

# Networked Haptics

MARTIN OLOFSSON  
and  
SEBASTIAN ÖHMAN



**KTH Information and  
Communication Technology**

Master of Science Thesis  
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:106

# Networked Haptics

**Martin Olofsson & Sebastian Öhman**

[molofs@kth.se](mailto:molofs@kth.se)

[sohm@kth.se](mailto:sohm@kth.se)

2009-08-21

*Examiner*

**Prof. Gerald Q. Maguire Jr.**

*Supervisors*

**Prof. Lars Weidenhielm (Karolinska Institutet)**

**Prof. Marilyn E. Noz (NYU & Karolinska Institutet)**

**Prof. Gerald Q. Maguire Jr. (KTH)**

School of Information and Communication Technology  
Kungliga Tekniska Högsolan - Royal Institute of Technology (KTH)  
Stockholm, Sweden



# Abstract

Haptic feedback is feedback relating to the sense of touch. Current research suggests that the use of haptic feedback could give an increase in speed and accuracy when doing certain tasks such as outlining organ contours in medical applications or even filling in spreadsheets. This master thesis project has two different goals concerning haptic feedback. The first is to try to improve the forces for the SensAble PHANTOM Omni<sup>®</sup> Haptic Device when used in an application to outline contours in medical images, to give the user better feedback. The PHANTOM Omni is a device able to read in user movement of an arm attached to it in three dimensions, but it is also able to output forces through this arm back to the user, i.e. giving haptic feedback. By improving these forces and thus providing better feedback, we hope that speed and accuracy increases for a user working with the mentioned application.

The second part of the project consists of evaluating if delays in a network between the haptic feedback device and the place where the data sets are located impact the user perceived quality or the outcome of the task. We do this by considering a number of potential architectures for distributing the image processing and generation of haptic feedback. By considering both of these goals we hope to demonstrate both a way to get faster and more accurate results when doing the tasks already mentioned (and other tasks), but also to understand the limitations of haptic performance with regard to distributed processing.

We have successfully fulfilled our first goal by introducing a haptic force which seems quite promising. This should mean that the people working with outlining contours in medical images can work more effectively; which is good both economically for hospitals and quality of service-wise for patients.

Our results concerning the second goal indicate that a haptic system for outlining contour can work well when using this new haptic force, even on low quality data links (which can be used for example in battlefield medicine or by specialists to conduct long distance operations or examinations) -- *if* the system architecture distributes the functionality so as to provide low delay haptic feedback locally.

We have tried to compare our results from the second part with a model for the impact of network delay on voice traffic quality developed by Cole and Rosenbluth, but as there is not necessarily a numeric correspondence between the quality values that we used and the ITU MOS quality values for voice we cannot make a numeric comparison between our results and that model. However our experimental data seem to suggest that the decrease in perceived quality was not as fast as one might expect considering simply the ratios of the voice packet rate (typically 50 Hz) and the 1000 Hz rate of the haptic feedback loop. The decrease in quality seems to only be about one half of what the ratio of these rates might suggest (i.e., a factor of 10x faster decrease in quality with increasing delay rather than 20x).

# Sammanfattning

Haptisk återkoppling är återkoppling som fås genom känseln. Nutida forskning visar att användandet av sådan återkoppling kan öka effektiviteten vid vissa arbetsuppgifter inom sjukvård, till exempel vid förberedande uppgifter inom strålbehandling, men också vid kontorsarbete såsom att fylla i värden i ett kalkylark. Detta examensarbete har två mål som rör haptisk återkoppling. Det första är att försöka förbättra krafterna som ges från den haptiska enheten SensAble PHANTOM Omni<sup>®</sup> Haptic Device vid användning i ett medicinskt datorprogram rörande strålbehandling, i syfte att förbättra användareffektiviteten. PHANTOM Omni är en maskin som har en arm kopplad till sig som både kan läsa av rörelser och ge ut krafter med hjälp av en inbyggd motor, det vill säga ge haptisk återkoppling. Genom att förbättra dessa krafter och därigenom ge mer realistisk återkoppling hoppas vi att effektiviteten kan öka för en användare av det nämnda datorprogrammet.

Det andra målet är att utvärdera hur fördröjningar i ett nätverk mellan den plats där enheten är placerad och den plats där informationen som ska bearbetas finns, påverkar upplevelsen för användaren och därmed resultatet av arbetet. Vi genomför detta genom att analysera flera olika tänkbara arkitekturer, där placeringen av bilddatat och uträkningen av krafter som ska ges ut av den haptiska enheten varierar. Genom att undersöka dessa två olika aspekter hoppas vi att vi både kan visa ett sätt att få snabbare och bättre resultat när man arbetar med uppgifter av den karaktären som redan beskrivits, men också att förstå begränsningarna för att använda haptisk återkoppling i distribuerade system.

Vi har framgångsrikt lyckats uppfylla vårt första mål genom att utveckla en kraft till den haptiska enheten som verkar lovande. Om denna kraft funkar i praktiken innebär det att personer som arbetar med förberedande uppgifter inom strålbehandling kan göra dessa uppgifter effektivare vilket är positivt både ekonomiskt för sjukhusen och kvalitetsmässigt för patienterna.

De resultat vi har fått fram avseende vårt andra mål indikerar att användandet av en haptisk enhet inom medicinsk bildbehandling kan fungera bra med vår nyutvecklade kraft, även på nätverkslänkar med dålig kvalitet (som kan vara fallet exempelvis när medicinska specialister utför undersökningar eller operationer på distans) – *om* systemet är uppbyggt så att den haptiska återkopplingen sker lokalt med en minimal fördröjning.

Vi har försökt att jämföra våra resultat från nätverksdelen med en modell beskriven av Cole och Rosenbluth, som ger kvaliteten på rösttrafik som en funktion av fördröjningen i ett nätverk. Dock finns det inte nödvändigtvis någon korrelation mellan de värden vi har fått fram och den kvalitetsskala för rösttrafik som de använde. Vi kan därmed inte göra en jämförelse rakt av mellan våra resultat och deras modell. Däremot så pekar de data vi har fått i våra experiment på att den användarupplevda kvaliteten inte sänktes lika snabbt som man kunde ha väntat sig om man bara tar hänsyn till förhållandet mellan uppdateringsfrekvenserna för rösttrafik (vanligtvis 50 Hz) och den haptiska återkopplingen (1000 Hz). Kvalitetssänkningen verkar vara hälften av vad detta förhållande skulle kunna antyda (det vill säga en faktor på 10 gånger snabbare sänkning i kvalitet med ökande fördröjning snarare än 20 gånger).

# Acknowledgements

We would like to thank all the three professors that have been involved in this master thesis project, ultimately making it possible: Professor Lars Weidenhielm at the Karolinska Institute for letting us use his office and equipment during our entire project, Professor Marilyn E. Noz at the New York University and Karolinska Institute for great help and guidance throughout the whole project, and Professor Gerald Q. Maguire Jr. at the Royal Institute of Technology for providing swift feedback and answers whenever we had any questions and for giving us lots of valuable comments concerning our report.

We would also like to thank Stig Larsson and Bo Nordell at the Karolinska Institute for giving us tours around their departments at the Karolinska University Hospital in Solna, showing us their PET-CT and MRI units respectively.

We are very grateful for all of the volunteers that participated in our tests, helping us gather data for our analysis and giving us feedback about the testing system.

Last but not least, we would like to thank our friends and our respective families for their continuous support and encouragement during this project.

# Table of contents

List of Figures .....	vii
List of Tables.....	ix
List of Examples.....	x
List of Acronyms and Abbreviations .....	xi
Chapter 1 - Introduction .....	1
Chapter 2 - Background .....	2
2.1 Haptics.....	2
2.2 SensAble PHANTOM Omni Haptic Device.....	2
2.2.1 Writing applications for the SensAble PHANTOM Omni Haptic Device .....	4
2.3 Two-dimensional haptic devices .....	5
2.4 OpenDX.....	6
2.5 UDP.....	6
2.6 Traffic Control – tc.....	7
2.6.1 Traffic control: delay setup .....	7
2.6.2 Traffic control: packet loss setup .....	9
2.6.3 Traffic control: reset.....	10
2.7 Difference in coordinate systems .....	10
2.8 Summary of background .....	14
Chapter 3 - Related work .....	15
3.1 Design considerations for stand-alone Haptic Interfaces communicating via the UDP Protocol .....	15
3.2 Haptic Feedback for Medical Imaging and Treatment Planning.....	15
3.3 Voice over IP Performance Monitoring .....	15
3.4 Nyudemo .....	16
3.5 Summary of the related work .....	22
Chapter 4 - Method .....	23
4.1 Goals.....	23
4.1.1 Forces .....	23
4.1.2 Networked haptics.....	23
4.2 Plan for this thesis project .....	23
4.2.1 The existing system.....	24
4.2.2 Forces .....	24
4.2.3 Networked haptics.....	25
Chapter 5 - Forces .....	26
5.1 Introduction to forces.....	26
5.1.1 What is a force?.....	26
5.1.2 How are we going to use forces .....	26
5.1.3 Different potential forces .....	26
5.2 Possible implementable forces .....	27
5.2.1 Bump force.....	27
5.2.2 Magnetic force.....	28
5.2.3 Spring force .....	29
5.2.4 Wall force.....	30
5.2.5 Viscous force.....	30

5.3	Spring forces.....	31
5.3.1	Anchored Spring Forces.....	31
5.3.2	FrictionlessPlane .....	33
5.4	Viscous forces.....	34
5.4.1	Viscosity.....	34
5.4.2	Bump force.....	35
5.5	Implementing a spring force in the OpenDX HapticsModule.....	36
5.5.1	Overview .....	36
5.5.2	Movement.....	37
5.6	Stage one: Implement a test application .....	39
5.6.1	Overview .....	39
5.6.2	Implementation.....	39
5.6.3	The spring force .....	40
5.6.4	Callback loop.....	40
5.6.5	Evaluation of the spring force test application.....	40
5.7	Stage two: Adding the spring force to the HapticsModule in two dimensions .....	40
5.7.1	Evaluation of the spring force OpenDX HapticsModule implementation.....	41
5.8	Stage three: Implement in a OpenDX network in three dimensions .....	41
5.9	Implementing a magnetic force .....	41
5.9.1	Overview .....	41
5.9.2	Test application .....	41
5.9.3	Evaluation of the magnetic test-application .....	43
5.9.4	Use the magnetic force to make tracing of a line easier.....	44
5.9.5	Implement the magnetic force into the OpenDX network .....	44
5.9.6	Evaluation of the magnetic force OpenDX HapticsModule implementation ....	44
Chapter 6 -	Networked haptics.....	45
6.1	Networked haptics introduction.....	45
6.1.1	Networked haptics: What will we do? .....	45
6.1.2	Networked haptics: Initial configuration.....	45
6.1.3	Networked haptics: How to split the code .....	45
6.2	Proposed method 1 .....	46
6.3	Proposed method 2 .....	47
6.4	Proposed method 3 .....	48
6.5	Obstacles with these methods.....	49
6.5.1	Idea A .....	49
6.5.2	Idea B .....	49
6.5.3	Idea C .....	49
6.5.4	A fourth approach.....	50
6.5.5	Caching and problems due to caching.....	50
6.5.6	A different approach.....	51
6.5.7	How do we solve the problem of rapid movement of the locator .....	51
6.5.8	Conclusion.....	52
6.6	Network tests: Description of the test setup .....	52
6.7	Network tests: How was the testing done? .....	52
6.8	Network tests: Evaluation.....	53



Chapter 7 - Analysis.....	55
7.1 Forces.....	55
7.1.1 Anchor points in different images.....	55
7.1.2 Summary of force analysis.....	61
7.2 Networked haptics.....	62
7.2.1 Delay and packet loss testing.....	62
7.2.2 Test results.....	63
7.2.3 Network caching.....	70
7.2.4 Network analysis.....	77
Chapter 8 - Conclusions and Future Work.....	78
8.1 Conclusions.....	78
8.2 Suggestions for others working in this area.....	78
8.2.1 Examine how much computation actually needs to be done each time the haptic loop runs.....	78
8.2.2 Usage of another haptic device.....	78
8.2.3 If we had to do it again, what would we have done differently?.....	78
8.3 Future work.....	79
8.3.1 Better ways to calculate anchor points.....	79
8.3.2 Improve movement along lines.....	80
8.3.3 Cached network haptics.....	80
8.3.4 Experimental measurements with very low added delays.....	80
8.4 Work that we have not completed.....	80
8.4.1 Introduce a way to disable the forces.....	80
8.4.2 Extending the work and experiments to 3D.....	80
8.4.3 Add a viscous force and wall force.....	80
8.4.4 Completely split the code for the network part.....	81
8.5 The most obvious next steps.....	81
8.6 Hints for someone who is going to follow up this work.....	81
References.....	82
Appendix A: Medical images with anchor points.....	84

## List of Figures

Figure 1: Photo of the PHANTOM Omni Haptic Device .....	3
Figure 2: PHANTOM Omni with illustrated force components and possible movement of arms .....	4
Figure 3: The white arrow shows the breakpoint where the response time starts to differ after the command in the lower part of the figure was issued.....	9
Figure 4: Results of the <i>ping</i> command when tc was configured to generate a 50% packet loss .....	10
Figure 5: Coordinate system used by the force field array .....	11
Figure 6: Mapping of the pixel gradient values into the force_field array .....	11
Figure 7: Coordinate system used by OpenDX.....	12
Figure 8: An extremely zoomed in image of a hip prosthesis.....	13
Figure 9: Coordinate outputs in OpenDX .....	13
Figure 10: PHANTOM Test – Test and calibration program for the haptic device.....	13
Figure 11: Relation between delay and perceived quality for voice over IP traffic (based upon the equations shown in [23]) .....	16
Figure 12: Nyudemo - HapticOpen page .....	17
Figure 13: Nyudemo - Import page.....	18
Figure 14: Nyudemo - Slices page .....	19
Figure 15: Nyudemo - Display page .....	20
Figure 16: Nyudemo - findslicemax page .....	20
Figure 17: First screenshot while testing Nyudemo.....	21
Figure 18: Second screenshot while testing Nyudemo .....	22
Figure 19: OpenDX network running with two partially completed regions of interests.....	24
Figure 20: Bump force .....	28
Figure 21: Magnetic force shown for two different shapes .....	28
Figure 22: Spring force .....	29
Figure 23: Wall force .....	30
Figure 24: Viscous force .....	31
Figure 25: Right-handed three dimensional Cartesian Coordinate system[25] .....	32
Figure 26: OpenDX configuration panel.....	37
Figure 27: Algorithm for implementing spring forces in the HapticsModule .....	1
Figure 28: The "frame" test shape.....	39
Figure 29: Magnetic force scan.....	42
Figure 30: A scenario with the haptic device in close proximity to the computer computing and displaying the image data .....	46
Figure 31: Data flow for proposed method 1 .....	46
Figure 32: A scenario with a local haptic device and image display, but with remote computation of forces.....	47
Figure 33: Data flow for proposed method 2 .....	47
Figure 34: Updated data flow for proposed method 2 .....	48
Figure 35: Configuration for testing delays .....	48
Figure 36: Using a remote loopback of force messages for testing .....	49
Figure 37: Possible problem with stale cached state.....	50
Figure 38: Hip phantom with a threshold value of 0.3.....	57

Figure 39: Hip phantom with a threshold value of 0.5.....	58
Figure 40: Hip phantom with a threshold value of 0.5.....	59
Figure 41: Zoomed in hip phantom with a threshold value of 0.3.....	60
Figure 42: Zoomed in hip phantom with a threshold value of 0.5.....	61
Figure 43: Anchor version – User 1 – Delay .....	64
Figure 44: Anchor version – All users comparison – Delay .....	64
Figure 45: Force version – User 1 – Delay .....	65
Figure 46: Force version – All users comparison – Delay.....	65
Figure 47: Force version – User 1 – Packet loss .....	66
Figure 48: Force version – Comparison – Packet loss .....	66
Figure 49: Anchor version – User 1 – Packet loss .....	67
Figure 50: Anchor version – Comparison – Packet loss .....	67
Figure 51: Force VS anchor – Delay.....	68
Figure 52: Force VS anchor without user 3 – Delay .....	68
Figure 53: Force VS anchor – Packet loss .....	69
Figure 54: Force VS anchor without user 3 – Packet loss .....	69
Figure 55: Box-plot of the four images with 0.3 gradient threshold.....	71
Figure 56: Box-plot of the four images with 0.5 gradient threshold.....	72
Figure 57: Box-plot of three images with 0.5 gradient threshold .....	72
Figure 58: Probability of a cache hit with one slice after 1 second .....	74
Figure 59: Probability of a cache hit with one slice after 15 seconds.....	75
Figure 60: Probability of a cache hit with 100 slices after 1 second.....	75
Figure 61: Probability of a cache hit with 100 slices after 15 seconds .....	76
Figure 62: Probability of a cache hit with 100 slices after 30 seconds .....	76
Figure 63: Probability of a cache hit with 100 slices after 60 seconds .....	77
Figure 64: Abdomen with a threshold value of 0.3.....	84
Figure 65: Zoomed in abdomen with a threshold value of 0.3 .....	85
Figure 66: Abdomen with a threshold value of 0.5.....	86
Figure 67: Zoomed in abdomen with a threshold value of 0.5 .....	87
Figure 68: Hip patient with a threshold value of 0.3.....	88
Figure 69: Zoomed in hip patient with a threshold value of 0.3 .....	89
Figure 70: Hip patient with a threshold value of 0.5.....	90
Figure 71: Zoomed in hip patient with a threshold value of 0.5 .....	91

# List of Tables

Table 1: Maximum and usable workspace of the Omni device ..... 14  
Table 2: Description of images ..... 1  
Table 3: Number of anchor points in different images and with different thresholds ..... 70  
Table 4: kB of data for 1 slice ..... 73  
Table 5: MB of data for 100 slices ..... 73

## List of Examples

Example 1: Haptic device initialization call .....	4
Example 2: Force output enabling call.....	5
Example 3: Servo control loop scheduler starting call.....	5
Example 4: Code added to the AnchoredSpringForce program .....	32
Example 5: Example output from the modified AnchoredSpringForce program.....	32
Example 6: Direction flag in FrictionlessPlane program .....	33
Example 7: Check for plane penetration in the FrictionlessPlane program.....	33
Example 8: Force calculation in the FrictionlessPlane Program.....	33
Example 9: Code added to enable output of calculated force .....	33
Example 10: Check for pop-through and possibly apply force.....	33
Example 11: Example output from the modified FrictionlessPlane program.....	34
Example 12: Callback function of the bump force implementation .....	35
Example 13: The startBump() function.....	36
Example 14: The bumpCallback() function .....	36
Example 15: Magnetic force algorithm.....	1

## List of Acronyms and Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
CT	Computed Tomography
FIFO	First In First Out
HDAPI	Haptic Device Application Programming Interface
HLAPI	Haptic Library Application Programming Interface
IP	Internet Protocol
NIC	Network Interface Card
NETEM	Network Emulation
OpenDX	Open Data Explorer
QDISC	Queuing Discipline
RAM	Random Access Memory
ROI	Region of Interest
RTP	Real-time Protocol
TC	Traffic Control
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol



# Chapter 1 - Introduction

In medical science, an interesting field of current research is the use of haptic feedback in different applications as a tool for increasing productivity. The concept of haptic feedback, also known as haptics, means that the user gets feedback through the sense of touch. Haptics has been extensively used in medical applications for education, particularly in surgical simulation -- so that a user can practice surgery in order to improve their surgical technique. Haptics is also being studied for use within medical applications as a mean to speed up certain tasks, for example outlining organ contours on CT-images for use in radiation treatment. Research suggests that the use of haptic feedback in addition to regular visual feedback could give an increase in speed and accuracy when doing such medical tasks, as described in the masters thesis by Eva Anderlind[1]. Furthermore, other tasks of a non-medical nature may also take advantage of these types of feedback systems, for example filling in data in spreadsheets\*, training in flight simulators or to enhance the user experience in gaming.

This master thesis project builds upon Anderlind's work by examining how delays between the haptic feedback device and where the data sets are located, impacts the quality of the outcome of the task. The reason for examining such delays is because of the interesting possibility to use haptics in medical tasks over longer distances, for example letting surgeons perform tasks from a safe place while the patient is located in a battlefield hospital, when using distributed computing with the user interface implemented on one computer and computation taking place on one or more other computers, or for collaborative haptics. Additionally, we also wanted to understand if the *type of force* which is used changes the effect of delay on the user's performance.

In order to experimentally evaluate the effects of delay on haptic feedback during one or more tasks, we needed an implementation or simulation in order to be able to do experiments with changes in the delay. One such system was already implemented by Professor Marilyn E. Noz in OpenDX and was described in Eva Anderlind's thesis[1]. This application gives haptic feedback through gradient based forces multiplied by an exponential function and also through a viscous resisting force when moving the pointer. However, there are other kinds of forces that could be used as haptic feedback forces. At present it is not clear what kind of forces are the best suited for giving haptic feedback, therefore we will try to evaluate several different types of forces in order to learn what forces are likely to be most useful for haptic feedback purposes. As an example of another type of force, Karljohan Lundin Palmerius modeled surface forces in his dissertation *Direct Volume Haptics for Visualization* as a gradient force with an added viscous force[3]. In addition, he added a friction force that resists movement unless a large enough force is applied, hoping to achieve a force that is as natural as possible[3]. When implementing and testing our own forces we kept his results in mind.

