could click on the colored dot to enable or disable the tracking on that hand feature. In default, the finger color was green and palm color is blue. Meanwhile, only palms were indicated. We suggested to keep this configuration in order to have a better control experience.



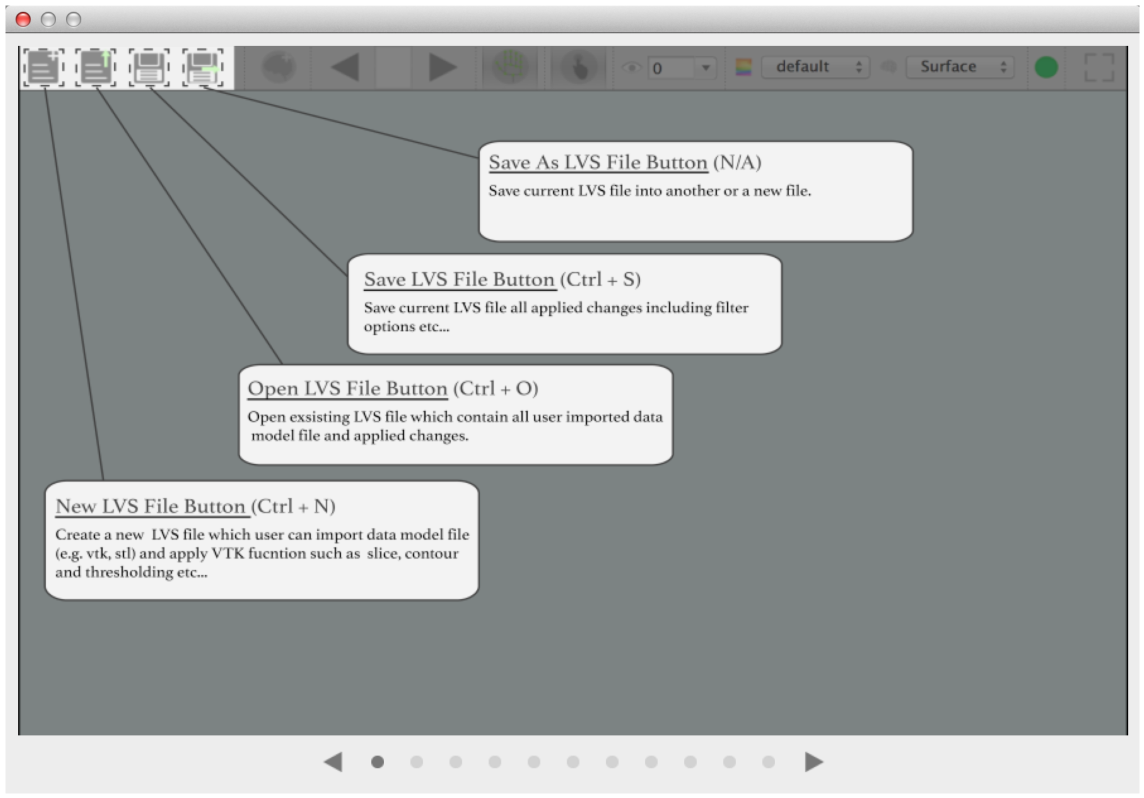
Finger : Green

Palm : Blue

Figure 24: Movement Tracking Edit Dialog

5.4.6 System Learning Support

To help users to understand and learn the system, a tutorial would be provided to users when the system launches. The tutorial showed the functionality of each button in the toolbar. It also showed the interaction method of the system like zoom and rotation.



Save As LVS File Button (N/A)
Save current LVS file into another or a new file.

Save LVS File Button (Ctrl + S)
Save current LVS file all applied changes including filter options etc...

Open LVS File Button (Ctrl + O)
Open exsisting LVS file which contain all user imported data model file and applied changes.

New LVS File Button (Ctrl + N)
Create a new LVS file which user can import data model file (e.g. vtk, stl) and apply VTK fucntion such as slice, contour and thresholding etc...

Figure 25: Tutorial Dialog

5.4.7 Presentation Mode

Presentation mode could be enable in either toolbar or menu-bar. The main difference of this mode was the layout of the main frame. Normally, the main frame include three components (See section 5.4.1 for details), but in presentation mode except "VTKPanel" all other components were hidden, this created more space to display the data object.

5.4.8 Interaction Mechanism

The integrated gesture interface, had a unique interacting mechanism. This mechanism applied on Flow Gesture and Three finger Gesture. The idea of this mechanism was creating neutral area in the middle of the Main Frame. The neutral area was calculated by multiplying the interacting area with a fixed ratio. When users perform those gestures, they could move their hand towards the edges to control the object to rotate or move towards the same direction. The closer to the edge the faster the rotation or translation speed. Although this mechanism was not humanized, it could allow users to maintain a more robust interaction.