Another issue is that a force that produces good feedback when used in a low delay system might be a bad choice when the delay increases. Therefore we implemented several different types of forces and tested them both with and without delay, then we evaluated what type(s) of force(s) were best in different scenarios.

---

\* The desire for using haptic feedback in filling in spreadsheets actually dates back as early as 1992, see the section "The Failure of Force Feedback" on page 25 of van Mensvoort's dissertation *What You See is What You Feel: On the simulation of touch in graphical user interfaces.*[2]



# Chapter 2 - Background

## 2.1 Haptics

Haptics is a word originating from the Greek word *hapthestai* meaning “to contact” or “to touch”[1]. Haptics is generally used as a description for the science of using tactile and force feedback in computer applications. However, it can more correctly be described as referring to the sense of touch regardless of context[3].

By using a haptic device for output in addition to the usual visual output via a monitor/display, there are indications that one can decrease the time used and increase the accuracy of performing some tasks. This use of haptic force feedback has been evaluated in several studies with positive results. The general impression is that the feeling of realism is increased and the interaction in turn is made more precise. Eva Anderlind explains this by saying that people tend to trust what they feel more than what they see, but at the same time states that any discrepancies between the haptic and the visual feedback can seriously decrease performance[1]. We conclude from this that an important aspect is to make sure that the scenarios that are going to be used in our experiments have consistent visual and haptic feedback.

The haptic device we used (a SensAble PHANTOM Omni<sup>®</sup> - for details see section 2.2) is using a point interaction system, where you control the device through an arm similar to a pen with a probe at the tip[3]. The probe is the point that allows the user to interact with the computer, but this device can only locate one point and give feedback for this point – one point at a time, whereas in the real world we can feel objects with our whole hand containing many different receptors (or probes) at the same time. This is a limitation of the haptic feedback interface that we are using, but studies show that people can be given sufficient feedback through point interaction that they can easily explore and manipulate a world with it[1].

A good example of how well haptics with point interaction works is an experiment by Ridel and Burton[4]. The goal of their experiment was to evaluate how accurately people could process different gradients of line graphs on either paper or via the haptic system while blindfolded. Their virtual haptic system consisted of a Logitech WingMan Force Feedback Mouse[11] (also mentioned in section 2.3 on page 5); while the analog system was raised lines on paper. The experiment showed that the test subjects were very accurate at the task. The results were also very similar between both the virtual and physical media, even though the physical paper would seem to have offered more data to the users. Their conclusion was that a haptic mouse allowed very accurate readings of simple line graphs.

## 2.2 SensAble PHANTOM Omni Haptic Device

The SensAble Technologies PHANTOM Omni<sup>®</sup> Haptic Device is a haptic feedback device with six degree-of-freedom positional sensing[5]. This device, shown in Figure 1, allows the user to input coordinates in three dimensions, as well as getting force feedback. The user can not only move the cursor within an X and Y plane, but also in Z which allows the user to navigate in 3D. The device will be used in our experiments as it has already been used by physicians for some medical tasks and there is a desire to improve the sense of

feedback that this device can offer users working with large three dimensional data sets. Via experiments we evaluate whether using this device yields a notable increase in accuracy and/or speed of certain medical tasks.



**Figure 1: Photo of the PHANTOM Omni Haptic Device**

Figure 2 shows the same photo of the haptic device with added illustrations to explain the usage of the device. The user holds the part of the haptic device that looks like a pen and the user is able to move it in three dimensions. The interaction is in a single point, in contrast to for example the human hand which has the ability to feel shapes via its large amount of receptors. The tip of the pen, marked with a green circle in Figure 2, is the interaction point and on a monitor this point is the equivalent to the cursor for an ordinary mouse.

The pen can be moved in x-, y-, and z-direction, marked by the blue arrows, and the device also outputs force feedback as a vector of those directions. In order to register this movement and to output forces the arms can be moved and the sphere rotated as showed by the red arrows. The pen also has two buttons for user input.



Figure 2: PHANTOM Omni with illustrated force components and possible movement of arms

### 2.2.1 Writing applications for the SensAble PHANTOM Omni Haptic Device

There are two different application programming interfaces (APIs) available for use with the Omni Haptic Device: HDAPI and HLAPI. HDAPI provides low-level access to the haptic device and offers the programmer direct control over how to render forces. In contrast, HLAPI provides high-level haptic rendering. HLAPI is designed to be used in applications that synchronize haptics and graphics threads and is designed to be familiar to OpenGL programmers. When testing the device we almost exclusively used the HDAPI to get a feel for how the device works without spending too much time writing graphics code. Next we will explain how the code is structured to make the use of the Omni Haptic Device possible in a program using HDAPI.

The first calls in a HDAPI application are for device initialization, the first of these is an explicit call to initialize the haptic device that is to be used. This is shown in Example 1.

```
1. HHD hHD = hdInitDevice(HD_DEFAULT_DEVICE);
```

Example 1: Haptic device initialization call

`HD_DEFAULT_DEVICE` is the default haptic device. We have only been working with a single haptic device attached to our computer, but if multiple haptic devices are used then the code must initialize all of the devices by calling this function with different names as arguments. The next step is to enable force output from the device; since all forces are turned off at initialization for safety reasons. This enabling of forces can be seen in Example 2.

```
1. hdEnable(HD_FORCE_OUTPUT);
```

**Example 2: Force output enabling call**

At this point there are still not any forces being sent to the device – forces will only be applied after the scheduler is started. The line above merely enables the possibility of force output from the device. To start the scheduler that runs the feedback servo control loop, i.e. the control loop that calculates the forces to send to the haptic device, the programmer makes a call as in Example 3.

```
1. hdStartScheduler();
```

**Example 3: Servo control loop scheduler starting call**

Haptic feedback differs from visual feedback in that while a 30 Hz refresh rate is sufficient for the eyes to **not** perceive any discontinuities in an animation, a 1000 Hz refresh rate is needed for a user not to perceive force discontinuities or force fidelity losses. Thus to render a stable haptic feedback this servo control loop has to be executed at 1000 Hz, therefore the scheduler call creates a new high priority thread that runs at that rate.

There are two types of scheduler calls: asynchronous and synchronous. The difference between these is that the synchronous call only returns when it is complete, so the application thread waits for this call to return before continuing, whereas an asynchronous call returns immediately after being scheduled. Synchronous calls are best used for getting the state of the scheduler, e.g. position or button state queries or for variable modifications during runtime, such as increasing the stiffness of a spring force. Asynchronous calls on the other hand are the most common choice for managing the haptic loop that outputs forces to the device (to be perceived by the user). If the asynchronous call returns `HD_CALLBACK_CONTINUE` the called function will continue to run until it eventually returns `HD_CALLBACK_DONE`. The call can terminate because of an error or because the program is terminated. Asynchronous calls managing the haptic loop should usually be called *before* the scheduler is started so that they begin executing as soon as the scheduler is started.

To define a section of code where the state of the device is guaranteed to be consistent, the programmer can create haptic frames similar to frames used for visual feedback. The syntax for creating a haptic frame is `hdBeginFrame()` and to define the end of the frame's scope `hdEndFrame()` is used. When a new frame is created the device state is updated and saved for use within that frame so that all state queries in the frame give consistent return values. At the end of the frame all state changes are pushed back to the device, e.g. force changes. Most of the operations in haptic programs should be framed to ensure data consistency in the program. According to the SensAble Technologies' *Sensable Open Haptics Toolkit Programmer's Guide*[6], it is recommended that the scheduler run one haptic frame per tick (when the scheduler is called) per device, but there is a possibility to override this.

## 2.3 Two-dimensional haptic devices

In addition to using the three dimensional SensAble Technologies PHANTOM Omni device we also considered using a two-dimensional mouse with haptic feedback capability as

it is probably one of the simplest ways of setting up a force feedback interface. Due to its simplicity, one can model forces in an easy way and conduct experiments that are easy to analyze and get results from. This could have been a useful way to recognize what kind of forces are the best for supplying haptic feedback and in turn speed up the task at hand. The mouse that was available to us was a Belkin Nostromo n30[7]. However, due to the fact that we started working with the PHANTOM Omni before getting access to this device and we had already got a good overview of working with a three dimensional device, we decided to continue working only with the PHANTOM Omni.

However, for a user new to working with haptic devices using a two dimensional device may be a good way to start. There are several applications that can generate simple haptic feedback that work with the Belkin Nostromo n30, for example Immersion's TouchWare Desktop[8] and iFeelPixel[9]. There are also other devices that work with the mentioned haptic feedback software, for example Logitech's iFeel Mouse[10], Logitech's WingMan Force Feedback Mouse[11], Kensington's Orbit 3D Trackball[12][13], Saitek's W07 Touch Force Gaming Mouse[14], and HP's Force Feedback Web Mouse[15].

## **2.4 OpenDX**

OpenDX[16] is an open source visualization software package based on IBM's Visualization Data Explorer. OpenDX uses an object-oriented data model and moreover handles all data input in a uniform way, regardless of what the source is. The package provides hundreds of different functions grouped into powerful modules, but it also allows the user to create or import custom made modules. It also features a graphical user interface where a programmer/user can create a visual program based upon placing the modules and functions that you want to invoke on a "programming canvas", and connecting the different parts with "wires" thus implementing a graphical data flow program. Using OpenDX we can easily create a program in an intuitive and visually well structured way, and with this program we can import haptic feedback modules allowing us to quickly and easily conduct experiments with different kind of forces.

There is another reason for using OpenDX and that is because we can use a visual network coded and supplied by Professors Marilyn E. Noz & Gerald Q. Maguire Jr. This visual network has implemented methods and forces used in medical image processing and was used earlier in Eva Anderlind's masters thesis project that examined how haptic feedback could speed up and make radiation treatment planning more accurate[1]. Professors Noz and Maguire implemented a haptics module that allows an OpenDX programmer to easily pass an array of three dimensional forces to be used by the haptic feedback loop. This code also sends the coordinates of the probe and the state of the buttons on the probe as UDP datagrams. A further advantage of using OpenDX is that we can leverage all of the existing work that has been done to read in medical images and manipulate them, without having to do very much of this work ourselves. This enables us to concentrate on our two project goals (see section 4.1 on page 23).

## **2.5 UDP**

The User Datagram Protocol (UDP)[17] is a minimal transport protocol that is part of the TCP/IP communication protocol stack. UDP is widely used in the Internet to carry real-time traffic (for example carrying real-time multimedia using the real-time protocol (RTP) which

in turn uses UDP as its transport protocol). Transmissions between devices utilizing UDP occur *without* session establishment. UDP does **not** provide any reliability, thus it is up to the programmer to provide their own timeouts, retransmissions, and acknowledgements if they want reliable data delivery, and as a result UDP has minimal overhead. The main alternative to UDP is TCP (Transmission Control Protocol), but that protocol requires that data be delivered in byte serial order. Otherwise, any loss of data forces all the data behind it to be buffered until the missing data is delivered. This can lead to very high variance in the time to deliver data between two applications. Hence TCP is unsuitable for our application<sup>†</sup>.

Since our haptic device is a real-time based system, retransmission of packets which arrive late with old data are of no use. Therefore UDP seems to be the correct choice of protocol; hence we will use UDP during our tests with networked haptics.

## 2.6 Traffic Control – tc

Traffic Control, (“tc”) is a UNIX tool that can be used to shape network traffic. Tc[18] is part of the Linux package “iproute2”[19]. Using tc, one can model a link with a certain amount of packet loss or a certain latency, by using actual traffic shaped by tc to simulate different network links between two computers. In this project we will use different values for packet loss and latency to try to determine how the perceived quality of haptic interaction varies as a function of the delay. The sub sections below describe the tests we made to learn how to configure and use traffic control.

### 2.6.1 Traffic control: delay setup

The following command can be used to add additional delay to traffic going through the target interface[20]:

```
tc qdisc add dev <device> root handle 1:0 netem delay <x>msec
```

Here **tc** is the program name. This command adds the specified queuing behavior to the indicated interface. The queuing discipline (Qdisc) is a *scheduling* mechanism. This scheduling mechanism determines how packets are handled; specifically it allows you to specify the flow of packets through a queue. The default scheduler (pfifo\_fast) is a First In First Out (FIFO) scheduler.

There are two different places tc can manipulate traffic: outgoing traffic (egress also known as `root`) and incoming traffic (ingress). In our experiments we used `root` to affect all the outgoing traffic through the specified interface (<device>).

The target device for the intended manipulation is specified as “dev <device>”. In Linux systems the Ethernet network interfaces are (typically) assigned a name of the form `ethX` where `X` is an index to the specific interface. Numbering usually starts at 0, e.g. if there are several Ethernet interfaces they would be named: `eth0`, `eth1`, `eth2`, ... .

The handle consists of the pair major:minor. The major is the name and identifier of the handler. This can be used to access the handler once created. If you have several classes under the same queuing discipline they are assigned different minor numbers.

---

<sup>†</sup> There exist other transmission protocols, such as the Stream Transmission Control Protocol, that could be used, but UDP is sufficient for our purposes.

Netemulation (Netem) is a part of the Linux kernel used to emulate packet loss, delay, duplication, and re-ordering of packets[21]. In Linux kernel 2.6 or later this kernel module should be included by default. You can include this module during kernel configuration through the following path:

**Networking --> Networking Options -->  
QoS and/or fair queuing --> Network emulator**

The delay parameter *delay <X> msec*; indicates that the traffic should be delayed by *X* milliseconds. As we will describe below this actually specifies a delay of at least *X* ms and does not guarantee a delay of exactly *X* ms.

The result of executing a `tc` command with a delay of 500 ms (shown in the lower window in Figure 3) can be seen in the upper window of Figure 3. The top window of the figure shows the output of a ping of the server as shown on our workstation and the bottom window shows the command being given on the server. These two computers are connected to the same network and the command *ping* is being executed on the workstation in order to send packets between the two computers. As soon as the `tc` command to add 500 ms of delay is executed on the server, we see that the delay increases accordingly. The change in the delay is highlighted by the arrow in the top window. As you can see, the delay is not exactly 500 ms, but a little more. The reason for this is that the `tc` queuing process adds 500 ms of delay, but does not take into account any processing delay. So when the process has been suspended for 500 ms there may be some additional time spent waiting for the operating system's process scheduler to give an execution time slice to `tc` – thus enabling the delayed packet to be sent.

```

C:\> Command Prompt - ping -n 9000 192.168.0.12
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time=504ms TTL=64
Reply from 192.168.0.12: bytes=32 time=505ms TTL=64
Reply from 192.168.0.12: bytes=32 time=501ms TTL=64
Reply from 192.168.0.12: bytes=32 time=502ms TTL=64
Reply from 192.168.0.12: bytes=32 time=506ms TTL=64
Reply from 192.168.0.12: bytes=32 time=505ms TTL=64
Reply from 192.168.0.12: bytes=32 time=501ms TTL=64
Reply from 192.168.0.12: bytes=32 time=506ms TTL=64
Reply from 192.168.0.12: bytes=32 time=506ms TTL=64
Reply from 192.168.0.12: bytes=32 time=503ms TTL=64
Reply from 192.168.0.12: bytes=32 time=504ms TTL=64
Reply from 192.168.0.12: bytes=32 time=504ms TTL=64
Reply from 192.168.0.12: bytes=32 time=501ms TTL=64
Reply from 192.168.0.12: bytes=32 time=502ms TTL=64

baaa:/home# ifconfig eth2
eth2      Link encap:Ethernet  HWaddr 00:1B:FC:74:2A:6E
          inet addr:192.168.0.12  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1099551 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2503936 errors:0 dropped:0 overruns:0 carrier:2
          collisions:0 txqueuelen:1000
          RX bytes:81637379 (77.8 MiB)  TX bytes:3737573954 (3.4 GiB)

baaa:/home# tc qdisc add dev eth2 root handle 1:0 netem delay 500msec
baaa:/home#

```

Figure 3: The white arrow shows the breakpoint where the response time starts to differ after the command in the lower part of the figure was issued.

### 2.6.2 Traffic control: packet loss setup

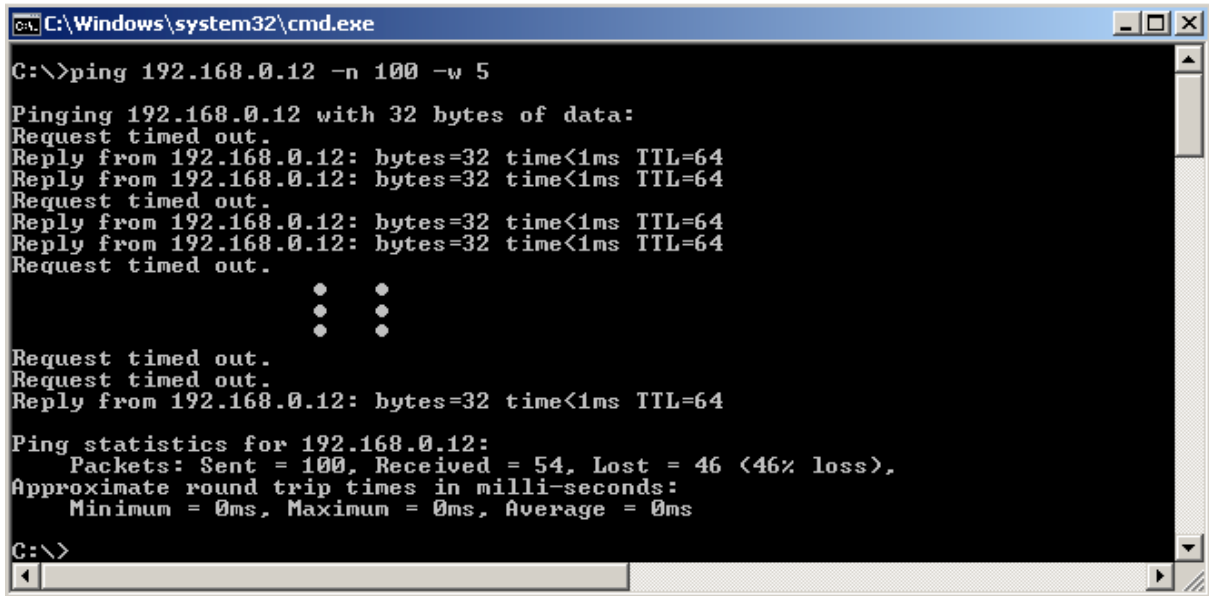
To emulate packet loss the following command can be used:

```
tc qdisc add dev <interface> root netem loss X%
```

The only difference from our previous delay example is that instead of specifying the *delay* parameter we use the parameter *loss* (which also is implemented by the package netem). Here *X* is the percentage of packets to be dropped. According to the tc manual page the smallest possible non-zero number is  $1/2^{32} = 0.0000000232\%$ , thus this is the smallest non-zero loss rate that can be specified.