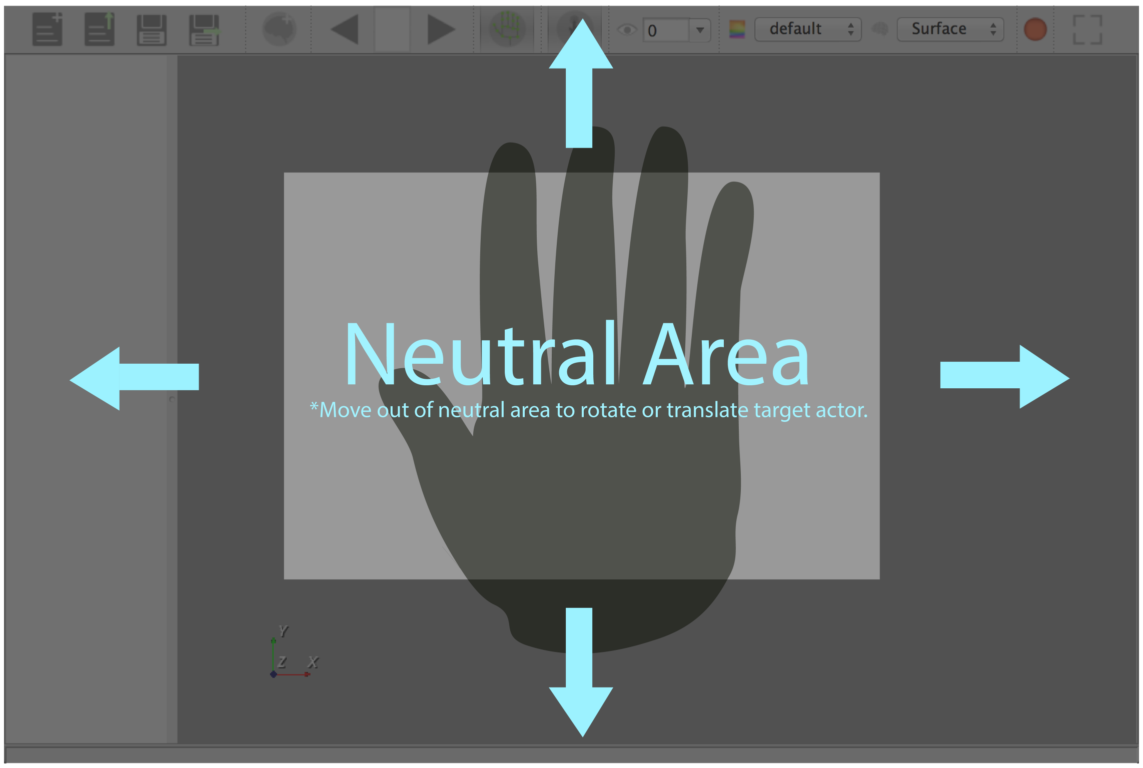

0
default
Surface
Neutral Area
*Move out of neutral area to rotate or translate target actor.
Y
Z
X

Figure 26: Interaction Mechanism

6 Testing

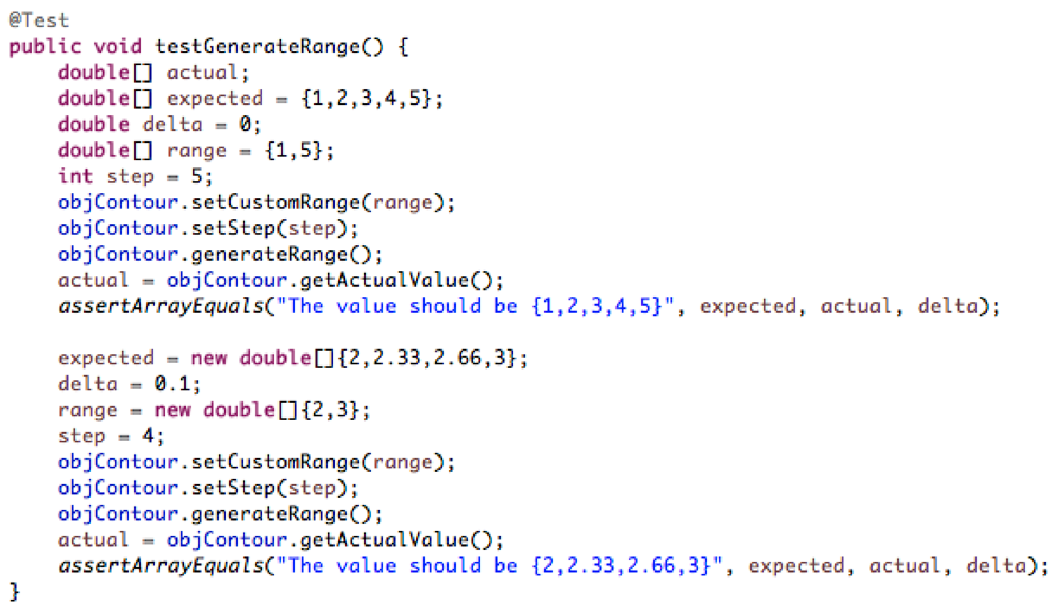
As mentioned in the section 4.2, in each stage a Unit Testing and Integration Testing had been done. In the final stage, a Functional test and Usability test had also been carried out. In addition, a test on gesture recognition rate had conducted as well. In the following subsections, for each test we would discuss about the result and how it was implemented.

6.1 Unit Testing and Integration Testing

Both the Unit Tests and Integration Tests were conducted by using JUnit testing framework.

Unit Test Codes that perform basic operation were tested under Unit test. Both normal cases and edge cases were tested for each method (if applicable). For example, in ”ObjectContour”, the method *generateRange()* will calculate a list of exact values based on given range and step. Therefore, two test cases had been setup on testing the performance with integer and double. All the unit tests regarding the system were included in the package “test.unit”.

Integration Test Integration test focused on the interaction between different modules and classes. However, many system inputs were in form of vision and also graphic representations. Such that, only a limited numbers of test cases could be setup.



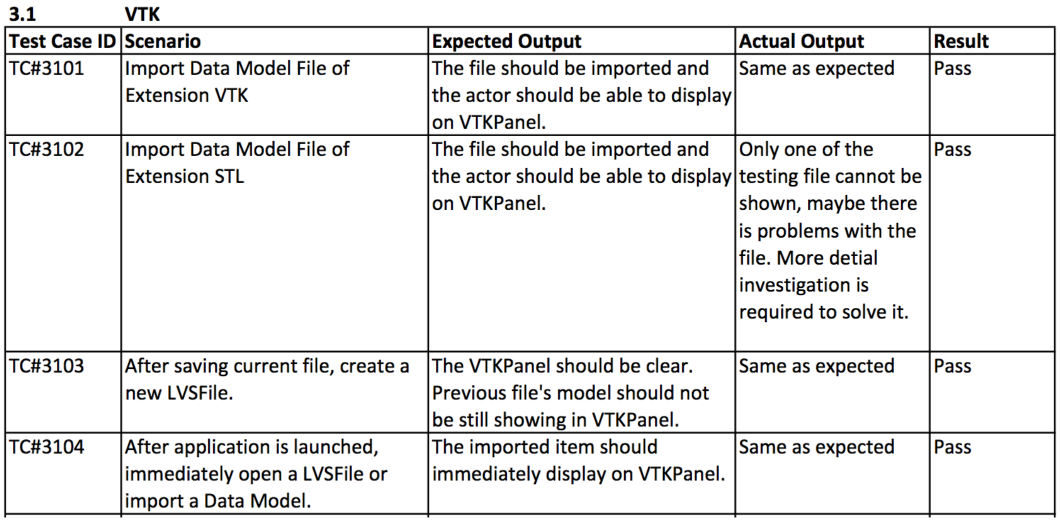
```
@Test
public void testGenerateRange() {
    double[] actual;
    double[] expected = {1,2,3,4,5};
    double delta = 0;
    double[] range = {1,5};
    int step = 5;
    objContour.setCustomRange(range);
    objContour.setStep(step);
    objContour.generateRange();
    actual = objContour.getActualValue();
    assertArrayEquals("The value should be {1,2,3,4,5}", expected, actual, delta);

    expected = new double[]{2,2.33,2.66,3};
    delta = 0.1;
    range = new double[]{2,3};
    step = 4;
    objContour.setCustomRange(range);
    objContour.setStep(step);
    objContour.generateRange();
    actual = objContour.getActualValue();
    assertArrayEquals("The value should be {2,2.33,2.66,3}", expected, actual, delta);
}
```