To test the loss rate functionality we configured the system to inflict a packet loss of 50 percent on communication with our server. The command *ping* was then used to send 100 packets between our workstation and the server. The results can be seen in Figure 4. Due to the limited number of samples in our test the result is not exactly 50 percent lost packets, but it is close enough to convince us that the setup is working as intended.





```
C:\Windows\system32\cmd.exe
C:\>ping 192.168.0.12 -n 100 -w 5

Pinging 192.168.0.12 with 32 bytes of data:
Request timed out.
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Request timed out.
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64
Request timed out.
      . .
      . .
Request timed out.
Request timed out.
Reply from 192.168.0.12: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.0.12:
    Packets: Sent = 100, Received = 54, Lost = 46 (46% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

Figure 4: Results of the *ping* command when tc was configured to generate a 50% packet loss

### 2.6.3 Traffic control: reset

In order to reset the interface to its default (i.e., without any manipulation of the traffic flow) the following command can be issued:

```
tc qdisc del dev <device> root
```

## 2.7 Difference in coordinate systems

One problem we encountered when developing forces in the provided OpenDX program, was that different parts of the system used different coordinate systems. The force field array is a one dimensional array that consists of the gradient values of all the pixels in the dataset. That is, for each pixel in every slice there is an x-, y-, and z-value for the pixel's gradient. The array is indexed in such a way that the first slice's top-left pixel's gradient values occupy the first three elements of the array. Then the pixel just to the right of the first occupies the next three elements. The array is filled one row at a time until all the gradients from the first slice are stored, then it continues with the next slice, etc. The natural way of thinking for us was that the top-left pixel of the first slice had the coordinates (0, 0, 0), thus the positive x direction is to the right and the positive y direction is downwards as shown in Figure 5.

Since there are three values per pixel the total number of elements in the array is  $x*y*z*3$ . For example, in a volume with 5 slices  $10*10$  pixels each, there are  $10*10*5*3=1500$  array elements. The way that that volume's gradient values are put into the array is shown in Figure 6 with the front slice as slice number 1 in the volume.

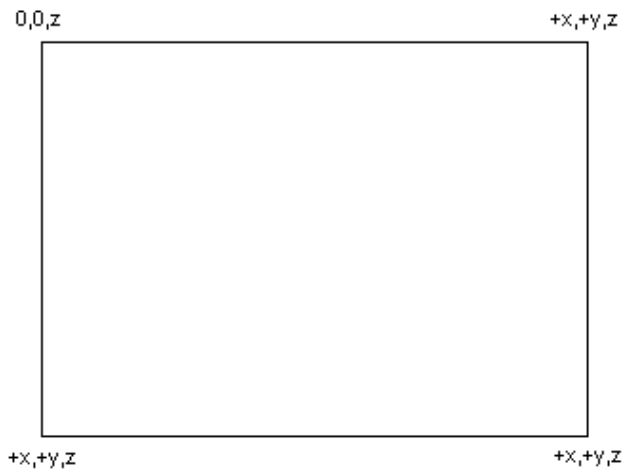


Figure 5: Coordinate system used by the force field array

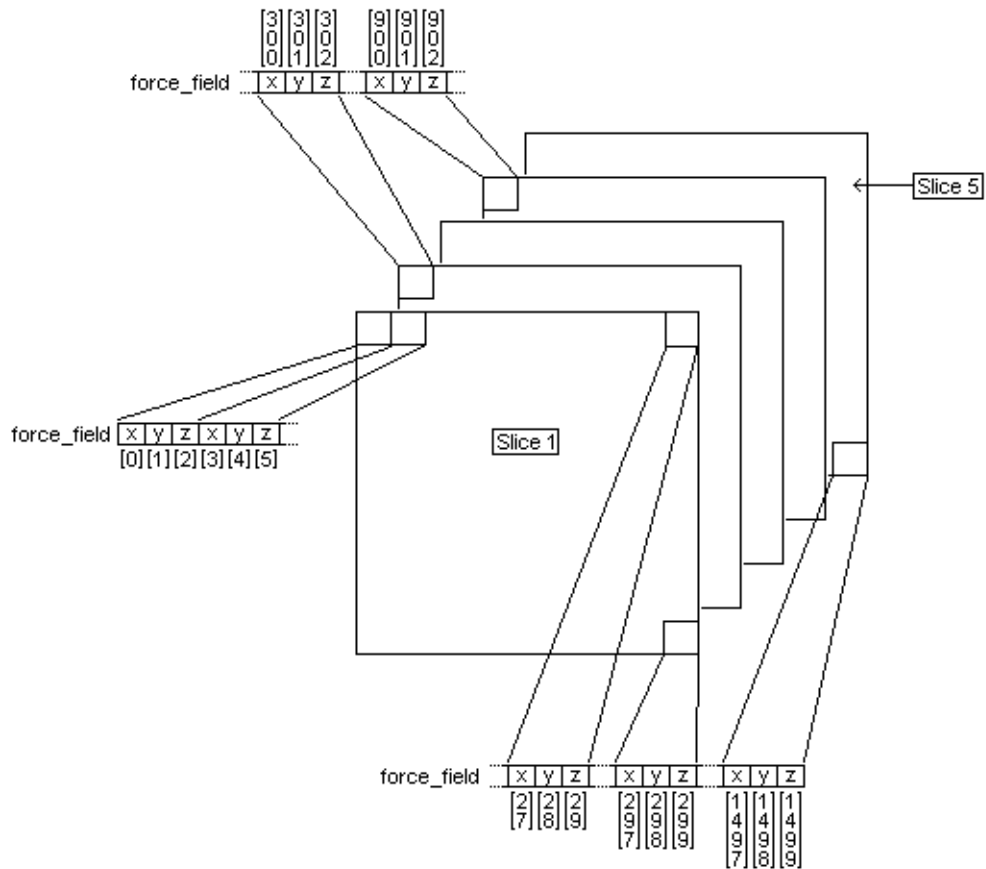


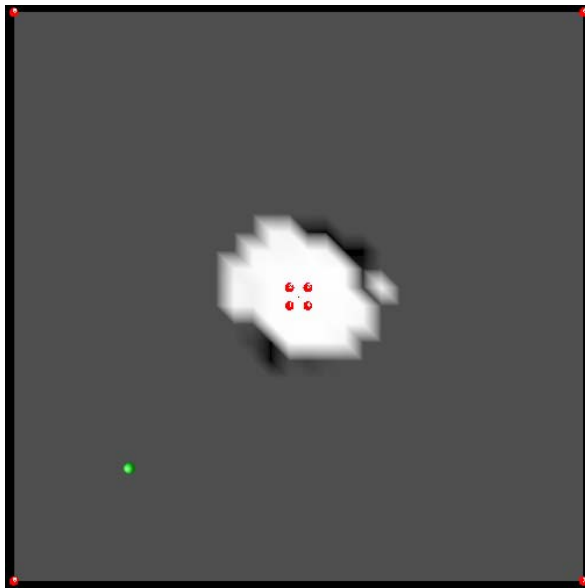
Figure 6: Mapping of the pixel gradient values into the force\_field array

As can be seen in Figure 5, there are no negative values in the force field coordinate system; but that is quite understandable since the coordinates that we are using are actually the indexing of an array. However, in the coordinate system used in the OpenDX network there are both negative and positive coordinates since origin is in the **center** of the volume, as shown in Figure 7.

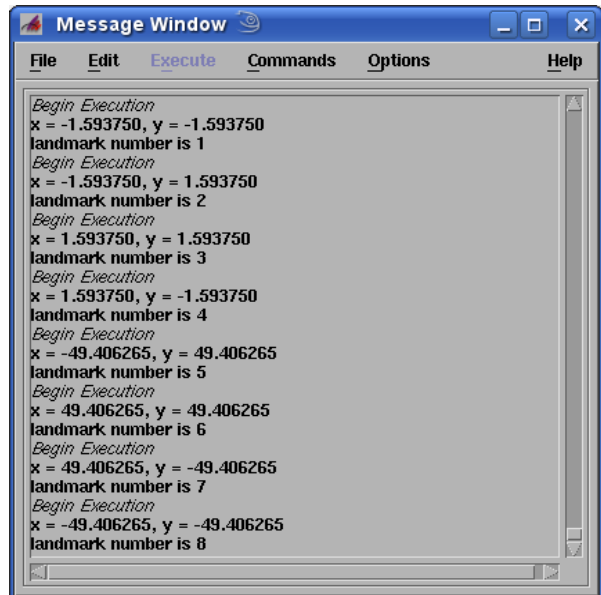


**Figure 7: Coordinate system used by OpenDX**

Additionally, the units used by OpenDX for x and y are millimeters, thus there has to be a mapping between the device coordinates, the OpenDX coordinates, and the force\_field array indices. Figure 8 show a very simple (magnified) image of a hip prosthesis. The image is 32 by 32 pixels in size and the red dots are landmarks that we placed on the corners of an image pixel and as close to the middle as possible, in order to show the values associated with their coordinates. Since the width and length of the image is even it is not possible to put a landmark right in the middle, i.e. at OpenDX coordinate (0, 0). The glyph, shown as a green dot in Figure 8, can be moved around with the haptic input device. The location of this glyph snaps to each pixel of the underlying image. In the image in Figure 8, the top-left red dot is landmark number 5, with the OpenDX coordinates  $x = -49.406265$  mm and  $y = 49.406265$  mm (as shown in Figure 9). The other seven landmarks in Figure 9 are indicated by the other seven red dots in Figure 8 with positive and negative values in compliance with the coordinate system shown in Figure 7.

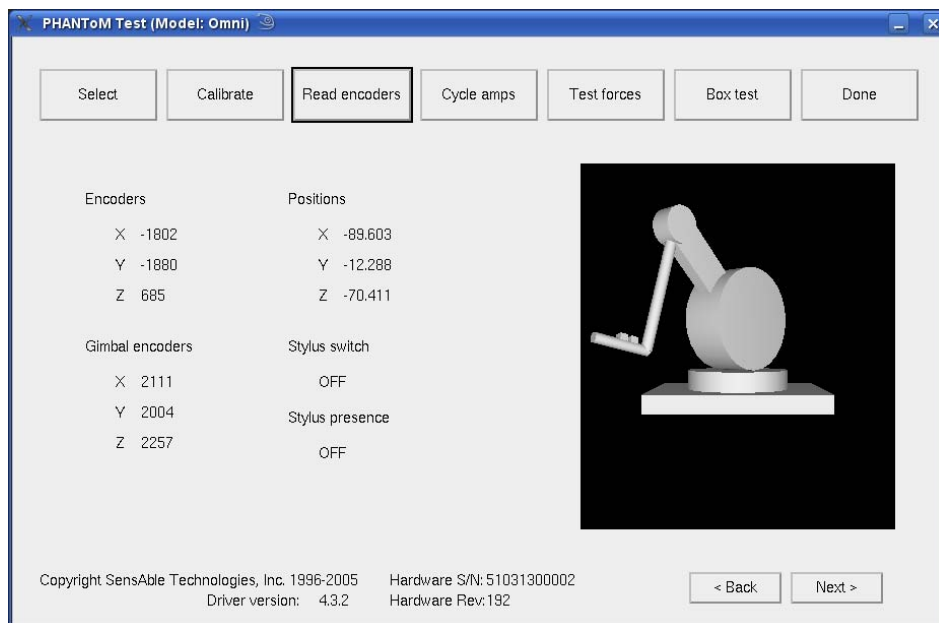


**Figure 8: An extremely zoomed in image of a hip prosthesis**



**Figure 9: Coordinate outputs in OpenDX**

There is also a third coordinate system used by the haptic device. Since it is a three-dimensional environment we also have a z-axis in addition to x and y axis. It is a standard right-handed three-dimensional Cartesian coordinate system. Figure 10 shows a screenshot from the test and calibration program supplied with the PHANTOM OmniDevice. The interesting values shown in this image are the “Positions” values. There is a rough three dimensional rendered model of the device in the black box that follows the movements of the device in the real world. At the moment when this screenshot was taken the device was rotated to the left, slightly lowered, and pushed inwards towards the base - this translates to the haptic device position values shown. According to the manufacturer these are measured in millimeters; however, we have not measured how accurate these values are.



**Figure 10: PHANTOM Test – Test and calibration program for the haptic device**

To learn the maximum range of position values of the device, we wrote a small program and used `hdGetDoublev(HD_USABLE_WORKSPACE_DIMENSIONS, aUsableWorkspace)` and `hdGetDoublev(HD_MAX_WORKSPACE_DIMENSIONS, aMaxWorkspace)` to learn the dimensions of the available workspace. The maximum workspace is the maximum extents of the haptic device workspace, although due to the mechanical nature of the device there is no guarantee that forces will be rendered correctly throughout all of this volume. Instead one should limit the usage of forces to within the usable workspace, which is a volume in which the forces are guaranteed to be rendered reliably. The workspace values we found using our program are shown in Table 1.

**Table 1: Maximum and usable workspace of the Omni device**

Axis	Maximum workspace		Usable workspace	
	Minimum	Maximum	Minimum	Maximum
<i>x</i>	-210	210	-80	80
<i>y</i>	-110	205	-60	60
<i>z</i>	-85	130	-35	35

As stated before these three coordinate systems are different, thus we have to map between the various coordinate systems as appropriate. For example, in the 32x32 pixel image in Figure 8, there is a force\_field matrix where *x* and *y* range from 0 to 31, i.e. 32 values for each of the indices. For the OpenDX system there are also 32\*32 (=1024) different points where a glyph can be. However, the OpenDX coordinate system ranges from -49.4 to 49.4 for both *x* and *y*, thus a changing by slightly more than 3mm for each movement of the glyph. Meanwhile the haptic device operates with much smaller increments (~0.055 mm according to the PHANTOM Omni<sup>®</sup> Haptic Device product info[5]) and it feels seamless to move the haptic device, even though the glyph's movement on the display does not immediately follow the movement of the haptic device.

## 2.8 Summary of background

In this chapter we have provided an introduction to all of the concepts, devices, and technologies that we will use in the rest of the thesis. Specifically we have described the haptics device that is to be used, the rate of the feedback servo control loop, how the device can be programmed, the software that we used for writing programs, the different coordinate systems that the different parts of the system use, and the tools that can be used to delay or drop network packets. Before we examine in detail how we use the haptics device and software to generate specific types of forces and send information about forces via UDP packets, we will introduce some of the related work that has help guide our work.

## Chapter 3 - Related work

This section describes related work in the area of haptics that we found useful prior to and during our work.

### 3.1 Design considerations for stand-alone Haptic Interfaces communicating via the UDP Protocol

Traylor, et al. in their paper “*Design Considerations for Stand-alone Haptic Interfaces Communicating via UDP Protocol*”[22], describe the results of building and testing a system consisting of a stand-alone haptic interface communicating with a computer over a network using UDP. A haptic device needs a fast update-rate in order to operate properly. Their measurements show that it is possible to achieve an update rate of 3800 Hz with the use of a 10 Mbit/s half-duplex Ethernet link. They found that a limitation in performance was due to the maximum rate of interrupts allowed by the operating system. When using a gigabit network card the default interrupt rate of 5000 interrupts per second limited the achieved haptic update rate to 2300 Hz. This report also discusses in detail why UDP was their choice of protocol. This report helped us understand the effects of network delay and the impact of the system’s maximum interrupt rate.

### 3.2 Haptic Feedback for Medical Imaging and Treatment Planning

Eva Anderlind's masters thesis *Haptic Feedback for Medical Imaging and Treatment Planning*[1] investigates if haptic feedback can produce speedups and increase accuracy when applied to medical imaging and treatment planning systems. The author studied physicians in their working context, using a haptic application implemented in OpenDX. An experiment was conducted with a group of physicians to evaluate their use of this application in order to see if haptic feedback gave any performance improvement for this task. The conclusion was that haptic feedback can decrease the time required to perform the tasks studied, i.e. outlining organs and volumes on CT-scans. However, no significant increases could be detected for accuracy or perceived usability[1]. A limitation of this study was the relatively small number of test subjects studied.

This thesis was important in both providing a motivation for their being a potential gain from using haptic feedback in these task. However, it was also cautionary concerning how hard it is to get volunteers for such experiments. This also proved to be problem for our experiments, i.e., we also had only a small number of volunteers.

### 3.3 Voice over IP Performance Monitoring

Cole and Rosenbluth in their paper “*Voice over IP Performance Monitoring*”[23] describe a method that uses a simplification of the ITU-T’s E-Model for monitoring and measuring the quality of Voice over IP applications. One of the conclusions of their report is that the transport level quantities that are relevant to measuring quality are (1) the delay and (2) the packet loss in the network. We want to adapt their method to measure how the quality perceived for haptics interactions over IP varies with regard to delay and packet loss. Their model provided the basic motivation for this thesis project. Figure 11 shows a plot of their model of user perceived quality of voice over IP as a function of the network delay.