Figure 27: Example of JUnit Test

6.2 Functional Testing

As mentioned in previous subsection, many components could not be tested by the integration test. Functional Test were used to cover those untested modules. The Functional Test were done after the UI components of a specific functionality were fully integrated. The tests were conducted by writing all test cases for a specific system functions and compared it to the actual result displayed on the GUI with respective user input. Similar to other tests, both the normal cases and the edge cases were tested thoroughly. Please find all the high-level Functional Test cases in the "Additional materials".

The Functional Test were categories by GUI component and the function it interacted with. The example test cases below showed the function that had interacted with the component VTK Panel. For instance, when users used the import data model file function with file of VTK extension, the system should had corresponding responses to the change. In this case, the VTK Panel should display the imported file object immediately. The test passed when the correct object was being displayed, or else the test failed if there was no object, wrong object displayed or even the system crashed.



3.1 VTK

Test Case ID	Scenario	Expected Output	Actual Output	Result
TC#3101	Import Data Model File of Extension VTK	The file should be imported and the actor should be able to display on VTKPanel.	Same as expected	Pass
TC#3102	Import Data Model File of Extension STL	The file should be imported and the actor should be able to display on VTKPanel.	Only one of the testing file cannot be shown, maybe there is problems with the file. More detial investigation is required to solve it.	Pass
TC#3103	After saving current file, create a new LVSFile.	The VTKPanel should be clear. Previous file's model should not be still showing in VTKPanel.	Same as expected	Pass
TC#3104	After application is launched, immediately open a LVSFile or import a Data Model.	The imported item should immediately display on VTKPanel.	Same as expected	Pass

Figure 28: Example of Functional Test

The overall result of all 104 test cases were satisfying, almost all of the test cases had passed. Except for case #3102, one of the testing file could not be decoded by the reader. To solve this, we might need a period of time to study since other file with the same extension (.stl) works normally in the system. Therefore, this individual file had been ignored at the moment.

6.3 Usability Testing

Usability test was one of the most effective way to improve the system. This could also help us to achieve the expectation of different users. In order to gain feedback from different range of users, testing was done across individuals of various visualisation and gesture control experience. Five users with at least either visualisation or gesture control experience and five users with no such experience had been invited for the test.

Throughout the test, only one tester managed to break the system. The reason of causing the crash, was due to the compatibility problem of VTK with Java and OSX. The following table listed the result of the test.

Questions	**Average Score**
GUI design	3.7
”Look and Feel”	3.8
Performance	4.1
Gesture easy to memorise	3.3
System easy to learn	3.4
Gesture accuracy	3.5

The overall score of the test was around 3.6, some aspects were below mean, the reason mainly due to the nature of visualizing software. Although, we had tried to implement a simple UI but users still need a period of time to adapt to the system, especially for new users. In the meantime, we noticed that there were still much room for us to improve the gesture management and performance. Nevertheless, based on the feedback from users, we had summarised them in the following points:

- Gesture not humanised
- Gesture Tracking should provide more information (e.g. operation)
- GUI was too plain
- More Function should be implemented

6.4 Recognition Rate Testing

Considering our system was using an uncommon approach (See 5.2 for details) to perform gesture analysis, we had conducted a series of tests to investigate the accuracy of our approach. We had invited two users to participate in the test, one of them was a new user with no experience (User 1), another one was a trained user (User 2). Each user had to perform a list of operations, each operation was performed repeatedly for 10 times. Those tested operations were cases we foresee users would use most. If the user could complete the operation the trail would be recorded as success. However, if the gesture lost track during any point of the test, that trail would be failed. The test results and the operation list were recorded as below:

1. Perform Pointing Gesture and move from left to right
2. Perform Two Fingers Gesture and move towards the screen then backwards
3. Perform Stop Gesture for 5s
4. Perform Flow Gesture and rotate the object with sequence left,right,up,down
5. Swipe from left to right to switch actors
6. Click on the over-layer button
7. Perform Clap Gesture to terminate the system

Operation	**User 1 Success**	**User 2 Success**	**Total Success**	**Recognition Rate(%)**
1	10	10	20	100
2	10	10	20	100
3	5	10	15	75
4	7	9	16	80
5	10	10	20	100
6	2	5	7	35
7	4	7	11	55