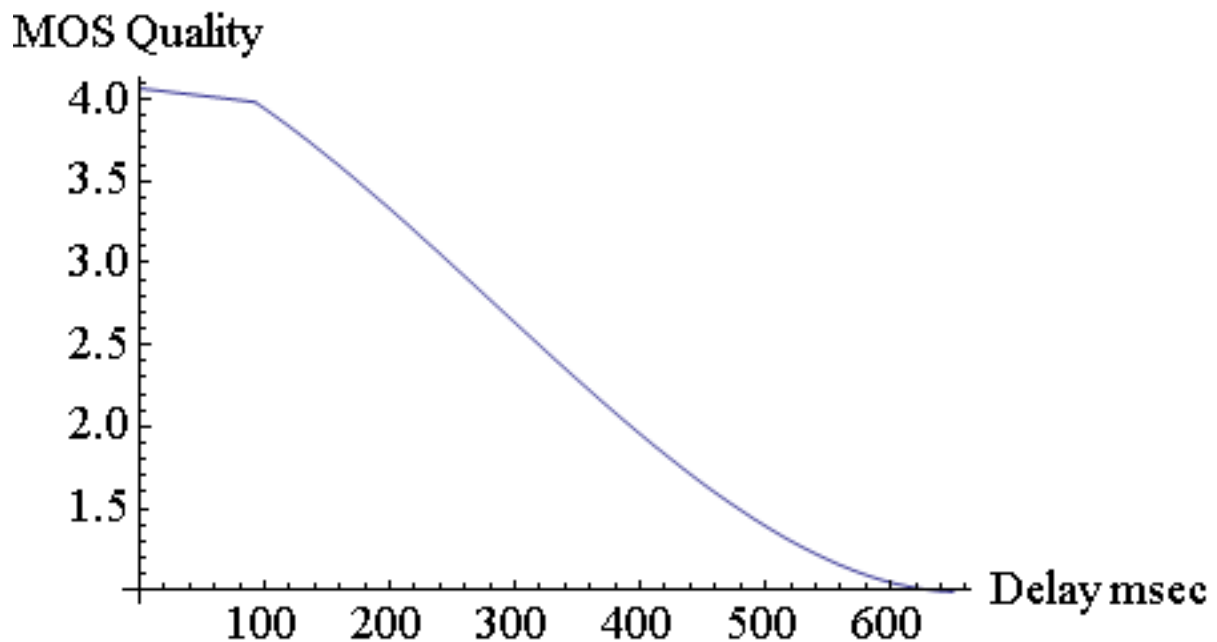


Figure 11: Relation between delay and perceived quality for voice over IP traffic (based upon the equations shown in [23])

### 3.4 Nyudemo

Nyudemo is an OpenDX program coded by Profs. Noz and Maguire. This program consisted initially of six pages of data flow diagrams in the OpenDX, and was named *test.net*. During our work with this project we came across some difficulties with getting the 3D environment working, so Prof. Noz tried to enhance the network to get it in shape for 3D. In the end we did not have time to do any tests on 3D images, nor did the 3D system work perfectly, but some enhancements were introduced into the network and it was renamed *test2.net*.

Using *test2.net* we implemented some input configuration sliders in order to be able to control our magnetic and anchor forces better. This enhanced network was used in the later parts of this project and was used in all of our subsequent tests and experiments. This network consists of only five pages of data flow diagrams and each of these five pages is described below.

Figure 12 shows the HapticOpen page. This page is responsible for opening and initializing the haptic device. The first input to the HapticOpen operator is the complete path name of an image file (as selected by the user using the FileSelector operator). The second input is a string indicating the name of the default haptic device. The third input is an integer indicating whether a slice or an iso-surface is being shown. The fourth input is another integer - the value 0 indicates a single slice of a volume (which the user will be constrained to); while the value 1 indicates that a single slice can be chosen from anywhere within the volume. When the HapticOpen module ("m\_HapticOpen") executes it take the inputs from the input arcs and initializes the haptic device (including starting the haptic server loop) and outputs a *device\_handle* that can be used to access the haptics device and a *file\_name* (as selected using the file selector).

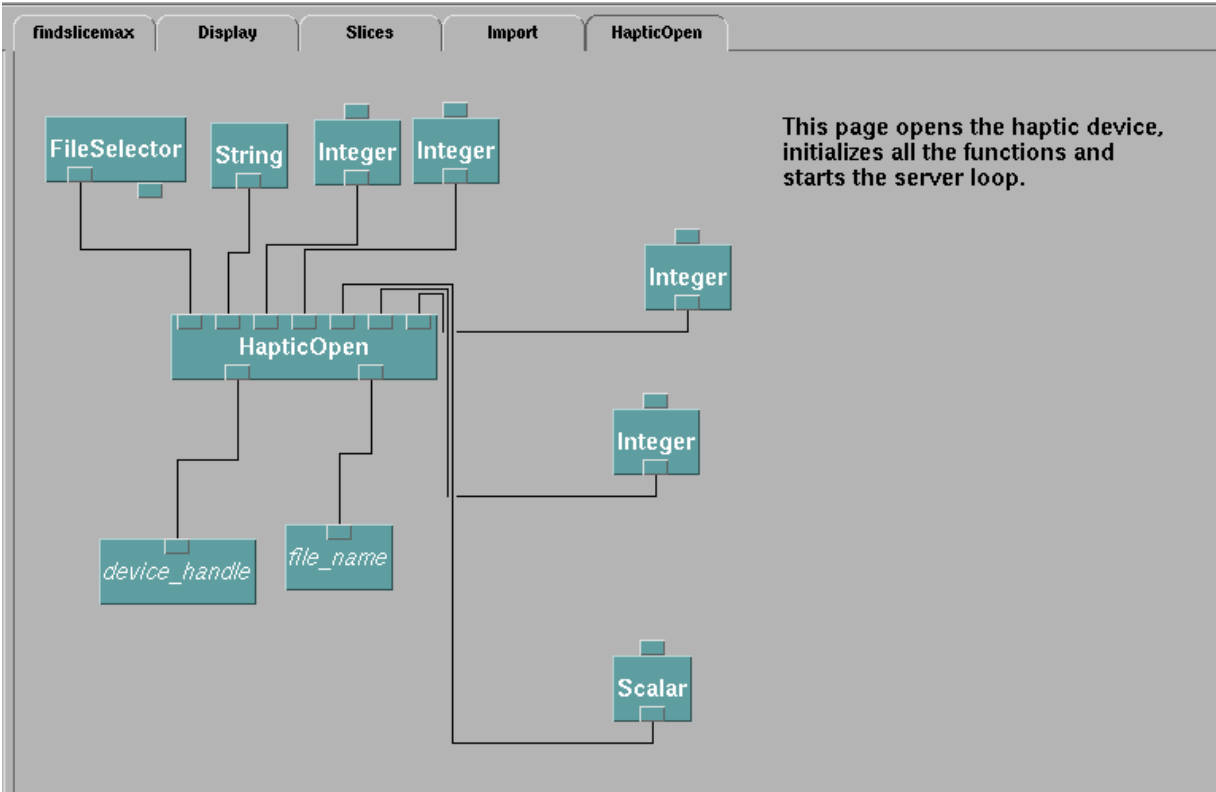


Figure 12: Nyudemo - HapticOpen page



Figure 13 shows the Import page. This page is responsible for the import of the data set and the calculation of the gradient vectors. It is also here that the user can reduce the volume (i.e., subsample) before the gradients are calculated. This page is also where the coloring of the volume is performed using the window width and window level scalar interactors in preparation for showing it as a 3D rendered volume. Coloring is the process of mapping the voxel values of the input image to colors to be used when rendering (i.e., displaying) the image. Interactors are graphic user interfaces that can be used to change the value of an OpenDX variable. At the bottom of this page is the HapticForce module that takes the gradient vectors and slice maximums as input and uses this information to apply forces via the haptic device.

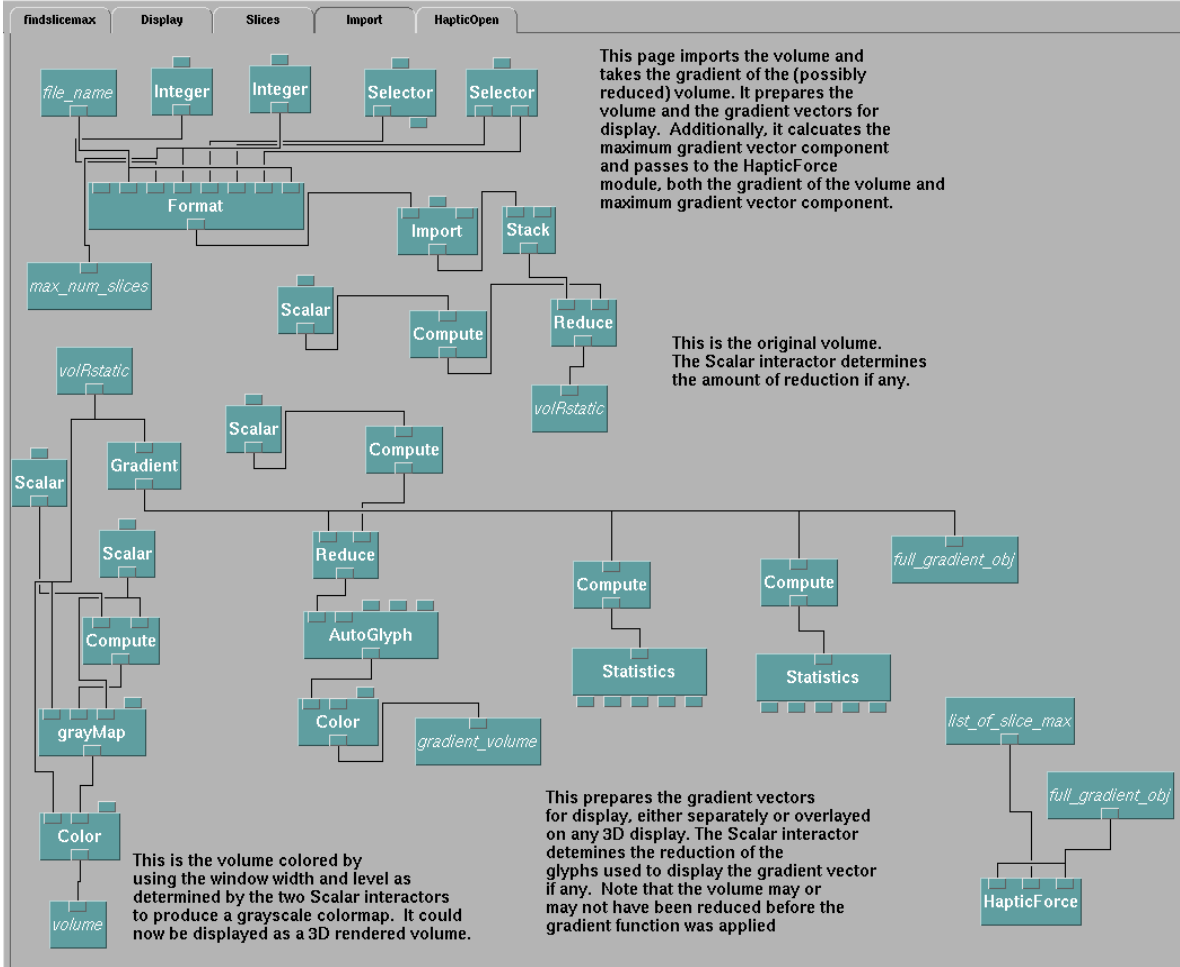


Figure 13: Nyudemo - Import page

Figure 14 shows the Slices page. This page takes the original volume and creates a slice selected by the slice integer interactor. This slice is then colored based upon the window level and width selectors, then passed on to the Display page.

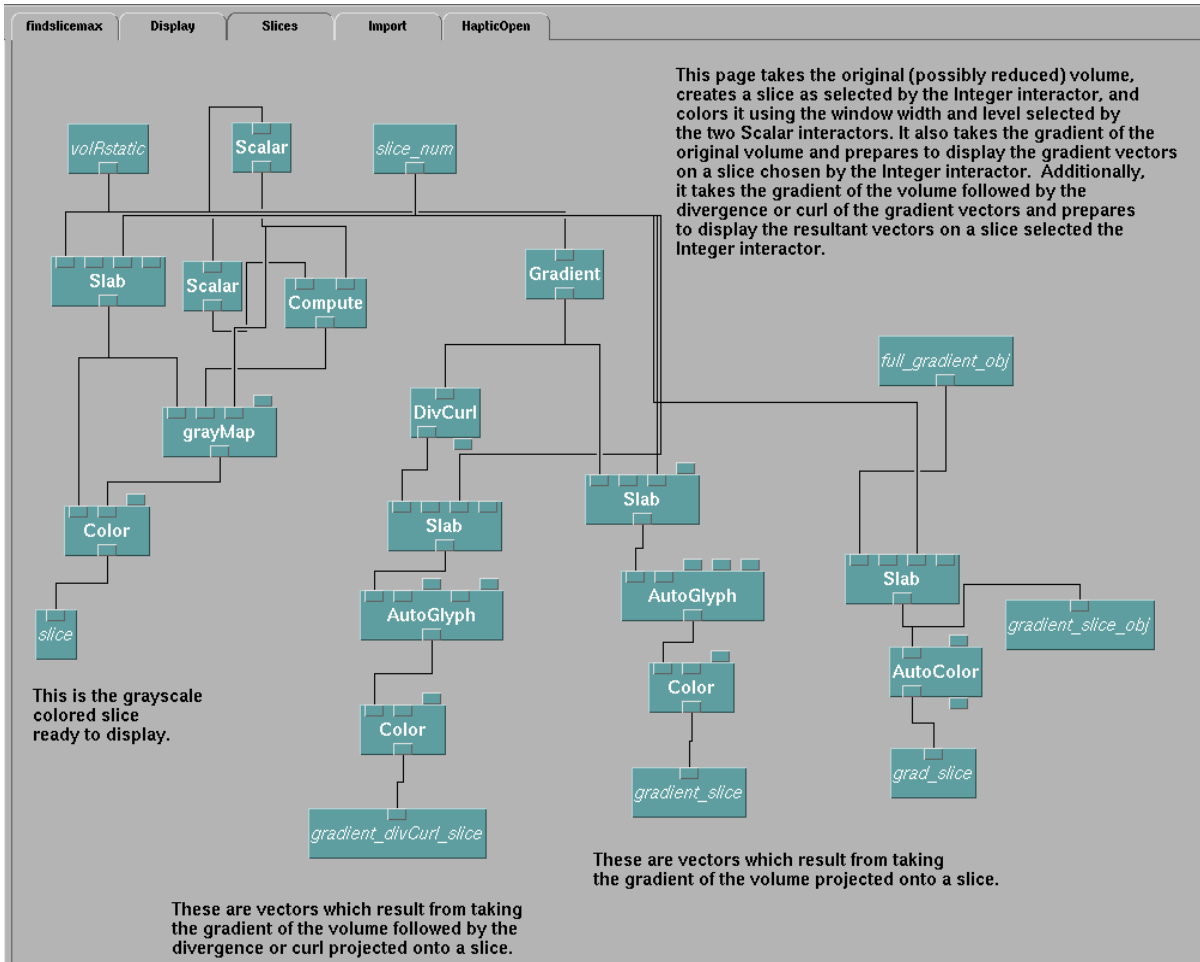


Figure 14: Nyudemo - Slices page

Figure 15 is a screenshot of the Display page. This page is responsible for displaying the volume and different slices based on the various options chosen via the earlier pages.

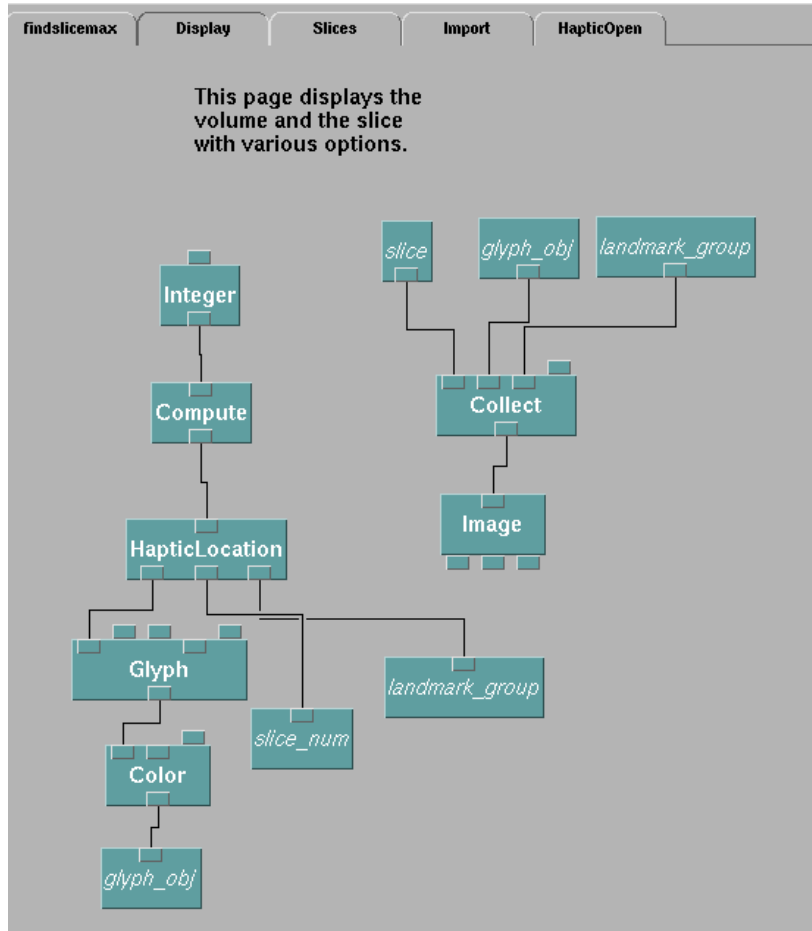


Figure 15: Nyudemo - Display page

Figure 16 is a screenshot of the findslicemax page. This page finds the maximum of the x, y, and z values for each slice and puts them in a list.

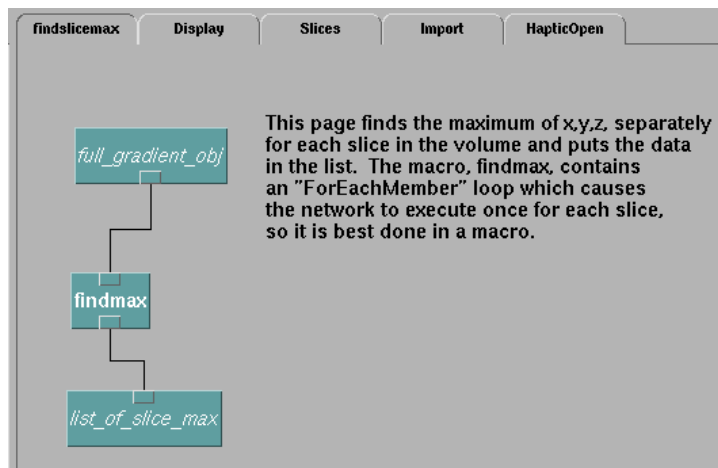


Figure 16: Nyudemo - findslicemax page

We ran this program with a simple image in order to understand how the forces were applied. The forces often forced the glyph a little bit into the whiter area, which is not optimal. The user wants to stay at the border between the black and white areas, because that is where you want to place the landmarks to outline the object. The image processing required to do this seemed at a first glance to be somewhat hard to implement. Initially the image was parsed when loaded and the intensity value of each pixel was input into the OpenDX network. Next the gradient at each pixel was calculated and saved into a matrix implemented as a one-dimensional array. The elements of this array were the three force values associated with each pixel in succession. For example the x, y, and z gradient vector values of the pixel (0,0,0) are located at `force_field[0]`, `force_field[1]`, and `force_field[2]` respectively. The forces were created from these gradients for each pixel.

We took a couple of screenshots while testing the program to try to identify the direction of the forces and see of what magnitude they were. Figure 17 shows the gradient at (14, 11) that is the topmost red landmark and the value of the gradient is (0.000, -0.387, 0.000). The haptic device produces a force that makes the glyph want to go downwards; this seems consistent with our expectation, since  $-y$  is towards the bottom of the image.

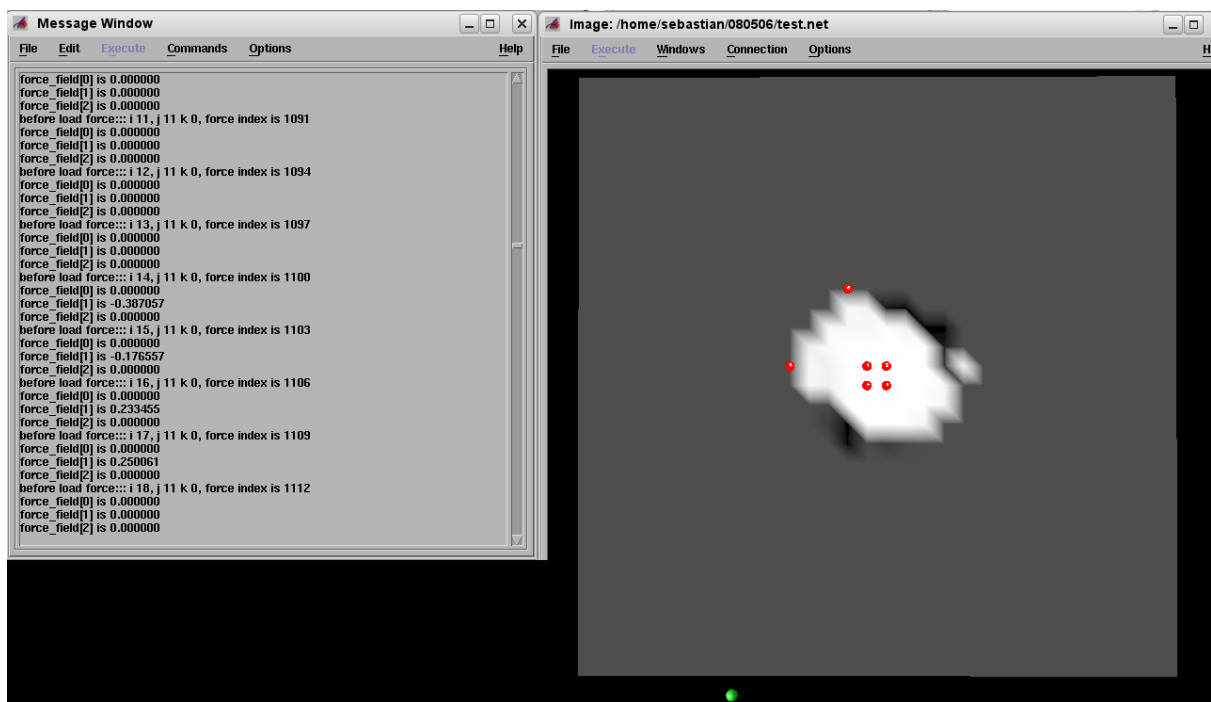


Figure 17: First screenshot while testing Nyudemo

Figure 18 shows in a similar fashion the gradient at (11, 15), i.e., the leftmost landmark. The gradient value is (0.396, 0.000, 0.000), thus the force should be directed to the right, which is also consistent with our expectation.

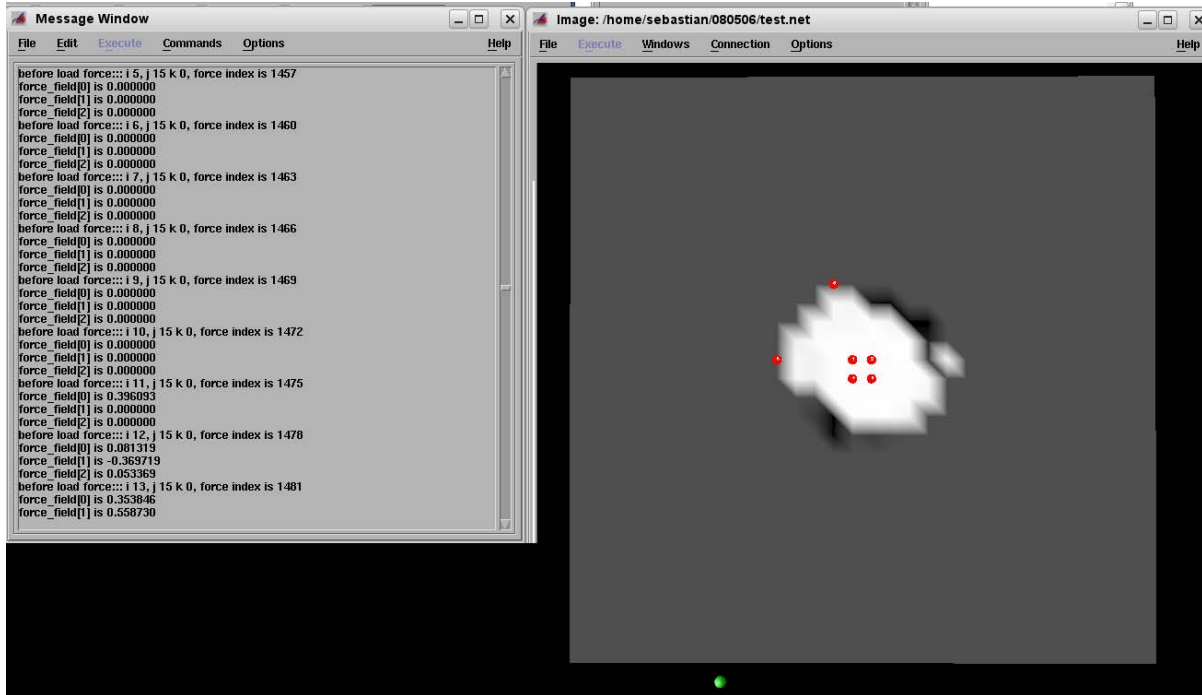


Figure 18: Second screenshot while testing Nyudemo

However, we thought that forcing the user from the border area between grey and white areas, into the white area, was not optimal considering that the user wants to place landmarks on the border. This led us to investigate if it was possible to generate forces so that the glyph would stay on the border, instead of the device forcing the glyph into another area. The changes required to do this are described in Chapter 5 - .

### 3.5 Summary of the related work

Eva Anderlind's masters thesis suggested that haptic feedback was useful, while cautioning us of the difficulty of doing user experiments that relied on volunteers. The work of Traylor, et al. indicated that added delay could substantially affect the feedback control servo loop. The work of Cole and Rosenbluth suggested that at least for voice over IP that with delays below ~250 milliseconds that the perceived quality of the voice was good to very good. The existing OpenDX Nyudemo gave us a substantial code base, while enabling us to focus on the haptic forces and avoid having to worry about the image formats or basic image processing operations. Given this background we were prepared to consider our specific project goals and how we would achieve them. These are the topics of the next chapter.

# Chapter 4 - Method

## 4.1 Goals

Since this thesis has two major parts, forces and networked haptics, we have two different (but inter-related) sets of goals. The first major set of goals concerns understanding and being able to implement different types of forces. The second major set of goals concerns examining what the effect of network effects (explicitly delay and packet loss) are on the user's perception of these different types of forces.

If we are successful in understanding both the forces and how the network affects these forces, then we expect to be able to fit a model (such as Cole and Rosenbluth – see section 3.3) to the experimental data. Unfortunately, as of the time of writing this thesis we have been unable to create a model for haptics performance over IP networks. However, we understand how to overcome (at least) some amount of delay and loss, while maintaining user perceived performance.

### 4.1.1 Forces

When we started working with the existing software the haptic forces were not functioning as a user would have wanted them to. The forces were not helping the user to stay with the glyph on the border of regions of interest, but rather forced the glyph into these regions. In order to improve this behavior and hopefully help the user to stay with the glyph at these borders, we needed to understand (1) how haptic forces work & (2) how we could control the device to generate different types of forces. As a result we had two sub goals concerning the forces part of the thesis project:

1. List and discuss different types of forces.
2. Implement and evaluate the suggested forces.

### 4.1.2 Networked haptics

Our second major goal was to understand how a haptic system behaves when used over a network. The sending of data or forces via the network introduces delay and may also introduce packet loss. As a result we had two sub goals for the networked haptics part of the thesis project:

1. Analyze how delay and packet loss affects the user's perception of the haptic system.
2. If delay negatively affects the user's perception of the haptic force, then find a solution that allows the haptic system to work despite high delay.

## 4.2 Plan for this thesis project

In keeping with the set of goals and sub goals described above we began by focusing on forces, starting with the existing system and its forces, then introducing new forces. After developing a solid understanding of haptic forces we initiated measurements of network effects on the user perceived haptic performance. Details of these steps in our plan are described below.

## 4.2.1 The existing system

The existing system that we used as a base was described in section 3.4. The task that we will focus on is the same task as explored by Eva Anderlind in her thesis (see section 3.2), i.e. a user outlining regions of interest (ROI) in medical images. As noted earlier this task is ordinarily done by using an ordinary mouse. In order to both speed up this task and make it more accurate Eva Anderlind showed that a haptic device can be beneficial. Thus in our first step we used this existing system to learn about haptic forces and to understand the task that was to be done by each user. An example of the existing system running with two partially completed ROI is shown in Figure 19.

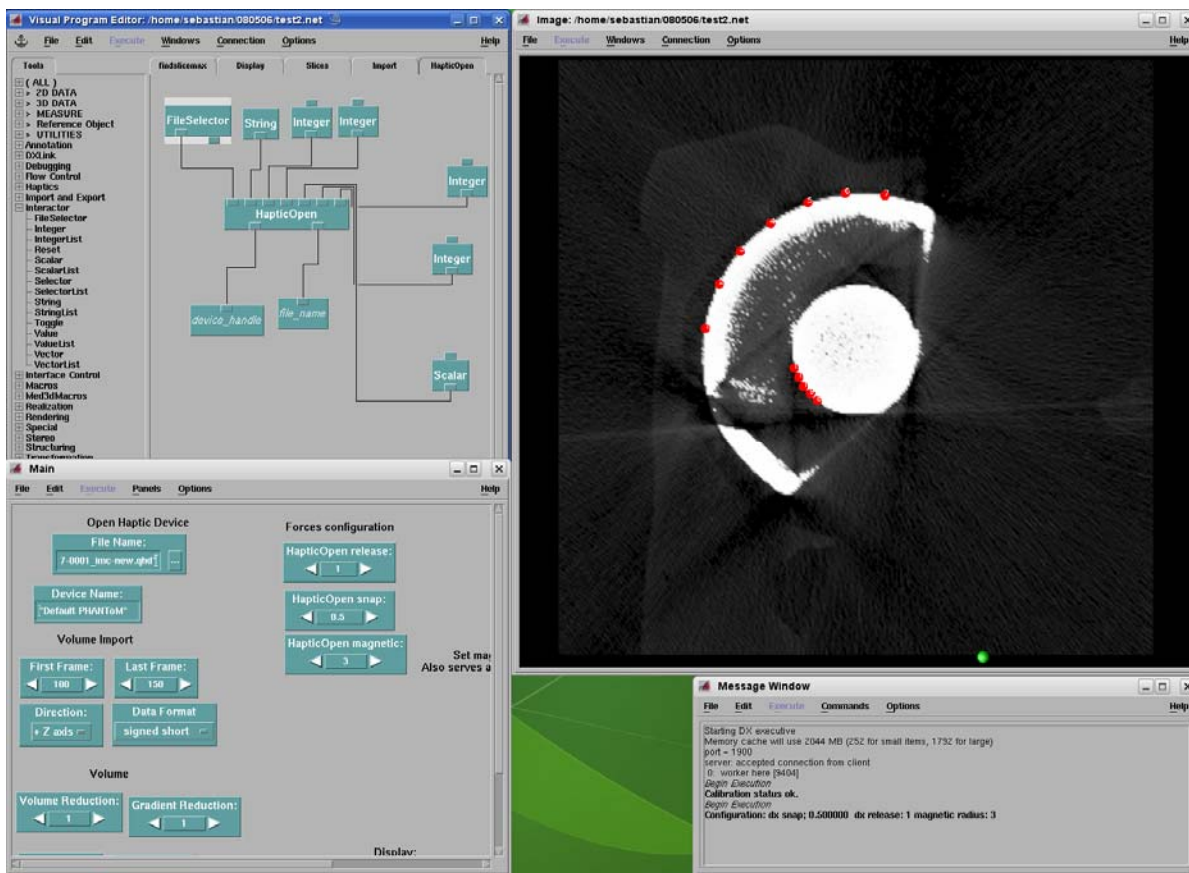


Figure 19: OpenDX network running with two partially completed regions of interests (ROI)

## 4.2.2 Forces

We want to find a force that helps users to do their work (in this case the selected task) faster and potentially with better precision. Thus in our second step we developed a list of different possible forces. Following this we implemented these forces in the system, in order to later evaluate how each of these is perceived by the user. Specifically we want to understand which type of force seems to work best. This chosen force would be used for the second part of the thesis. This work will be described in Chapter 5 -

### 4.2.3 Networked haptics

In this step we analyze the behavior of the haptic system when the task is performed over a network when delay or packet loss is introduced. We will use *TC* to introduce this delay and/or packet loss between two different computers (see section 2.6). This work will be described in Chapter 6 - .