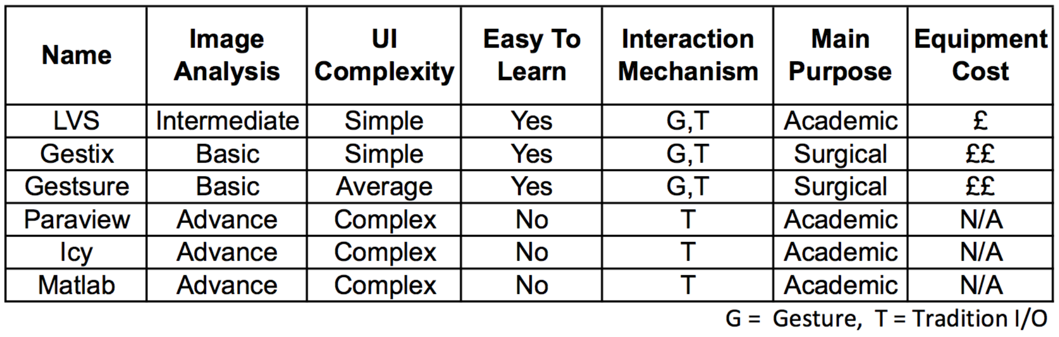
With the aid of the results, there were several points we might conclude. First, some gestures had a high recognition rate due to its significant feature like Pointing and Two Fingers. Secondly, in general cases the gesture interface required time to learn, the recognition rate of the trained user was relatively higher. However, we had also noticed that, under this approach some gestures with similar feature might have higher chance to be failed like Three fingers and Four fingers. In short, the gesture analysis and management still had much room for improvement. Furthermore, it was not the only emphasis of this project, therefore, the result was acceptable.

7 Evaluation

Evaluation is an important section to assess the system. In general, we were able to follow most of the original specification, and the system was successfully implemented. Our project had fulfilled all the functional requirements and almost all non-functional requirements. However, due to compatibility problem, we had switched to use Swing and AWT rather than Qt to implement the GUI. In this section, we are going to review the system based on several aspects. In section 7.1 we will compare our system with other existing software in the market. Following by section 7.2, we will evaluate the performance of the system. Finally, in section 7.3 we will discuss some specific aspects of the system in the perspective of HCI and software engineering.

7.1 Related Work

There are couples of similar software in the market, but each of them are having various targeted group of users, level of analysis and UI complexity (Figure 29).



Name	Image Analysis	UI Complexity	Easy To Learn	Interaction Mechanism	Main Purpose	Equipment Cost
LVS	Intermediate	Simple	Yes	G,T	Academic	£
Gestix	Basic	Simple	Yes	G,T	Surgical	££
Gestsure	Basic	Average	Yes	G,T	Surgical	££
Paraview	Advance	Complex	No	T	Academic	N/A
Icy	Advance	Complex	No	T	Academic	N/A
Matlab	Advance	Complex	No	T	Academic	N/A

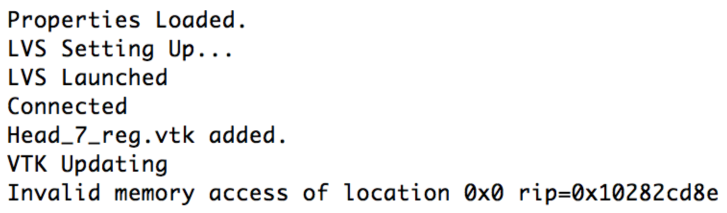
G = Gesture, T = Tradition I/O

Figure 29: Comparison table of related work

From the above table, we can see that most of the visualisation software with gesture interaction are designed for surgical use only. For other academic use software, since these software aimed to provide much variety to image analysis, all of them are having a complex UI and did not support gesture interaction. In contrast to these software, our system managed to provide intermediate image analysis with the aid of gesture interaction. In the meantime, our system's target users are student and researchers. A simple and clean GUI had been achieved, this can facilitate some situations like a presentation. In current stage, we have only implemented the emphasis, therefore, lacking of functionality our system are certainly not the best in the market. However, we believe after further development, our system will be more competitive in the market.

7.2 System Performance

In this project, we have demonstrated a robust and efficient visualisation system. The system was able to demonstrate a fast and significant visualisation process. When loading large data model files, some system might took a long period of time or even crashed; some of them might be lagging when users interacted with an object. Moreover, our system was reliable that rarely crashed. This was achieved by the exception handling mechanism of our system. In order to achieve high maintainability, our system was able to catch most of the exception and log it into a text file. The text file could be sent to the developing team for further investigation. As mentioned above, there were two types of exceptions unable to be caught. First, the VTK library support java development by providing a Java Wrapping, but the core is still in C++. This type of exceptions were generated inside the C++ core(Figure 30). Therefore, sometimes we were unable to identify the source of the problem, and this to caused the delay of the development progress.



```
Properties Loaded.
LVS Setting Up...
LVS Launched
Connected
Head_7_reg.vtk added.
VTK Updating
Invalid memory access of location 0x0 rip=0x10282cd8e
```

Figure 30: VTK exception

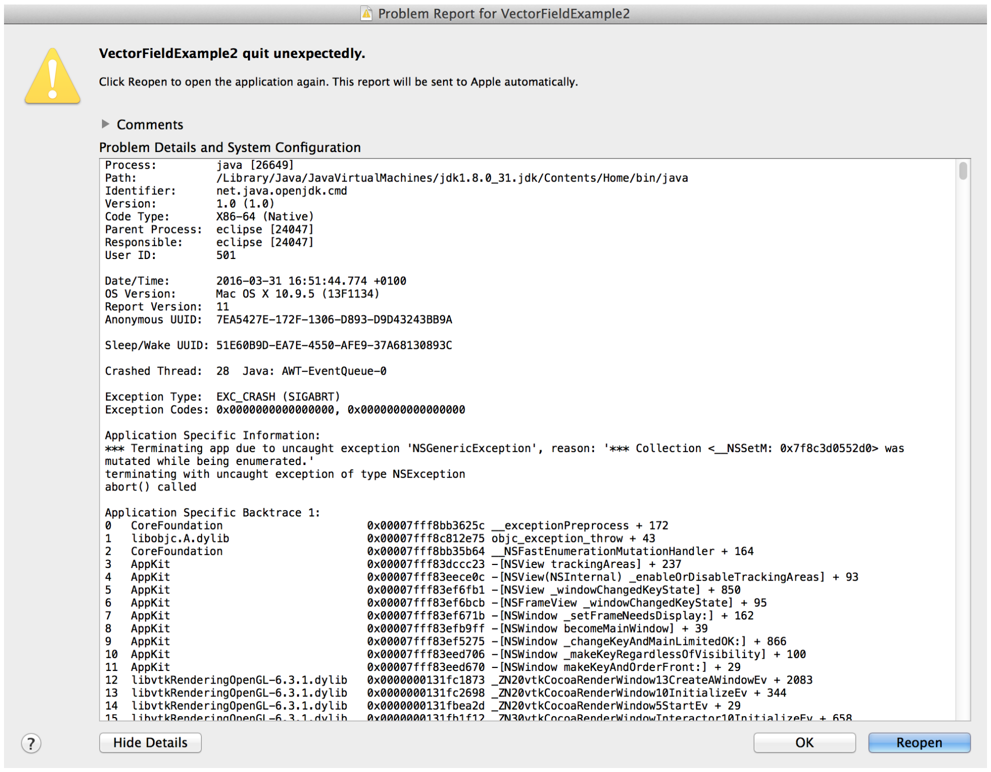
Another type of exception was generated by the Leap Motion API. These exceptions were caught and hidden by the API. This made us to spend plenty of time to discover such unknown errors. For instance, when we were trying to call some methods to get the hand features but there is no hand being detected by the sensor.