Once we can introduce a controlled amount of delay and/or packet loss the next step is to test the system with different users in order to analyze how the perceived quality of the haptic interaction during the task changes with different amounts of delay and/or packet loss. The experiments and the analysis of the data from them are given in Chapter 7 - .

Based upon the analysis of this experimental data we will describe a solution that can maintain the quality of haptic interaction for this task, despite some amount of added delay and/or packet loss. While we describe the idea in section 7.2.3 and give an evaluation, a complete implementation and measurements of this remain for future work.



# Chapter 5 - Forces

## 5.1 Introduction to forces

In this chapter we will describe what a force is, how we are going to use forces, some different possible forces, and our implementations of some of these possible forces.

### 5.1.1 What is a force?

Halliday, Resnick, and Walter define a force as an interaction that can cause an acceleration or deceleration of a body[24]. Such an interaction is, loosely speaking, a push or a pull on the body and the force is then said to *act* on the body. The way that a force and the acceleration relate to each other was first understood by Isaac Newton (1642-1727) and is called *Newtonian mechanics*.

A force is a vector quantity, i.e. it has both a magnitude and a direction. This means that force has x-, y-, and z-components to the force. The magnitude of this vector describes how strong the force is. Forces combine according to standard mathematical vector rules.

### 5.1.2 How are we going to use forces

In the original system the forces that were applied at each pixel were based on the gradient of the pixel intensity values in the pixel itself and its neighbors. Considering that the original gradient was calculated in three dimensions, the computations were actually in terms of a voxel (volumetric pixel, the three-dimensional counterpart to a pixel) and its 26 neighbors. After the gradient was calculated it was normalized to assure that the force sent to the device would not be too strong for the device to handle. The magnitudes of these forces were perceived to be at a good level for the user, in that they could clearly be felt, but were not too strong. However, the direction of the force did not help in the process of finding the border between a brighter and a darker area, since the gradient on such a border is directed into the brighter area, thus the force from the device pushed the point locator (and hence the glyph) into this area instead of helping the user stay on the border.

Thus we wanted to continue to use the gradient as a way to find regions of interest, but to apply another force that was more suited to help the user stay on the border of the region instead of pushing them away. This led us to explore alternative forces.

### 5.1.3 Different potential forces

After using the existing system and reading the documentation concerning the haptic device we tried a number of the sample programs provided by the manufacturer. Based upon this experience we made a list of the types of forces that we might consider further in the context of this thesis project:

- Spring forces: Forces that follows the formula  $\mathbf{F} = -\mathbf{k} \cdot \mathbf{x}$ , where  $\mathbf{F}$  is the force,  $\mathbf{k}$  is the spring constant and  $\mathbf{x}$  is the distance the spring has been extended or contracted. The spring force is the most common type of force calculation used in haptics rendering due to its versatility and because it is simple to use.
- Viscous or damping forces: Forces that follows the formula  $\mathbf{F} = -\mathbf{b} \cdot \mathbf{v}$ , where  $\mathbf{F}$  is the force,  $\mathbf{b}$  is the damping constant and  $\mathbf{v}$  is the velocity of the body that is being affected

by the viscous force. (Note that velocity is the derivative with respect to time of the position.)

- Friction forces: There are several friction forces that can be simulated with the haptic device according to the Programmer's Guide[6]. These include coulombic friction and viscous friction that can be represented by the equation  $\mathbf{F} = -\mathbf{c}*\text{sgn}(\mathbf{v})$ . These two forces help to create smooth movement and transitions when changing directions due to friction being proportional to velocity for slow movement. Friction forces also include static and dynamic friction. Static friction is friction between two objects that have no relative motion to each other, e.g. prevents an object from sliding down a sloped surface. Dynamic friction is between two objects that are moving relative to each other and is resisting this motion.

The three types of forces in the above list are forces that are motion dependant, i.e. they are computed based on the motion of the haptic device. There are also time dependant forces that are computed as a function of time. In the list below are two examples of time dependant forces:

- Constant forces: A force with a fixed magnitude and direction. Such a force can be used to compensate for the gravity that is affecting the pen or end-effector of the haptic device, thus making it feel weightless.
- Impulse forces: An impulse force is an instantaneously applied force. In practice, for a haptic device, this type of force is best applied over a small duration of time.

## 5.2 Possible implementable forces

In order to find out which of the potential forces might be the most suitable we created a list of candidate forces to be examined in more detail. These candidates are:

- Bump force
- Magnetic force
- Spring force
- Wall force
- Viscous force

We tried to implement each of these different types of forces in order to test them in the real system. However, before describing the implementations and experiments with these candidates, we first describe each of these candidates in the following sub sections.

### 5.2.1 Bump force

The idea behind the bump force is to have the haptic device make a small "bump" when the tip of the device enters or passes a potential area of interest. The bump will be created by either a fast short force in one direction or a cycle of forces back and forth to create a vibration like experience. See Figure 20. Our implementation of this force is described in section 5.4.2.

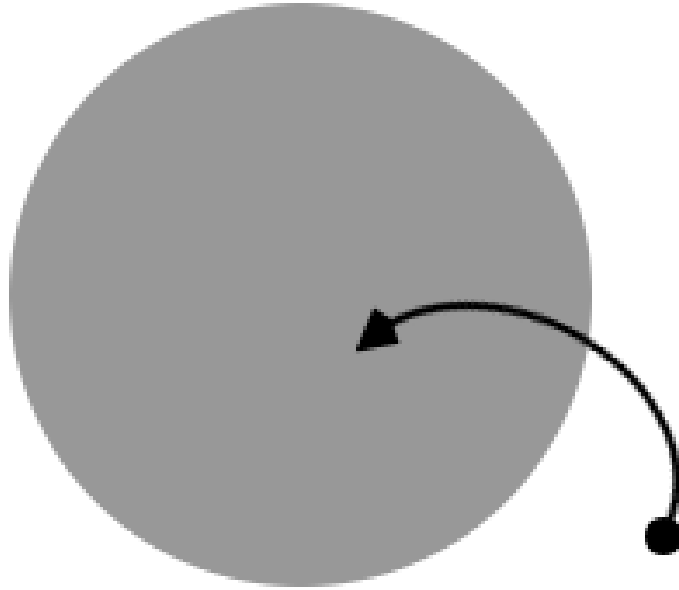


Figure 20: Bump force

### 5.2.2 Magnetic force

The magnetic force is supposed to pull the cursor and haptic device towards an area of interest in the same way a magnet attracts metal. This force may be combined with any of the other forces to keep the user at the right position or allow them to trace a line or shape. Figure 21 illustrates that as soon as you are close enough (specified by a parameter), illustrated by the black dotted line, you will be attracted to the edge of the area of interest. Our implementation of the magnetic force is described in section 5.9.

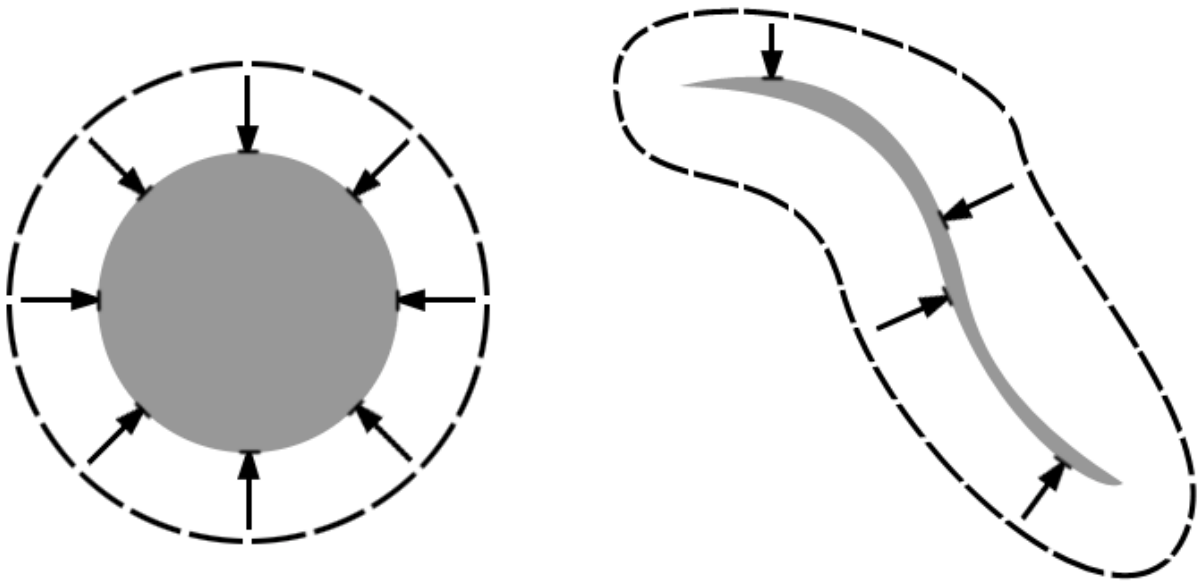


Figure 21: Magnetic force shown for two different shapes

### 5.2.3 Spring force

A spring force is supposed to help the user to keep the cursor at the right position or allow them to move along a curve or shape by imposing a spring like force if they either try to leave the area or move in the wrong direction (as determined by the curve or the shape) that the user tries to follow.

Figure 22 illustrates the concept. If you start out at point (1), then move towards point (2) it will feel like a spring is trying to drag you back to point (1). The further away from point (1) you get, the harder the force will try to pull you back. If you continue towards point (3) you will pass the threshold for release, illustrated by the black dotted ring, and the force will disappear. Our initial implementations of a spring force are described in section 5.3 on page 31.

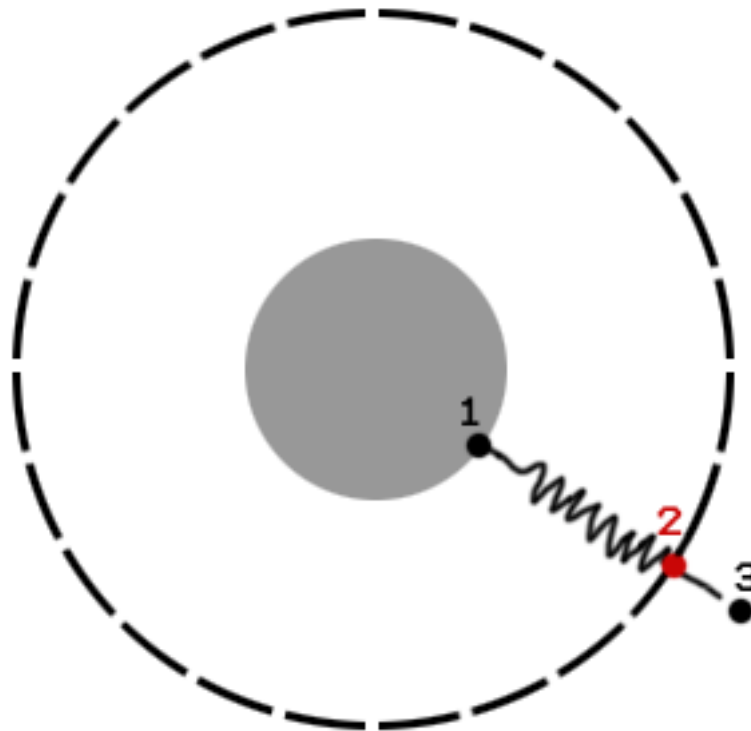


Figure 22: Spring force

## 5.2.4 Wall force

Here we thought that we could “encapsulate” areas of interest with walls, therefore disallowing the user from entering them. This would give the user the ability to know when he or she is at the right position by not being able to move further. Figure 23 illustrates an example of this where the cursor cannot pass the solid black ring in order to enter into the gray circle. One type of force that has the possibility of acting as a wall force is discussed in section 5.3.2.

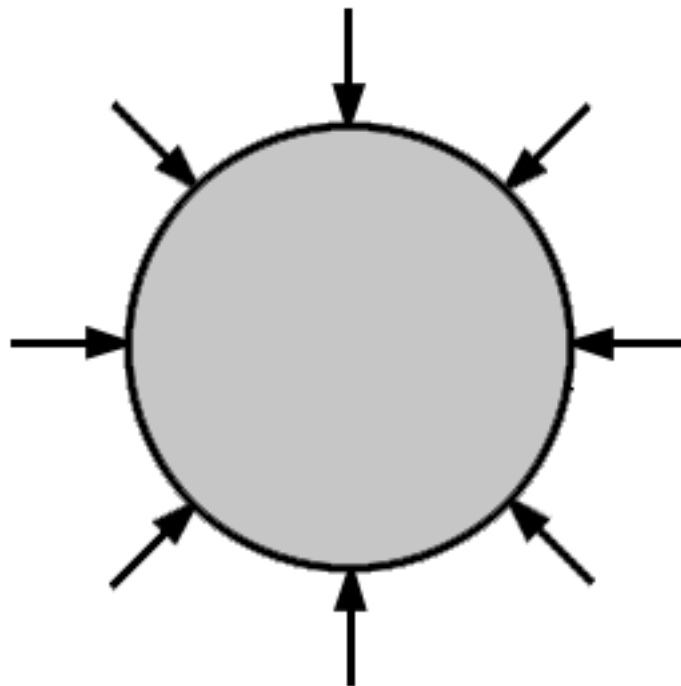


Figure 23: Wall force

## 5.2.5 Viscous force

A viscous force is a force in the opposite direction of the current movement and velocity of the cursor. A viscous force makes it harder to move the cursor as long as it is inside the area of interest, therefore the user would be able to feel both when he or she is in the area of interest (a viscous region) and when exiting the area since the viscous force would disappear outside this region.

Figure 24 shows opposing forces while inside the circle, but not outside. The idea is to also have these forces inside smaller regions, but we choose a larger circle for illustrative purposes. More on our tries to implement this type of force is described in section 5.4.1.

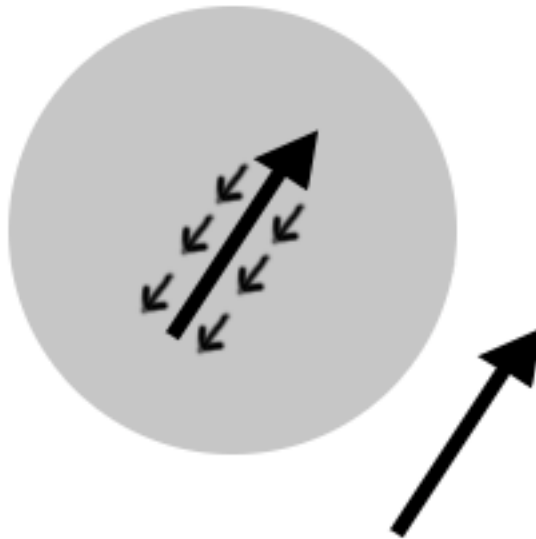


Figure 24: Viscous force

## 5.3 Spring forces

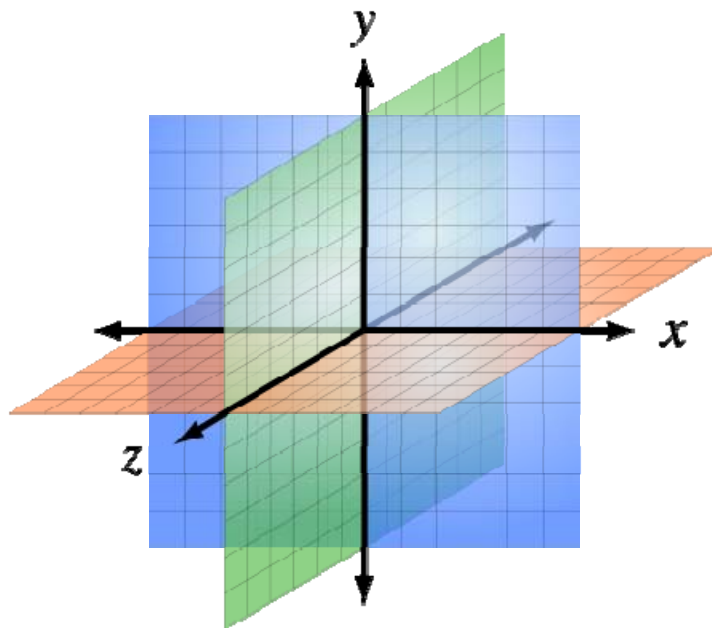
We explored several types of spring forces, as described in the sub sections below.

### 5.3.1 Anchored Spring Forces

SensAble supplies with the Omni device some small sample programs to illustrate what the device is able to do. One of these programs is called *AnchoredSpringForce*. This program simulates a spring with adjustable stiffness. The way the program functions is that when you hold down the first button on the device an anchor is created at that position in space. When you move the probe away from this anchor while still holding down the button, you feel a spring force trying to drag the probe back. The force that is applied is created using Hooke's Law:  $\mathbf{F} = -\mathbf{k}*\mathbf{x}$ , where  $\mathbf{F}$  is the applied force,  $\mathbf{x}$  is the relative distance from the anchor and  $\mathbf{k}$  is the spring constant. The spring constant is variable in the program and can be set to a value between 0.0 and 1.0 using increments of 0.025.

The spring is not fixed in direction, so you can move around the anchor freely and feel the same force as long as you are at the same distance from the anchor. It is hard to find an equivalent in the real world, since that requires something like a spring attached to a ball that is fixed in an electric or a magnetic field. However, it still is a good simulation of a spring force.

The coordinate system in the program is defined with the positive x axis to the right, positive y axis upwards, and positive z axis up from the paper pointing towards the reader, i.e. a right-handed three-dimensional Cartesian coordinate system. See Figure 25.



**Figure 25: Right-handed three dimensional Cartesian Coordinate system[25]**

We modified their example program to provide force vectors as a function of the probe's coordinates by adding the code shown in Example 4.

```

2. if(force[0]<force_old[0]-0.500 || force[0]>force_old[0]+0.500)
3. {
4.   printf("\nResulting force is %f, %f, %f\n", force[0], force[1], force[2]);
5.   force_old[0]=force[0];
6. }

```

**Example 4: Code added to the AnchoredSpringForce program**

The code in Example 4 checks if we have supplied enough force in the x-direction to exceed a threshold, and if so, it prints out the new force vector's coordinates in the form of (x, y, z). The output we got is shown in Example 5.

```

1. Resulting force is -0.001, 0.131, -0.045
2. Resulting force is -0.614, 0.139, -0.424
3. ...
4. ...
5. ...

```

**Example 5: Example output from the modified AnchoredSpringForce program**

A positive number means that the force vector is in the positive direction of the coordinate system, whereas a negative number means that the coordinate is in the negative direction respectively. Higher numbers generally means that we are further away from the anchor, which in turn causes the haptic device to apply a stronger force to pulls us back towards the anchor.

### 5.3.2 FrictionlessPlane

Another example program is called *FrictionlessPlane*. This program simulates a plane at (x, 0, z) that has no friction when probing along it. However, it has a spring force pushing you back when you try to penetrate the plane. The force applied is similar to the one in *AnchoredSpringForce*, since it is also using the formula  $\mathbf{F} = -\mathbf{k} \cdot \mathbf{x}$ . By default  $\mathbf{k}$  has a value of 0.25 and you can penetrate the plane noticeably. By default at an applied force of 5 N (i.e., five Newton), you will pop through the plane and the direction of the plane will change. After doing this you will feel the force the other way if you try to go back to where you started. This is implemented by a direction flag set to either 1 or -1. Here 1 means the plane is facing +Y while -1 means the plane is facing -Y. The default value is 1, this means that the plane is facing +Y to begin with. The setting of this flag is shown in Example 6.

```
1. static int directionFlag = 1;
```

**Example 6: Direction flag in FrictionlessPlane program**

The part of the code that checks if the plane has been penetrated is shown in Example 7.

```
1. # Get the position of the device:
2. hdGetDoublev(HD_CURRENT_POSITION, position);
3. # Check if the probe has penetrated the plane:
4. if ((position[1] <= 0 && directionFlag > 0) || (position[1] > 0) &&
    (directionFlag < 0))
```

**Example 7: Check for plane penetration in the FrictionlessPlane program**

If the plane has been penetrated, the program calculates a force perpendicular to the plane and with a magnitude depending on the penetration distance and chosen spring stiffness as shown in Example 8.

```
1. double penetrationDistance = fabs(position[1]);
2. hduVector3Dd forceDirection(0,directionFlag,0);
3. double k = planeStiffness;
4. hduVector3Dd x = penetrationDistance*forceDirection;
5. hduVector3Dd f = k*x;
```

**Example 8: Force calculation in the FrictionlessPlane Program**

We then added one line of code to output the calculated force as in Example 9.

```
1. printf("\nResulting force is %f, %f, %f\n", f[0], f[1], f[2]);
```

**Example 9: Code added to enable output of calculated force**

Next there is a check to see whether the pop-through threshold has been reached, if so then the plane should change facing, otherwise the force is applied as in Example 10.

```
1. if (f.magnitude() > popthroughForceThreshold)
    directionFlag = -directionFlag;
2. else hdSetDoublev(HD_CURRENT_FORCE, f);
```

**Example 10: Check for pop-through and possibly apply force**

Example output from the program is shown in Example 11. The increase in force is obviously because we tried to push through the plane more and more and 4.979 is just before the pop-through threshold had been reached.



1. Resulting force is 0, 0.196, 0
2. Resulting force is 0, 0.945, 0
3. Resulting force is 0, 1.761, 0
4. Resulting force is 0, 2.139, 0
5. Resulting force is 0, 3.921, 0
6. Resulting force is 0, 4.979, 0

**Example 11: Example output from the modified FrictionlessPlane program**

This example program is quite interesting because with a high enough stiffness we could use this to simulate a hard structure such as bone or a prosthesis that should not be penetrable. This is of course harder than in this theoretical example with a plane because we cannot simply set the x and z parts of the force vector to zero, but instead we have to calculate the shape of the object and use varying x, y, and z values for the components of the force at different points of the object.

This was the closest we came to implementing the wall force from section 5.2.4. In the end there was not enough time and motivation to implement the wall force since we already had implemented some forces that we thought might be suitable. However, if a wall force is to be implemented in the future, the ideas that we had to make it work correctly were:

- Find out a way to get the shape of the volume or area that is going to be impenetrable.
- Find vectors that are perpendicular to this volume or area at every point and make sure that they are pointing outwards from it. Using gradients might work.
- Decide what magnitudes those vectors will have, probably by basing them on the penetration distance as in the *FrictionlessPlane* example program.

The most difficult to realize of these ideas is probably how to find a way to get the shape of the volume and how to know if this volume has been penetrated, and as said we did not have time to find a solution to this problem.

## 5.4 Viscous forces

### 5.4.1 Viscosity

The example program *Viscosity* is a simple program that simulates a viscous damping effect (i.e.,  $\mathbf{F} = -\mathbf{b} \cdot \mathbf{v}$ ). Initially the program did not work well, but after examining the code we realized that the damping constant,  $\mathbf{b}$  in the formula, was set to too high a value and the consequence was that the haptic device applied a too strong force. This force was enough to literally grab the device's arm out of your hands if you did not have a firm grip. This happened when we tried to move the arm rapidly as the force is controlled by a formula that opposes the movement proportional to the velocity of the device. Our solution was to make sure the damping constant was set to a smaller value as originally intended in the program. However, when we tested the program with that value the force was very weak. We tried to setting the damping constant to the maximum value recommended for the device, i.e. by setting it to `HD_NOMINAL_MAX_DAMPING`, but the force still felt quite weak and was not what we had hoped.

At this point we understood why the damping constant initially had been set to a higher value than recommended. Our theory is that for high velocities the force would be too strong for the device if the damping constant is not limited, due to the nature of the force equation being used. In spite of the viscous force not being very effective in this example we wanted to give it a try in other applications; because it could prove useful as a way to increase the accuracy when outlining contours.

One idea we had was to get the direction from the velocity parameter, but set the magnitude to a fixed value. In this way we would be able to get a force that is similar to the viscous force discussed in section 5.2.5, but without the strange behaviours we described earlier in this section of strong forces grabbing the device out of the user's hand. However, after more testing we realized that reading of the velocity parameter from the device was buggy. It tends to stop updating the velocity from time to time and instead output an old value for as long as 10 seconds. This effectively prevented us from implementing a viscous force, since if the user would change direction and the velocity parameter was not updated correctly, then the result would be a force going in completely the wrong direction.

## 5.4.2 Bump force

In the process of evaluating how good different forces were in terms of giving useful feedback, we started thinking about what kind of forces we wanted to try. One idea was to apply a small force to the user when entering a specific area with the haptic device. The specific type of force we thought about was that the user would feel a small “bump” when entering the area either as an impulse force perpendicular to the direction of motion or perhaps as some kind of vibration.

We started to write a small program consisting of a main function with the required haptic device initialization, a call to the device scheduler starting our callback code, and a while loop running until we hit a key to stop. In our callback function we check if we have moved the device across the y-axis, if so, then we applied a force for a certain amount of time. By moving the device across the y-axis we mean that the y component of position changed from a positive to a negative value. Our callback function can be seen in Example 12.

```
1. hdGetDoublev(HD_NOMINAL_MAX_FORCE, &forceMagnitude);
2. hduVecSet(kickForce, 0, (forceMagnitude*0.5), 0);
3. if(last_position[1]<=0 && position[1]>0){
4.   printf("\n Passed 0 \n");
5.   hdSetDoublev(HD_CURRENT_FORCE, kickForce);
6.   printf("\n BUMP! \n");
7.   printf("Current kickForce is: %f, %f, %f\n", kickForce[0], kickForce[1],
           kickForce[2]);
8.   startBump();
9. }else{
10. hdSetDoublev(HD_CURRENT_FORCE, zeroForce);
11. }
12. hdGetDoublev(HD_CURRENT_POSITION, last_position);
```

**Example 12: Callback function of the bump force implementation**

The `startBump()` function on line 8 of Example 12 is a small function consisting of an asynchronous call to `bumpCallback()` and waiting for it to complete before continuing. This function is shown in Example 13.

```
1. void startBump()
2. {
3.   int nFr = 0;
4.   hdScheduleAsynchronous(bumpCallback,&nFr,
   HD_DEFAULT_SCHEDULER_PRIORITY);
5. }
```

**Example 13: The `startBump()` function**

Our `bumpCallback()` function consists of the code shown in Example 14.

```
1. HDCallbackCode HD_CALLBACK bumpCallback(void *pUserData)
2. {
3.   static int pFrameCount = 0;
4.   printf("bump Callback code ran! - %d \n", pFrameCount);
5.   pFrameCount += 1;
6.   if (pFrameCount < 100)
7.     return HD_CALLBACK_CONTINUE;
8.   else{
9.     pFrameCount = 0;
10.    return HD_CALLBACK_DONE;
11.  }
12. }
```

**Example 14: The `bumpCallback()` function**

These two functions, `startBump()` and `bumpCallback()` were a way for us to stall the scheduler for 100 loops so the user could feel a force for a certain amount of time. One hundred loops was roughly 100 ms in our tests, because we ran the scheduler at the default rate of 1000 Hz.

The end result of this program was a clearly palpable force felt as a bump, when you moved across the y-axis. We had this force in mind for later use, but eventually chose not to implement it into the OpenDX program. This choice was based on the nature of the force, as it would not really fit in to the concept of following a line. However, it would probably work well as a way to feel the cell walls of a spreadsheet.

## 5.5 Implementing a spring force in the OpenDX HapticsModule

In this section we describe how we implemented a spring force (the force is described in section 5.3.1) in the OpenDX HapticsModule, thus we could use it with actual image values.

### 5.5.1 Overview

The idea is to use the gradients in the force matrix as positions where we want to attach the spring. Whenever the glyph is at a position in the image where the gradient is larger than a predefined anchor threshold, we set the anchor at that position. The anchor will then correspond to one end of the spring and the current position of the glyph will be the other end of the spring. This would correspond to Figure 22, with (1) being the anchor and (2) being the position of the glyph.

## 5.5.2 Movement

When the glyph is at a position where the corresponding gradient is 0, then no force will be applied. This allows the user to move the glyph freely in the image where there are no regions of interest. When the glyph comes in contact with a point where the gradient is larger than the anchor threshold, the anchor is set. As long as any further movement is along a line or in a region where the gradient is larger than the anchor threshold, then the anchor will be moved and there will be no resulting force. However, as soon as the glyph leaves the line or region, the anchor will remain set. The distance between the anchor and the current position of the glyph will be used to calculate a force in the opposite direction in order to drag the glyph back to the anchor's position. If the user tries to leave the current region or line, the distance will increase (as will the force) until the release threshold is reached. When this happens the force will disappear, allowing the user to easily move the glyph to other parts of the image. Figure 27 shows a flow chart of the proposed algorithm.

We need to define two values:

1. `Anchor threshold` determines how large the gradient needs to be in order to set the anchor.
2. `Release threshold` determines how far away from the anchor the glyph needs to be in order to be released.

The actual value to use for the thresholds will probably change from image to image and may also depend on what parts in the image the user wants to snap to. Hence it was desirable to make it easy to change this value. Note that the anchor threshold always has to be larger than 0, since otherwise we would set the anchor on every single point of the image. We added the possibility to change both the anchor and release threshold in the application's configuration via a set of interactors. See Figure 26.

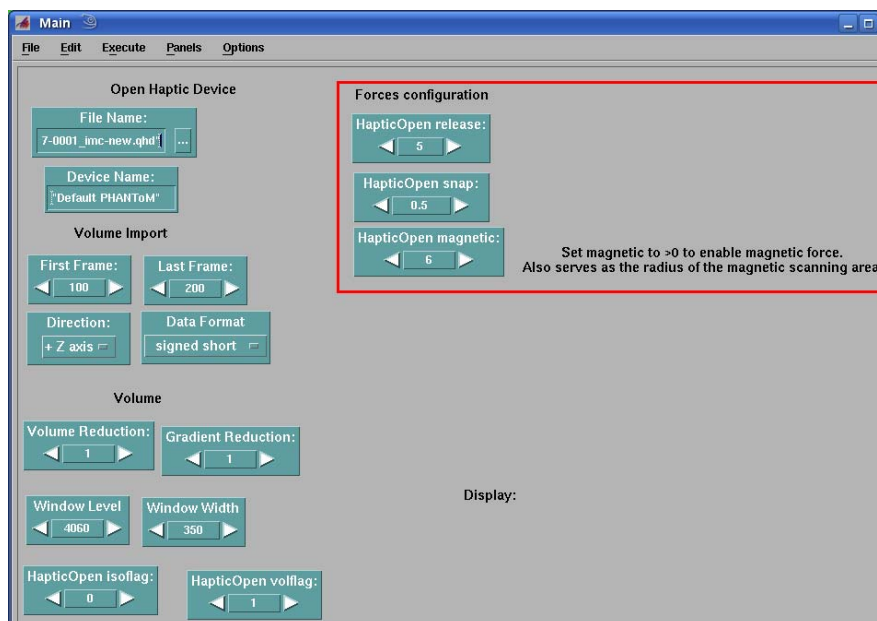
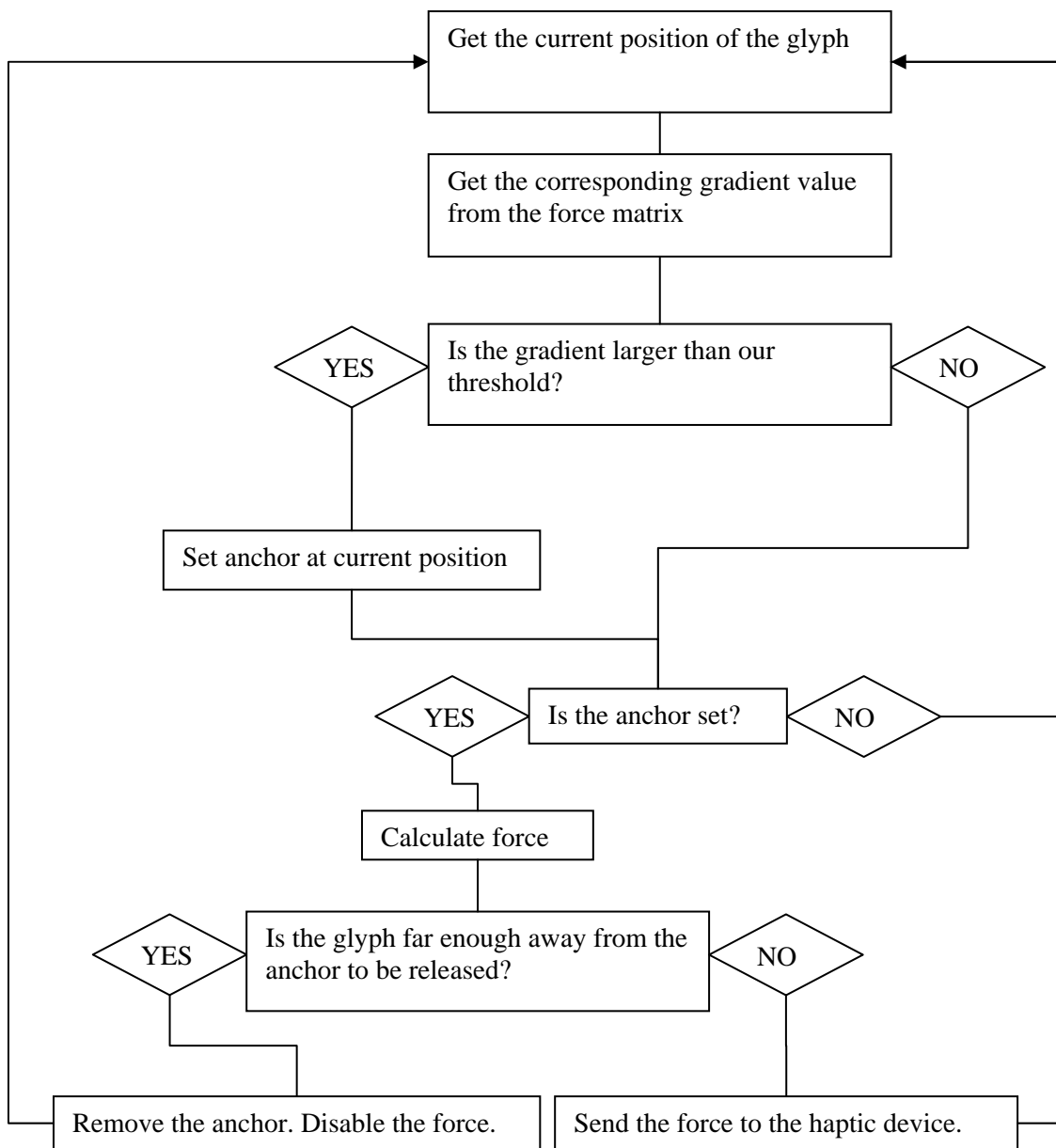


Figure 26: OpenDX configuration panel



**Figure 27: Algorithm for implementing spring forces in the HapticsModule**



### 5.6.3 The spring force

The force is calculated as  $\mathbf{F} = -\mathbf{k}\mathbf{x}$ ; where  $\mathbf{x}$  is the vector distance between the anchor and the current position of the glyph and  $\mathbf{k}$  is the stiffness of the spring. The larger the value of  $\mathbf{k}$  the more difficult it will be for the user to expand the spring. The API offers a method to query the device, in order to get the maximum stiffness of the spring that the device can handle:

```
hdGetDoublev(HD_NOMINAL_MAX_STIFFNESS, &gMaxStiffness);
```

Since we only have a two dimensional matrix and no visualization of the movement we ignore the z-dimension and always set the component of the force in the z-direction to 0.

### 5.6.4 Callback loop

The callback loop functions as described by Figure 27. We need to supply a value for the threshold. This is the maximum magnitude of the force vector before we release the anchor; that is the maximum length of the spring as shown by the dotted line in Figure 22. The callback loop consists of:

- First the current position is converted to an integer in order to be able to index into our force matrix.
- If the value from the force matrix at the current position equals 1, then we place the anchor and set the RenderForce to true.
- If RenderForce is true, then we calculate the force as explained above, see section 5.6.3. If the force is smaller than our release threshold value, then we apply the force, else we set RenderForce to false and send a zero force to the device.

### 5.6.5 Evaluation of the spring force test application

The force worked as we had expected. The anchor is set when entering the frame without any delay or strange behavior. The probe can also be moved around freely as long as you follow the frame. The one problem is that it can be a bit hard to trace a line or shape. As soon as you move just a bit outside the shape you are trying to follow, a force will be felt, stopping your movement until the cursor is moved sideways to move back onto the frame. One possible solution to this could be to combine a magnetic force (see section 5.2.2) with this spring force. If the cursor is still trying to follow the line, but leaves the frame just a little bit, then the anchor will be moved to the closest point of the frame and the cursor will be dragged back to the frame.

## 5.7 Stage two: Adding the spring force to the HapticsModule in two dimensions

In order to implement the spring force into the OpenDX HapticsModule we had to change the HandleCallback function. The functionality of the algorithm is exactly the same as in the test application implemented in section 5.6.2. The only modification we had to make was to change the way it locates the current position in the force matrix. Our test application used a simplified matrix where every point contained one of two values: 0 or 1. The way the matrix containing the gradients is implemented in HapticsModule was described in section 2.7 and shown in Figure 6 in the same section. Since the gradients are stored in a single array and the mapping between the haptic device coordinate system and the OpenDX coordinate

system is not 1:1 some conversion is needed in order to find the position in the array that corresponds to the current position of the glyph.

### **5.7.1 Evaluation of the spring force OpenDX HapticsModule implementation**

The problem with tracing lines from the test application still persists. We tried using two different thresholds in order to solve the problem with “nagging” when tracing lines. The idea was to lower the threshold when an anchor is already set in order to make movement along a line easier. The result was that the movement became easier, but the precision became much worse. The loss of precision was not acceptable and the idea of using two thresholds was dismissed.

Except for the problem that tracing lines is not as smooth as we would have wanted it to be, we felt that the implementation of the spring force worked well.

## **5.8 Stage three: Implement in a OpenDX network in three dimensions**

The current OpenDX net and the HapticsModule code do not consider a 3D environment (although the 3D image is read in and used to calculate a three dimensional force matrix). The limited time available for this thesis project did not allow us to extend our implementation to three dimensions; therefore we did not try our spring force in 3D.

## **5.9 Implementing a magnetic force**

This section describes our implementation of a magnetic force.

### **5.9.1 Overview**

The idea behind the magnetic force is to make the glyph on screen, and with it also the haptic device, automatically move to points of interest when in close proximity. The implementation of this is quite straightforward. Our spring force implementation checks if the current position of the glyph meets the requirements for an anchor to be set. In order to produce the desired magnetic force we scan all the points in an area (or volume) around the glyph and check if any of these points meet the requirements for shifting the anchor point.

### **5.9.2 Test application**

We began by using our test application (see section 5.6) as the basis for our magnetic force implementation. We decided to do the scan in the shape of a diamond, since we thought we had a fairly easy algorithm for this. The scan could of course also be done in the shape of a box or a circle.

Figure 29 is a graphic representation of the algorithm. The green dot in the middle represents the glyph. The blue dots are visited on the first loop of the scan and the red dots represent the second loop of the scan.



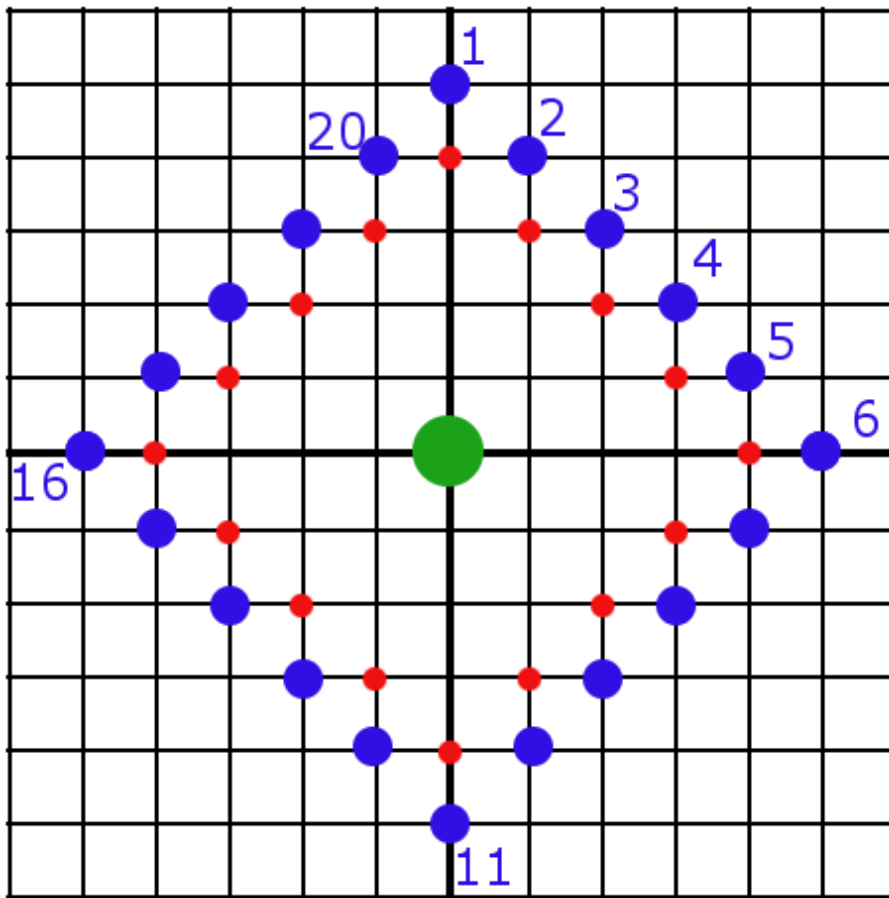


Figure 29: Magnetic force scan

Example 15 shows the code for this scan. The only value that can be modified is *radius*. This value decides how far away from the glyph we will scan for new anchor points, the metric is in terms of the number of points. The example has this metric set to 5, therefore our scan will begin at point (1) in the figure. We start the scan with the outermost shell of the diamond shape and progress inwards. This is due to the fact that we want the anchor point closest to the glyph as the final one.

The for-loop on line 7 steps through the different “shells”. The blue shell is the first one, the red which is the second, the loop then continues until the glyph in the middle has been reached. Lines 9-13 check to see if the point that currently is being scanned is an anchor point, if so an anchor is set. Lines 16-28 are responsible for changing the temporary position in order to rotate through the current shell.

We start out at position (1) in Figure 29. Since line 16 is true we will move the position to (2). This continues until we are at position (6). Now line 19 will be true and the movement will continue towards position (11). When we have reached point (20) and thus have completed the outer shell, then radius will shrink by one and the red shell will be evaluated.