On the hand, we are also aware of our system's weakness. Although, we have implemented a simple gesture interface, there are still a lot of limitations and managing problems. Some of the gesture interaction is not smooth and accurate enough, or even not humanized as the expectation of some users. In the meantime, though the VTK library provided various visualising algorithms, our system's functionality are still bounded to the library. In other words, the efficient and performance of our system are limited by the library.

7.3 Other Aspect

7.3.1 Compatibility Problem and Limitation on Java Version

Java is a platform independent language, and both VTK and Leap Motion support Java development. Initially, we thought that Java was the best language to implement the system. However, during the development processes we have encountered several compatibility problems. First, the Java developers have abandoned some old graph draw methodology after version 1.7, which VTK library relied on these method. Therefore, around 50% of the time, when users launched our system under Java version 1.7 or above, the system had crashed (Figure 31).



VectorFieldExample2 quit unexpectedly.

Click Reopen to open the application again. This report will be sent to Apple automatically.

Problem Details and System Configuration

```
Process:         java [26649]
Path:            /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java
Identifier:      net.java.openjdk.cmd
Version:         1.0 (1.0)
Code Type:       X86-64 (Native)
Parent Process:  eclipse [24047]
Responsible:     eclipse [24047]
User ID:         501

Date/Time:       2016-03-31 16:51:44.774 +0100
OS Version:      Mac OS X 10.9.5 (13F1134)
Report Version:  11
Anonymous UUID:  7EA5427E-172F-1306-D893-D9D43243BB9A

Sleep/Wake UUID: 51E60B9D-EA7E-4550-AFE9-37A68130893C

Crashed Thread:  28  Java: AWT-EventQueue-0

Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000

Application Specific Information:
*** Terminating app due to uncaught exception 'NSGenericException', reason: '*** Collection <__NSSetM: 0x7f8c3d0552d0> was
mutated while being enumerated.'
terminating with uncaught exception of type NSException
abort() called

Application Specific Backtrace 1:
0   CoreFoundation                      0x00007fff8bb3625c __exceptionPreprocess + 172
1   libobjc.A.dylib                     0x00007fff8c812e75 objc_exception_throw + 43
2   CoreFoundation                      0x00007fff8bb35b64 __NSFastEnumerationMutationHandler + 164
3   AppKit                              0x00007fff83dccc23 -[NSView trackingAreas] + 237
4   AppKit                              0x00007fff83eece0c -[NSView(NSInternal) _enableOrDisableTrackingAreas] + 93
5   AppKit                              0x00007fff83ef6fb1 -[NSView _windowChangedKeyState] + 850
6   AppKit                              0x00007fff83ef6bcb -[NSFrameView _windowChangedKeyState] + 95
7   AppKit                              0x00007fff83ef671b -[NSWindow _setFrameNeedsDisplay:] + 162
8   AppKit                              0x00007fff83efb9ff -[NSWindow becomeMainWindow] + 39
9   AppKit                              0x00007fff83ef5275 -[NSWindow _changeKeyAndMainLimitedOK:] + 866
10  AppKit                              0x00007fff83eed706 -[NSWindow _makeKeyRegardlessOfVisibility] + 100
11  AppKit                              0x00007fff83eed670 -[NSWindow makeKeyAndOrderFront:] + 29
12  libvtkRenderingOpenGL-6.3.1.dylib   0x0000000131fc1873 _ZN20vtkCocoaRenderWindow13CreateAWindowEv + 2083
13  libvtkRenderingOpenGL-6.3.1.dylib   0x0000000131fc2698 _ZN20vtkCocoaRenderWindow10InitializeEv + 344
14  libvtkRenderingOpenGL-6.3.1.dylib   0x0000000131fbea2d _ZN20vtkCocoaRenderWindow5StartEv + 29
15  libvtkRenderingOpenGL-6.3.1.dylib   0x0000000131fb1f12 _ZN30vtkCocoaRenderWindowInteractor10InitializeEv + 658
```

Figure 31: VTK compatibility with OSX and Java

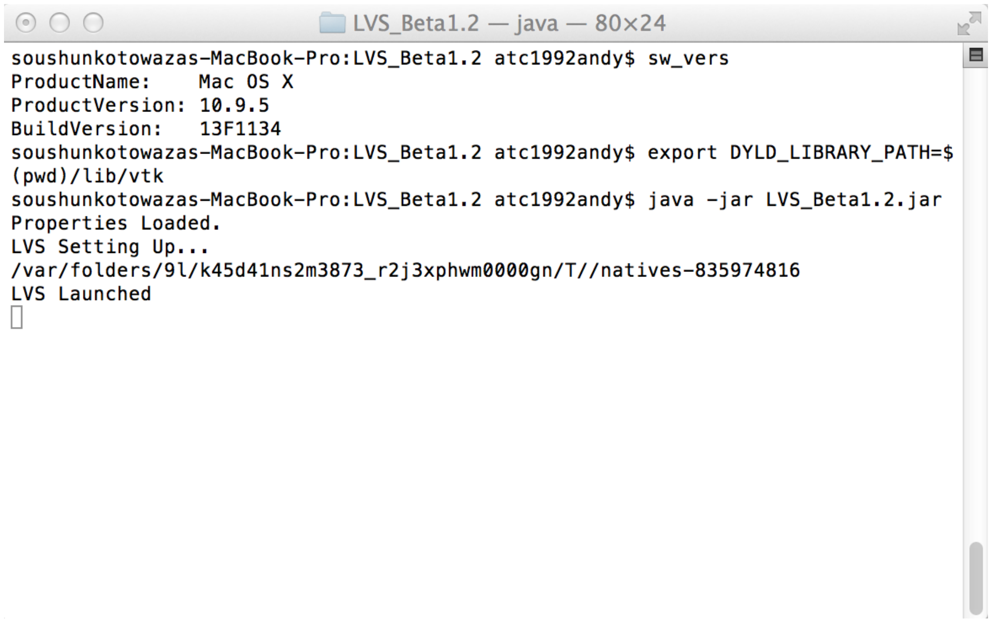
This problem has been known by the VTK library developing team for at least 5 years, yet, no solutions have ever been found or made. In addition, there were only a few VTK developers using Java on OSX. Hence, it cost us much effort to investigate it, and we had spent a couple of months to find the solution. The solution was to launch the system under Java version 1.6, and we realises that this might cause inconvenient to users. However, we have no other solution unless re-creating the project with a different programming language.

Another example was about the Qt library. Qt is a popular GUI framework among VTK developers that written in C++. It has a Java interface called QtJambi. However, it was well known by the library developing team that QtJambi could not function normally under some OSX machines. The problem was described as 'incompatible with some OSX architecture'. In the meantime, QtJambi itself did not include an essential class "VTKCanvas" that required by VTK. Such that, during the VTK developing stage, we had to abandon the Qt GUI and re-creating it by Swing and AWT.

The whole compatibility problem consumed a large part of the project developing time. Eventually, we can still complete the whole system on time, but it indicated that Java was not the best option.

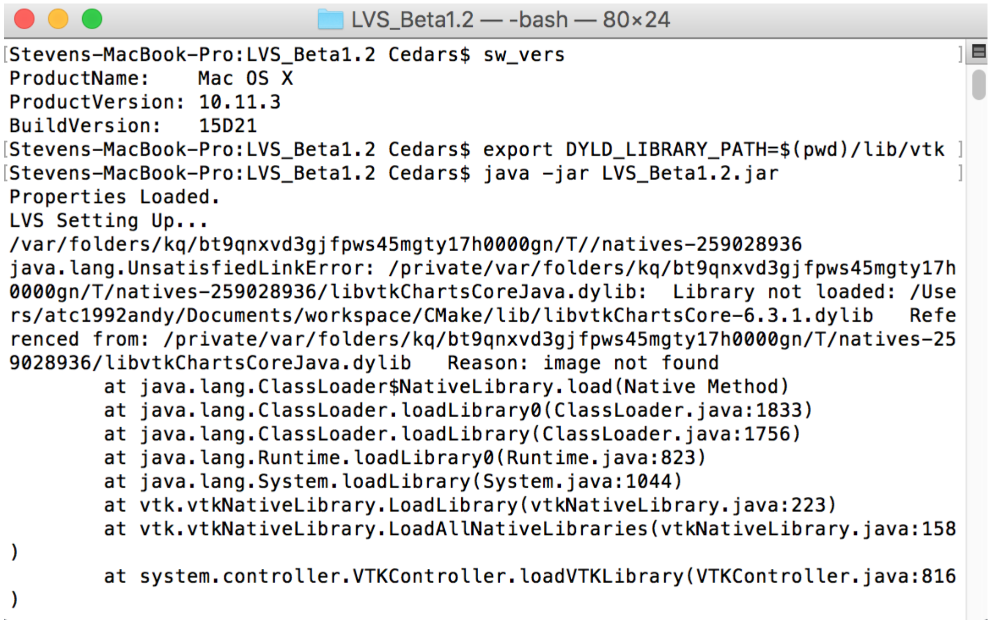
7.3.2 Distribution on Mac

In Java, a software can be distributed by extracting the source code into a Jar file. The Jar file contains all the implemented classes, other used Libraries (Jar File) and the project configurations. Since VTK and Leap Motion are not built on Java, additional native libraries and specific configurations are required. However, a recent update on OSX states that after version 10.11 El Capitan, due to a new security measure, all configuration on dynamic linker (dyld) environment variables is not possible unless System Integrity Protection (SIP) is disabled. These cause our system will only be able to be distributed on older OSX versions only. The following diagrams show the results by applying the same configuration under OSX version 10.9 and 10.11.



```
soushunkotowazas-MacBook-Pro:LVS_Beta1.2 atc1992andy$ sw_vers
ProductName:    Mac OS X
ProductVersion: 10.9.5
BuildVersion:   13F1134
soushunkotowazas-MacBook-Pro:LVS_Beta1.2 atc1992andy$ export DYLD_LIBRARY_PATH=$
(pwd)/lib/vtk
soushunkotowazas-MacBook-Pro:LVS_Beta1.2 atc1992andy$ java -jar LVS_Beta1.2.jar
Properties Loaded.
LVS Setting Up...
/var/folders/9l/k45d41ns2m3873_r2j3xphwm0000gn/T//natives-835974816
LVS Launched
```

Figure 32: LVS on OSX 10.9



```
Stevens-MacBook-Pro:LVS_Beta1.2 Cedars$ sw_vers
ProductName:    Mac OS X
ProductVersion: 10.11.3
BuildVersion:   15D21
Stevens-MacBook-Pro:LVS_Beta1.2 Cedars$ export DYLD_LIBRARY_PATH=$(pwd)/lib/vtk
Stevens-MacBook-Pro:LVS_Beta1.2 Cedars$ java -jar LVS_Beta1.2.jar
Properties Loaded.
LVS Setting Up...
/var/folders/kq/bt9qnxvd3gjfpws45mgty17h0000gn/T//natives-259028936
java.lang.UnsatisfiedLinkError: /private/var/folders/kq/bt9qnxvd3gjfpws45mgty17h
0000gn/T/natives-259028936/libvtkChartsCoreJava.dylib:  Library not loaded: /Use
rs/atc1992andy/Documents/workspace/CMake/lib/libvtkChartsCore-6.3.1.dylib   Refe
renced from: /private/var/folders/kq/bt9qnxvd3gjfpws45mgty17h0000gn/T/natives-25
9028936/libvtkChartsCoreJava.dylib   Reason: image not found
        at java.lang.ClassLoader$NativeLibrary.load(Native Method)
        at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1833)
        at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1756)
        at java.lang.Runtime.loadLibrary0(Runtime.java:823)
        at java.lang.System.loadLibrary(System.java:1044)
        at vtk.vtkNativeLibrary.LoadLibrary(vtkNativeLibrary.java:223)
        at vtk.vtkNativeLibrary.LoadAllNativeLibraries(vtkNativeLibrary.java:158
)
        at system.controller.VTKController.loadVTKLibrary(VTKController.java:816
)
```

Figure 33 : LVS on OSX 10.11

From Figure 32, we can see that under OSX 10.9, the system could be launched normally after pointing the system variable $DYLD_LIBRARY_PATH to the VTK native libraries. But in Figure 33, we discovered that the machine was unable to locate the native libraries under OSX 10.11, where the $DYLD_LIBRARY_PATH was not used due to the new security measure. Also, we noticed that the program will check the directory where the native libraries were generated. However, that path is unreachable for users because it was hard coded in the libraries where user will not be able to change it. Therefore, our system cannot be distributed on OSX 10.11 and onwards. This problem cannot be resolved unless Apple releases a new version that changes the setting or the users generate the native libraries and replace the vtk.jar by themselves. Resulting from these, we can only focus on the distributions on OSX versions before El Capitan.

7.3.3 Gesture Limitation & Management

As mentioned in section 5.2 we have used a very unique methodology to implement the gesture analysis. In section 6.4, we have proposed a simple test to measure the recognition rate briefly. The performance of the test was satisfying and the overall recognition rate could achieve over 80%. In fact, we have foreseen the limitations and problems of this approach. Although this approach helped us to implement the gesture interface in a short time, which only nine of them has been implemented. If the number of gestures increases, the performance of this approach may have significantly decreases. Meanwhile, when the number of gestures or the complexity of a specific gesture increased, it will be harder for us to setup the checking conditions. This means it may cost us much effort to implement those gestures than collecting a training set. Further work has to be done in order to analyse the performance of this approach comparing with other traditional approach.

Gesture management is another important issue we have to consider. Since there are two types of gestures, Static and Dynamic, where Dynamic Gesture is actually involving multiple Static Gesture recognition as to identify the start status and end status. Therefore, if a system is including both type of gestures, a good mechanism have to be used to manage the transition from one gesture to another. For instance, a Static Gesture holds an input control like the left key of the mouse. The next frame or the next gesture has to respond to this situation, like releasing the input control. If not, the system may perform some action that the user does not wish to be done. The current mechanism is recording the previous gesture type, if the next incoming gesture is not the same, relative control will have to be released. This mechanism is not proven or tested, therefore, we should still continue to seek for other valid options.

8 Discussion & Future Work

8.1 Programming Language for Implementation

Referring to the compatibility problems mentioned in section 7.3.1, Java might not be the best language to implement the system. If much time was given, we would re-create the project using C++. The reason of choosing C++ is, both VTK and Qt was written in C++, it will be more convenient for us to find out the errors and saving our time to solve them. In the meantime, we will also like to study other visualisation libraries to explore more functions that could be implemented.

8.2 Gesture vs Mouse

It is not easy to maintain an accurate gesture interface with high gesture variety. People might think that using a mouse can perform the same operation with higher accuracy. However in the perspective of HCI, Naturalness is an important aspect that we have to consider in modern software or system. Also, in some situations users might not prefer to use a mouse, for instance, a surgeon. Under these situations, a gesture interface provides a more natural interaction to users. Still, the accuracy and functionality of the current gesture interface requires more work to be completed. At the moment, it will be better to use the gesture interface to assist some operations like a touch-pad of a MacBook. Unless these deficiencies has been overcome, the gesture interface can only be acted as a supporting control instead of the only or main control.

8.3 Gesture Management

Following the previous sections, the gesture interface could be improved in many ways. One of them will be an idea of Gesture and Gesture command [21]. A gesture means a specific hand pose with significant feature, whilst gesture command can be treated as an action or a Dynamic Gesture. For instance, when a user performs a Pointing Gesture or a Two Fingers Gesture with a Click Command to accomplish two different operations. With this combination, the system can provide more variety of controls rather than analyse them linearly.

Another improvement that can be made is the mapping mechanism. The current mapping method was hard coding the operation in the gesture model. Although the developer may change the operation easily, users do not have the access right to the source code. To overcome this, a hash map can be used to map the gestures and the operations. This will also allow users to map the frequently used functions with the most comfortable gestures.

Lastly, as to improve the system accuracy, we may compare the approach that currently using with others. Another way will be combining them into a new approach. But all this proposed idea requires more time to be implemented and thoroughly tested.

9 Conclusion

In this project, we have demonstrated a robust visualisation system with a gesture interface. The overall performance of the system have been achieved as expected (See section 6 for details). In fact, there are several areas that we can improve (See section 7 & 8 for details). First, the gesture analysing approach currently using requires more effort to be tested and modified. Secondly, the functionality of the system can be enriched. Thirdly, throughout the development processes, we have experienced various compatibility problems. It indicated that Java might not be the best language to implement the project. Therefore, if we were given an opportunity to re-do the project, we would like choose C++ as the implementation language. These might speed up the development progress and result in enhancement on all aspects of the system, especially the GUI. Nevertheless, we were still able to complete the project and fulfilling most of the original specifications.

10 Bibliography

[1] Alex Colgan (9th August 2014) How Does the Leap Motion Controller Work?. Available at: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/. Last accessed 4th April 2016.

[2] ByungWoo Min, HoSub Yoon, Jung Soh, TakeShi Ohashi and Toshiaki Ejima. (1999). Visual Recognition of Static/Dynamic Gesture: Gesture-Driven Editing System. Academic Press.

[3] Feng Sheng Chen, Chih Ming Fu and Chung Lin Huang. (2003). Hand Gesture Recognition Using a Real-Time Tracking Method and Hidden Markov Models. Elsevier B.V.. Taiwan, ROC.

[4] Giulio Marin, Fabio Dominio and Pietro Zanuttigh. (2014). Hand Gesture Recognition with Leap Motion and Kinect Devices. Image Processing (ICIP). Paris.

[5] James M. Rehg and Takeo Kanade. (1994). Visual Tracking of High DOF Articulated Structures: An Application to Human Hand Tracking. Springer Berlin Heidelberg.

[6] Java SDK Documentation. Available at: https://developer.leapmotion.com/documentation/java/index.html. Last accessed 4th April 2016.

[7] Javier Varona, Antoni Jaume-i-Capò, Jordi Gonzàlez and Francisco J. Preales. (2008). Toward Natural Interaction through Visual Recognition of Body Gestures in Real-time. Elsevier B.V. Spain.

[8] Jintae Lee and Tosiyasu L. Kunii. (1993). Constraint-Based Hand Animation. 3, p110 - 127. Springer Japan. Tokyo.

[9] Juan Wachs, Helman Stern, Yael Edan, Michael Gillam, Craig Feied, Mark Smith and Jon Handler. (2007). Real-Time Hand Gesture Interface for Browsing Medical Images. Springer Berlin Heidelberg. Israel.

[10] Juan Wachs, Helman Stern, Yael Edan, Michael Gillam, Craig Feied, Mark Smith and Jon Handler. (2007). Gestix: A Doctor-Computer Sterile Gesture Interface for Dynamic Environments. Springer Berlin Heidelberg. Israel.

[11] K. K. Biswas and Saurav Kumar Basu. (2011). Gesture Recognition using Microsoft Kinect. Automation, Robotics and Applications (ICARA). Wellington.

[12] Kitware, Inc. Available at: http://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf. Last accessed 4th April 2016.

[13] Kitware, Inc. (2010). VTK User's Guide. Colombia - Latin America: Kitware, Inc.

[14] Maryam Khademi, Lucy Dodakian, Hossein Mousavi Hondori, Cristina V. Lopes, Alison McKenzie and Steven C. Cramer. (2014). Free-Hand Interaction with Leap Motion Controller for Stroke Rehabilitation. ACM. New York, USA.

[15] Matthew Turk and Mathias Kölsch. (2004). Emerging Topics in Computer Vision: Prentice Interface. 10, p455 - 519. Prentice Hall PTR Upper Saddle Rive. NJ, USA.

[16] Michal Roth and William T. Freeman. (1994). Orientation Histograms For Hand Gesture. Mitsubishi Electric Research Labs. Cambridge, UK.

[17] Moniruzzaman Bhuiyan and Rich Picking. (2009). Gesture-controlled user interfaces, what have we done and what's next?. Glyndŵr University. Wrexham.

[18] Particleincell (16th December 2011). Data Visualization with Java and VTK. Available at: https://www.particleincell.com/2011/vtk-java-visualization/. Last accessed 4th April 2016.

[19] Pragati Garg, Naveen Aggarwal and Sanjeev Sofat. (2009). Vision Based Hand Gesture Recognition. Springer Netherlands.

[20] Shahram Jalaliniya, Lars Büthe, Jeremiah Smith, Thomas Pederson and Miguel Sousa. (2013).Touch-less Interaction with Medical Images Using Hand & Foot Gestures. ACM. New York, USA.

[21] Thomas Baudel and Michel Beaudouin-Lafon. (1993). CHARADE: Remote Control of Objects using Free- Hand Gestures. ACM. New York, USA.

[22] Vladimir I. Pavlovic, Rajeev Sharma and Thomas S. Huang. (1997). Interpretation of Hand Gestures for Human-Computer Interaction a Review. 19, p677 - 695. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[23] VTK-Users. Available at: http://vtk.1045678.n5.nabble.com/VTK-Users-f1224199.html. Last accessed 4th April 2016.

[24] VTK Wiki. Available at: http://www.vtk.org/Wiki/VTK. Last accessed 4th April 2016.

[25] Will Schroeder, Ken Martin and Bill Lorensen. (2006). The Visualization ToolKit. USA. Person Education, Inc.

[26] Ying Yin and Randall Davis. (2010). Toward Natural Interaction in the Real World: Real-time Gesture Recognition. ACM. New York, USA.

[27] Yuxuan Huang, Zhongpan Qiu, Zhijun Song. (2011). 3D reconstruction and visualisation from 2D CT images. IT in Medicine and Education (ITME). Cuangzhou.

[28] Zhou Ren and Junsong Yuan. (2011). Robust Hand Gesture Recognition Based on Finger-Earth Mover's Distance with a Commodity Depth Camera. ACM. New York, USA.

[29] Zhou Ren, Jingjing Meng, Junsong Yuan. (2011). Robust Hand Gesture Recognition with Kinect Sensor. ACM. New York, USA.

11 Appendix

11.1 System Requirement

- Operation Platform : OSX
- Version : any version before 10.11 (El Capitan)
- Java Version : 1.6
- Device : Leap Motion Controller

11.2 Structure of Zip

1. Additional_Materials.pdf - All additional materials including Gantt Chart, Functional Test Cases, UML Diagrams and Manuals
2. LVS_Beta1.2 - Directory containing the binary version of LVS
3. LVS_Source - Directory containing the source version of LVS
4. SVN_Path - File containing the address of the project's SVN repository
5. TestingDataFiles - Directory containing some data model file that can be used to test the system.

11.3 Installation

In order to launch the software, users have to install java version 1.6. The Java SE Development Kit is aviable at: `https://support.apple.com/kb/dl1572?locale=en_US`

After the user has installed the Java version 1.6, the user can go to the **Binary Version** directory via terminal and run the **LVS.sh** to launch the system. Alternatively, if users have installed other version of Java 1.6, they can either change the shell-script or input the following commands to launch the application.

1. export JAVA_HOME=‘/usr/libexec/java_home -v 1.6.0_**YOUR_VERSION_NUMBER**‘
2. export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$(pwd)/lib/vtk
3. java -jar LVS_Beta1.2.jar