```

1. int radius=5;
2. int counter1;
3. int temp_set=0;
4. int temp_x=0;
5. int temp_y=radius;
6.
7. for (radius; radius>0; radius=radius-1){
8.   for (counter1=0; counter1<radius*4; counter1=counter1+1){
9.     if (matrix[xPos+temp_x][yPos+temp_y]==1){
10.      if (!(matrix[xPos][yPos]==1)){
11.        tempAnchor[0]=xPos+temp_x;
12.        tempAnchor[1]=yPos+temp_y;
13.        temp_set=1;
14.      }
15.    }
16.    if (counter1<radius){
17.      temp_y=temp_y-1;
18.      temp_x=temp_x+1;
19.    }else if (counter1<radius*2){
20.      temp_y=temp_y-1;
21.      temp_x=temp_x-1;
22.    }else if (counter1<radius*3){
23.      temp_y=temp_y+1;
24.      temp_x=temp_x-1;
25.    }else {
26.      temp_y=temp_y+1;
27.      temp_x=temp_x+1;
28.    }
29.  }
30.}

```

**Example 15: Magnetic force algorithm**

### 5.9.3 Evaluation of the magnetic test-application

The implementation in our test application worked very well. It behaved as expected and drags the user towards the anchor points without any delay or glitches. The only slight problem that the current algorithm has is that if we have several points in the same shell it will choose the point closest to the end, which is the one closest to it, or point (20).

A smarter way to do this is to keep track of which direction the glyph (the green dot) is moving, then give the corresponding side precedence. For example, if we are moving south-east, then points between (6) and (11) should be chosen as anchor points over the others.

#### **5.9.4 Use the magnetic force to make tracing of a line easier**

An idea was to use the magnetic force to make the movement along a line or shape easier. However, in practice this did not work well at all. If we allowed the magnetic force to set anchors while there already was an anchor set two problems arose.

First the glyph and haptic device started to move themselves, since the algorithm found new points at all times when on a line. The next problem occurred when there were several different lines or shapes in close proximity. The glyph started to bounce back and forth between these in an endless loop.

In order to make a smarter algorithm that helps the user to move along a shape more variables have to be taken into account, for example checking if the user is actively moving the glyph and if so in what direction they are moving it.

#### **5.9.5 Implement the magnetic force into the OpenDX network**

As with the spring force implementation, the only changes we needed to make between our magnetic test implementation and the implementation in the HapticsModule concerned the coordinate system. All other functionality works the same.

#### **5.9.6 Evaluation of the magnetic force OpenDX HapticsModule implementation**

We are very pleased with the way the magnetic force turned out. It really adds an extra layer to the functionality when the user is drawn towards the regions of interest (ROI) in the image. During all our testing the magnetic force has worked flawlessly. As described in section 5.9.4 we tried to use the magnetic force in the test application implementation to improve movement along lines. We were unfortunately not successful at implementing this idea in the HapticsModule either.

## Chapter 6 - Networked haptics

This is the second logical part of this masters thesis. We were asked to analyze the haptic system's behavior over a network with different amounts of delay and packet loss. After this, the next task was to see if we could make the system work over links with a large amount of delay.

### 6.1 Networked haptics introduction

In the following subsections we introduce the networked haptics part of this thesis project.

#### 6.1.1 Networked haptics: What will we do?

How does the system behave when there is delay or packet loss? It is desirable to be able to use the haptic device over a network. There may be several different reasons for this:

- The device is expensive; you cannot have one attached to each computer.
- The device is fragile, try to avoid unnecessary movement.
- The data may be at a different location than the operator.

In order to see if and how networked haptics would work we designed a method to use the haptic device over a network. When the system was up and running we could introduce both delay and packet loss in our test network in order to observe the behavior of the system under specific delay and packet loss conditions.

#### 6.1.2 Networked haptics: Initial configuration

The initial configuration was previously developed by M. E. Noz and G. Q. Maguire Jr and was described in section 3.4.

#### 6.1.3 Networked haptics: How to split the code

In order to do the network testing we have to decide how to split the system. The system has several different parts and the first question is where to place each of these parts, specifically the:

- Haptic device
- Screen (we assume that this consists of the frame buffer and display)
- Image data
- Force matrix
- Force calculation software

The goal is to place a network between the haptic device and the rest of the system. However, there is not a single obvious way to do this. We had three different ideas about how to divide the different parts of the system between the two locations. The basic equipment to have at each location is a computer and these two computers are connected through a network.

## 6.2 Proposed method 1

When the computers are in close proximity or even share the same screen it would be convenient to be able to enable a user at one of these computers to use the haptic device without having to move cables around.

This method has the following setup:

- Computer A has the haptic device connected.
- Computer B has the force calculation software, the image data, and outputs the image data to the screen.

In Figure 30 below is a picture of this setup.

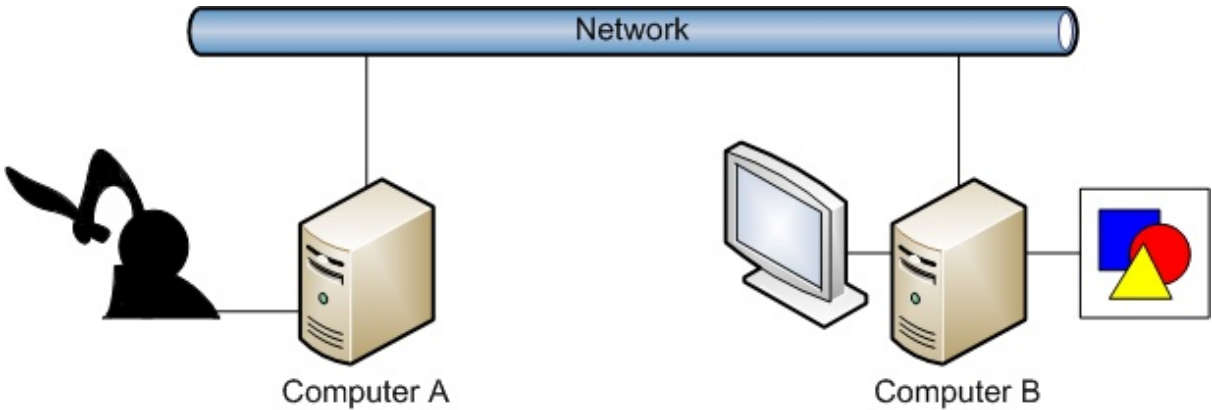


Figure 30: A scenario with the haptic device in close proximity to the computer computing and displaying the image data

The only thing computer A does is to get the device position, send the position to computer B, and receive a force update to apply to the haptic device. Computer B receives the position update, calculates the force, and sends it to computer A. Computer B also displays the image and shows the movement of the glyph. See Figure 31 below for the data flow of this method.

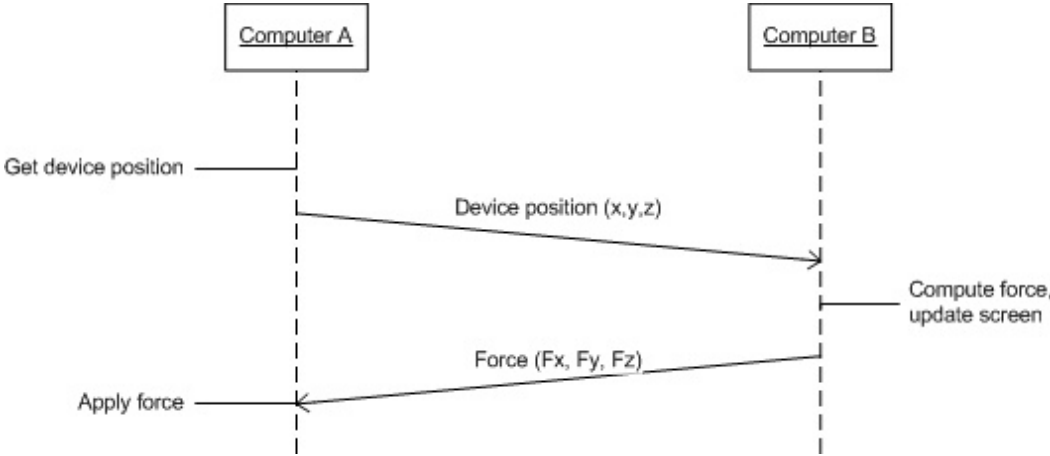


Figure 31: Data flow for proposed method 1

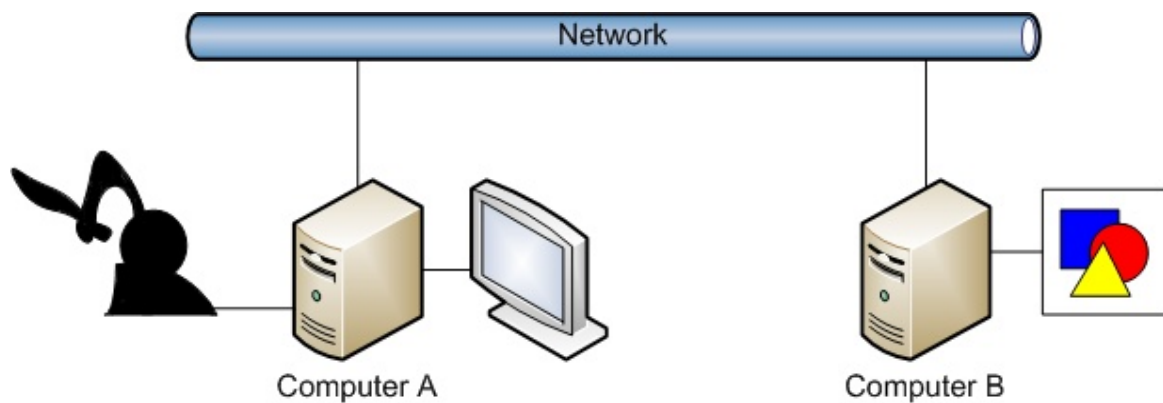
### 6.3 Proposed method 2

The second possible scenario is when the operator and the haptic device are at a different location than the image data. In this scenario we want to be able to use the system without first having to transfer all the image data.

This method has the following setup:

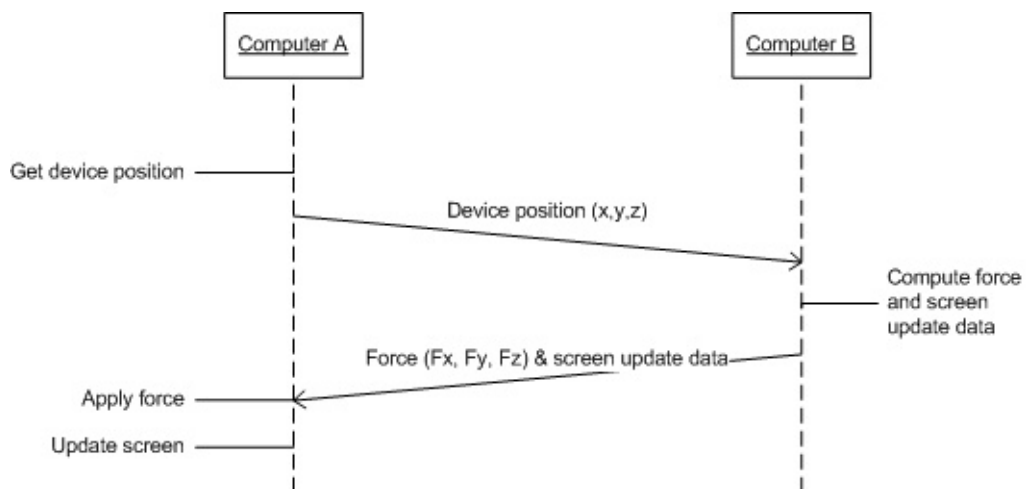
- Computer A has the haptic device and outputs to the screen.
- Computer B has the image data and the force calculation software.

See Figure 32 below for a picture of this setup.



**Figure 32: A scenario with a local haptic device and image display, but with remote computation of forces**

Here computer A gets the position of the device and sends it to computer B. Computer A then receives the force for the haptic device and an update for the screen. Computer B receives the position update and calculates the force. It then sends the force and a screen update. If either of the two computations show to be time consuming it may be a good idea to decide which one to prioritize and send two messages. One message for the force update and another message for the screen update. This allows the most important update to be sent without having to wait for the computation of the second calculation. See Figure 33 for the data flow for this method.



**Figure 33: Data flow for proposed method 2**

A slightly different approach to this idea would be to handle the screen updates locally since there is instant access to the device position available at Computer A. See Figure 34 for a flow chart of this approach.

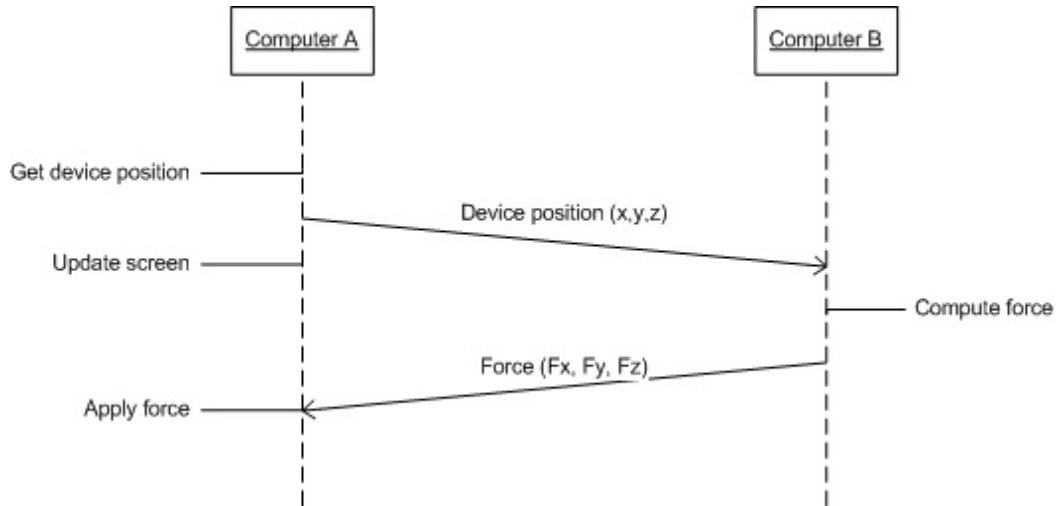


Figure 34: Updated data flow for proposed method 2

### 6.4 Proposed method 3

The last method was designed to quickly set up a test system linking the two computers. This method would allow us to quickly start our testing, but does not have real world usage.

This method has the following setup:

- Computer A has the haptic device, the force calculation software and outputs to the screen.
- Computer B would just bounce IP-packets back to computer A.

See Figure 35 for this setup.

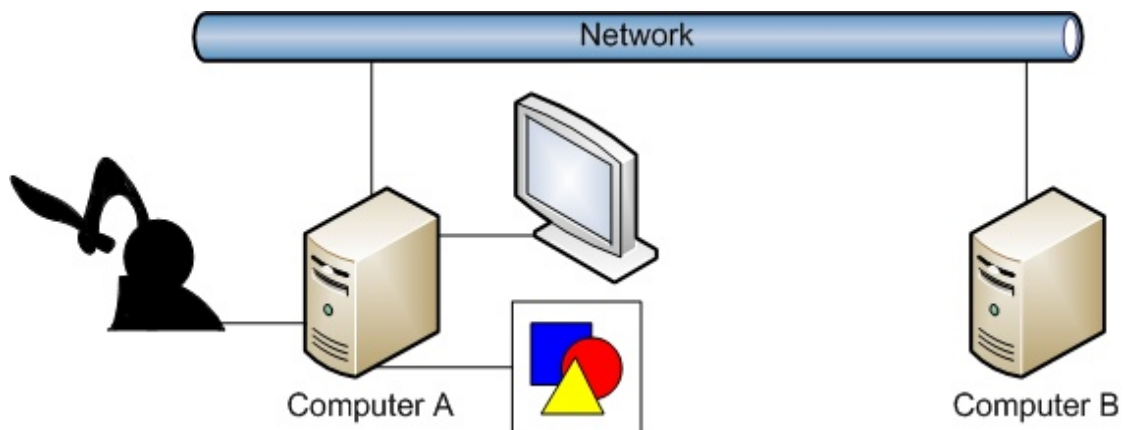


Figure 35: Configuration for testing delays

Computer A would get the device position and calculate the force. It would then send the calculated force to computer B, which in turn would bounce it back. While waiting for the force reply it would update the screen. The last step is to send the force to the device. The

only thing that computer B does is to receive an IP-packet and then directly send it back to computer A. Figure 36 shows the data flow for proposed method 3.

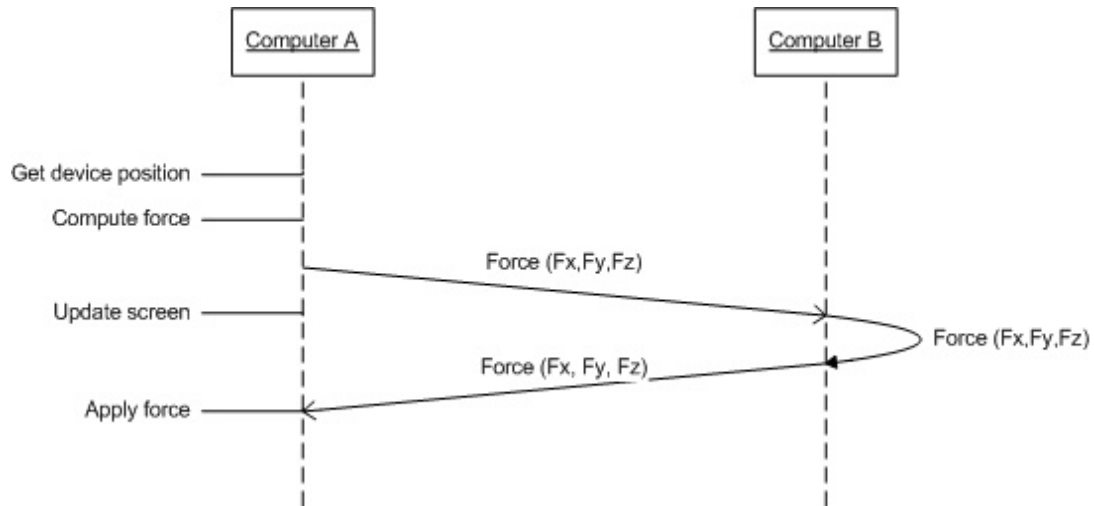


Figure 36: Using a remote loopback of force messages for testing

## 6.5 Obstacles with these methods

The haptic device loop runs at a default rate of 1000 Hz; as mentioned in section 2.2.1. If this speed cannot be maintained, then the device may start to behave strangely. When experiencing delay and packet loss a decision of how to maintain the control loop rate has to be made. The problem arises inside the haptic callback loop. Each iteration of the loop should send a force to the device. However, because it may take tens or hundreds of milliseconds between sending the position update and receiving the calculated force a strategy for how to maintaining the control loop rate needs to be formed. Possible solutions:

- A: Continue to send the last received force until a new update is available.
- B: Send a zero force until a new force is available.
- C: Block the loop and wait for an update.

### 6.5.1 Idea A

For small amounts of delay this solution could work well. But for longer delays this would output an unwanted force of a magnitude large enough for the user to notice and also move the haptic device in an unwanted direction.

### 6.5.2 Idea B

This solution has the advantage that it cannot produce any incorrect forces. However, there is a risk that the force from the haptic device will feel choppy instead of smooth.

### 6.5.3 Idea C

This idea is probably the worst alternative. If we block the loop in order to be able to do a force update, then all of the other things the loop does, such as checking for movement and starting and stopping the force rendering would also be halted.



### 6.5.4 A fourth approach

A fourth solution was proposed by G. Q. Maguire Jr. Instead of computing only the force at the current position, the remote computer should also calculate forces for the locations near the current location and send these force values in advance. With this approach as long as the movements are within the cached area no delay for applying the force will occur. In order to maintain a good cache hit rate the computation of new nearby forces should continue when the position is changed, even though the current force is already known. This because we still need to expand the area of known forces in advance of movement to one of these locations. Rapid movements might still move the cursor outside the cached area, thus we still need a way to handle the case when no fresh force data is available in the cache.

### 6.5.5 Caching and problems due to caching

The computer with the haptic device needs code to convert between the haptic device coordinate system and the force matrix which is indexed with integers. This matrix of forces is needed in order to maintain the cache and to allow a simple index operation to find the force to be applied at the new location – rather than having to wait for this force to be calculated (thus decoupling the feedback servo loop from the computation of forces). To transform from the device coordinate system (in mm) to the pixel or voxel coordinate system we need to know the transformation between the different coordinate systems. The cost of performing the transformation from one coordinate system to another is roughly 16 multiply and 12 add operations.

However, the problem becomes much more complex when a non-static force algorithm is used. For example, the spring force we developed, see section 5.5, may not work well with a cache. This is because the force at one location can be different, depending on if and where the anchor of the spring is located. At the time of creation of the forces in neighborhood, initially no anchor may be set, but this area may be large enough to include a point where the anchor can be set. In this case the anchor would be set and a spring force should be rendered, but since the forces are already cached, the wrong force would be sent to the device.

Figure 37 shows a possible lapse. (1) Shows the system's state when the cache is created. In (2) the user moves the glyph over the black line, and an anchor is set at the red dot. In (3) the glyph is back to the same position as in (1), but since we now have an anchor set we also should have a force applied at this location. If we simply used the cached value that was stored earlier at this point when the system's state was as shown in (1), then we will not get the correct force.

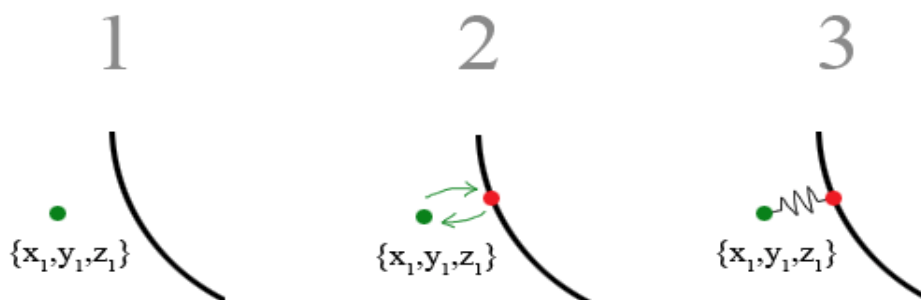


Figure 37: Possible problem with stale cached state

A possible solution for this could be to flush all the currently cached forces every time an anchor is set, moved, or released. The problem is that it would increase the delay from when the position is sent by the haptic device to the attached computer until the new force(s) are received by the computer attached to the haptic device. In order to discard the current cache at the computer attached directly to the haptic device we need to know the locations where the anchor would be set, moved, or released. If such a location is reached, then all the current forces would be discarded. However, it will now take a full roundtrip cycle of delay and the time to perform the new force calculations before any new forces could be in the cache.

Another possible solution to this problem would be to calculate all the different sets of forces in advance relative to all the different anchor positions within the selected area. The downside of this is that it would require more CPU time, bandwidth, and memory for storage. The required resources would scale linear with the number of possible anchor positions inside the current working area. The worst case scenario would be that every point inside the current working area is a possible anchor position. However, it should be kept in mind that each force consists of three double precision floating point values (i.e.,  $3 \times 8 = 24$  bytes). To put this in some perspective, if the working area were 10x10, then a complete set of forces for a single anchor point would be 2400 bytes (less than two full Ethernet frames worth of data); while if each of these points were a potential anchor point then it would take 100 times this much data (240000 bytes). It would take less than 14 milliseconds to transmit 200 packets using a 10 Mbps Ethernet and proportionally less time for faster links. As an optimization the cache could be refilled in advance, thus as the user moves the locator new values would be filled in primarily on the boundaries of the working area. The modeling of a sophisticated force caching scheme remains for future work.

### **6.5.6 A different approach**

What about local computation of the spring force and only send info about where to set and release the anchor instead? This could work great. All forces will be calculated locally and if you move outside the cached area, the only downside would be that you will not place your anchor for the time before you have received any new anchor position data. Since the force is computed locally and the release of the anchor depends of the length of the spring the force should work as intended.

### **6.5.7 How do we solve the problem of rapid movement of the locator**

Another problem that arose is how to handle rapid and long movements. At some point the movement of the location will be so fast that the glyph leaves the area for which the requested cache information has not arrived (in fact the computer doing the force calculations might not even have received the required for these forces yet). This can lead to unnecessary computation and the large amounts of data that has been required (only to be thrown away) may clog the network connection. Possible solutions include reducing the working area of cached points to avoid this becoming a problem; or if the movement speed exceeds some threshold, to disable caching. We have not calculated the tradeoffs between movement speeds and the amount of unnecessary computation and wasted/used bandwidth. This too remains a topic for future work.

### 6.5.8 Conclusion

Our conclusion is that a cache system (particularly for dynamic forces) is significantly more complex than for the static case; and involves many design decisions that depend on what type of force is being used. Hence we have chosen not to pursue this matter any further in this thesis project, but suggest that it is a ripe area for future work.

## 6.6 Network tests: Description of the test setup

We chose to use proposed method 3 as described in section 6.4 for our network tests. Therefore our network test setup consisted of the components in the following list:

- Two computers, where the first computer had:
  - An OpenDX program running with our test network and the 256\*256 hip phantom image loaded.
  - A haptic device connected it IEEE 1394a interface to a IEEE 1394a/b interface on the computer. (Note: no other IEEE 1394 devices were connected to this computer's interface.)
  - A network socket opened for sending/receiving UDP packets to the second computer by code included in the HapticsModule.
- The second computer had:
  - Our server bouncer program running. This application takes UDP packets sent from the first computer and returns them.
  - TC running to be able to shape the network traffic with regards to packet loss and delay.
- One test user was sitting at the first computer using the haptic device to interact with the OpenDX program.
- One operator that was sitting at the second computer. The operator altered the packet loss and delay parameters and record the user's orally stated perceived quality from each test run.

The two computers were connected through an Ethernet switch. We did some ping tests to check out the delay and packet loss between the computers and the result was that the delay varied between 0.1 and 0.2 milliseconds while we did not get any lost packets at all. Since the test delay increase steps were 5 milliseconds we think that was a negligible amount of delay that should have no impact on our test data validity. Additionally, the fluctuating delay caused by the scheduling of the tc process was between 4 and 8 milliseconds, and that further diminishes the influence of the innate delay between the computers.

## 6.7 Network tests: How was the testing done?

After setting up our testing system we started our network tests. The data that we wanted to collect was the user's perceived quality while increasing either delay or packet loss. The quality scale went from zero to four where zero was no loss of quality and four was a totally unusable system. A comprehensive description of the different quality steps in the scale can

be seen in section 7.2.1. Each user was told about the quality scale and given a chance to ask questions about it before testing began.

We first let the test user play with the haptic device and the image for a while without any artificially increased delay or packet loss. When the test user stated that they understood how the system felt to work with and how might help with the task of outlining a contour, we started the test. We gradually increased the delay in steps of 5 ms and ask the user to grade their perceived quality on our zero to four scale for each step. There was not a time limit set for the user for each step, thus the user could mark as many points on the contour as they wanted. We continued to increasing the delay until the user deemed the system was unusable, i.e. a grade of four. After this point was reached, we set the delay to 0 ms and instead increased the packet loss by 5% steps, until the user once more felt the system was unusable.

In these first two tests the data we sent to the second computer and back (i.e., reflected or bounced) was only the anchor positions. The forces were still computed locally at the first computer. Thus the worst case scenario for increased delay was when the anchor position arrived too late, thus the user had already moved far from that position and the release threshold was reached and disabled the force right after setting the anchor. For increased packet loss, the worst case scenario was that an anchor position was lost and no force was being set at all. Due to the similar nature of these scenarios, they were both expected to yield similar results for a user; for example, less smoother movement when trying to follow a line of interest.

In the second pair of tests we tried a different approach. Instead of just sending the anchor positions we sent the full force vector to the second computer and bounced it back. We did the same tests as earlier, i.e. first increasing delay until the user deemed the system unusable, then tests with increasing packet loss instead of delay. The difference between this method and the first was that when increasing delay the forces got back later than they should, with regards to where the user had moved the glyph on the monitor, whereas when increasing packet loss some of the forces were lost - quite similar to the first packet loss test but with the forces rather than just the anchor points.

Our data was collected in a spreadsheet. Using this collected date we drew a number of graphs showing how the different users graded the quality of the system for the different test parameters. An analysis of these data sets is given in section 7.2.1.

## **6.8 Network tests: Evaluation**

Our tests concerning the networked haptics part went well. All test users completed the tests and one of them even did the tests more than once. The results of the tests varied quite a bit between both different users and also between different testing rounds by the user who did the tests more than once. In that case, the user tended to be pickier from test to test, i.e. deemed the system unusable at a lower delay or packet loss than in the earlier tests.

With the anchor sent and increased delay some points were missed and there was sometimes a jerky movement if the user moved the glyph too fast. The same occurred when we increased the packet loss instead. Rapid movement of the input locator caused irregular movement and lead to a longer distance between anchor points, which for the user feels less smooth.

In contrast, when we sent the forces to the other computer and increased delay the perceived quality from the users went to unusable faster than in the first test, whereas when we sent the forces and increased packet loss, the perceived quality decrease was quite similar to the first packet loss test. More on the analysis of this test will be presented in section 7.2.

## Chapter 7 - Analysis

This chapter presents the analysis of the results of our tests and implementations.

### 7.1 Forces

We decided to use a system that looks for places in the image to set anchors instead of having fixed force values for each point of the image. When an anchor is set the system starts to decide how to render the force.

The current system creates a matrix of gradients for each pixel in the image. When the user moves the glyph around the image the gradient for each point it passes by is checked. If the gradient is larger than the threshold value, then an anchor is set and the system starts to render a force.

As discussed in section 8.3.1 on page 79 the current usage of gradients may not be the best way to decide which points should be anchor points. However, this was the method used for all of our testing.

#### 7.1.1 Anchor points in different images

This section will show the anchor points in three different types of images. There are two different sets of parameters for each image:

- **Level** and **width** - these levels determine how the image is displayed on screen.
- The **snap threshold** sets the value that the gradient has to exceed in order for an anchor to be set.

We used two different values for “level and width” on the first image. The lower values are a simple (blind) mapping of the pixel values to the displayed intensity and color. The larger values for level and width are representative of how a user would prefer to work with the image. The reason for showing this is that what the application reads in and computes the gradients from are directly reflected in the actual pixel values (visually these correspond to the lower values for window width and level). When the user changes these settings there might be a difference between the displayed image and the actual positions of anchor points. Because of this difference, at some places the anchor points seem to be missing. While in other places you get stuck (i.e., it is hard to move), even though it seems like there is nothing there. Again this is a problem concerning *how to calculate the anchor points*, rather than with the force algorithm.

The snap threshold was set to two different values. 0.5 is the value that worked best for our testing. While the value 0.3 was used to illustrate how many more bad points became anchor points at this lowered value. The light grey rows contain snap threshold values of 0.3.

**Table 2: Description of images**

Image	Figure	Size	Slice	Window level	Window width	Snap threshold	Number of points
Hip phantom	Figure 38	256*256	130	500	500	0.3	1673
Hip phantom	Figure 39	256*256	130	500	500	0.5	698
Hip phantom	Figure 40	256*256	130	3300	500	0.5	698
Hip phantom (zoomed in)	Figure 41	256*256	130	3300	500	0.3	1673
Hip phantom (zoomed in)	Figure 42	256*256	130	3300	500	0.5	698
Abdomen	Appendix A: Figure 64	512*512	50	1150	350	0.3	8442
Abdomen (zoomed in)	Appendix A: Figure 65	512*512	50	1150	350	0.3	8442
Abdomen	Appendix A: Figure 66	512*512	50	1150	350	0.5	3883
Abdomen (zoomed in)	Appendix A: Figure 67	512*512	50	1150	350	0.5	3883
Hip patient	Appendix A: Figure 68	512*512	110	1100	600	0.3	928
Hip patient (zoomed in)	Appendix A: Figure 69	512*512	110	1100	600	0.3	928
Hip patient	Appendix A: Figure 70	512*512	110	1100	600	0.5	437
Hip patient (zoomed in)	Appendix A: Figure 71	512*512	110	1100	600	0.5	437

Figure 38 shows that an image with a threshold of 0.3 results in a lot of unwanted anchor points. Each anchor point is shown as a red colored square.

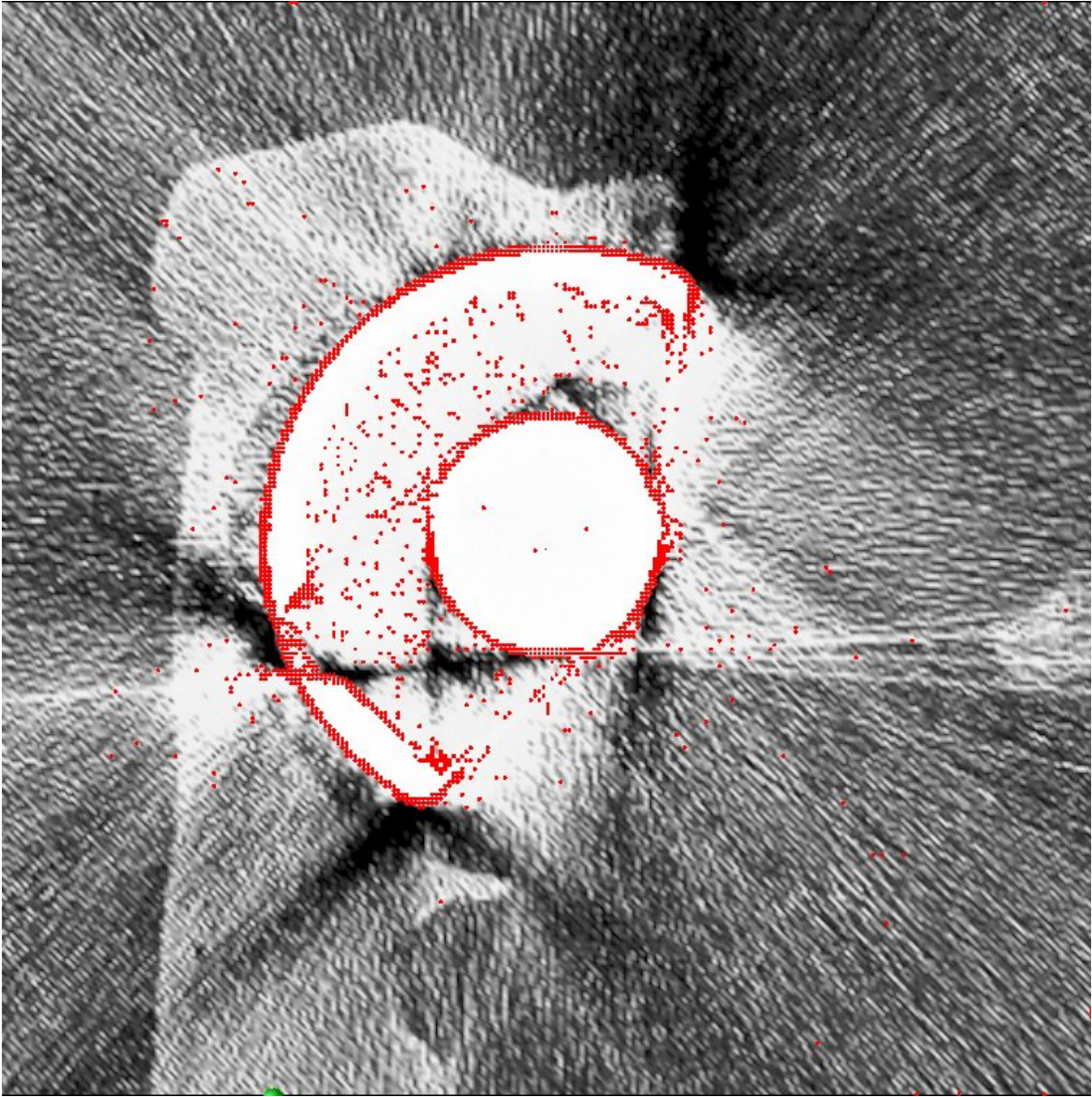


Figure 38: Hip phantom with a threshold value of 0.3



In comparison, Figure 39 has the settings that represent the actual image used when working. In this case, there seems to be some points missing. We can see gaps on the left side of the circle in the middle and on the upper side of the shape to the bottom left.

At first this seems strange, but when you change the settings to show the data with a low window level then the problem reveals itself. This is shown in Figure 40. Where Figure 39 seemed to be fine, edges now are white areas. This occurs because the computation of gradients does not produce a correct value at these positions. This is a problem that needs to be solved in order for the forces to work better. This the computation should consider the window width and level when computing the gradient values.

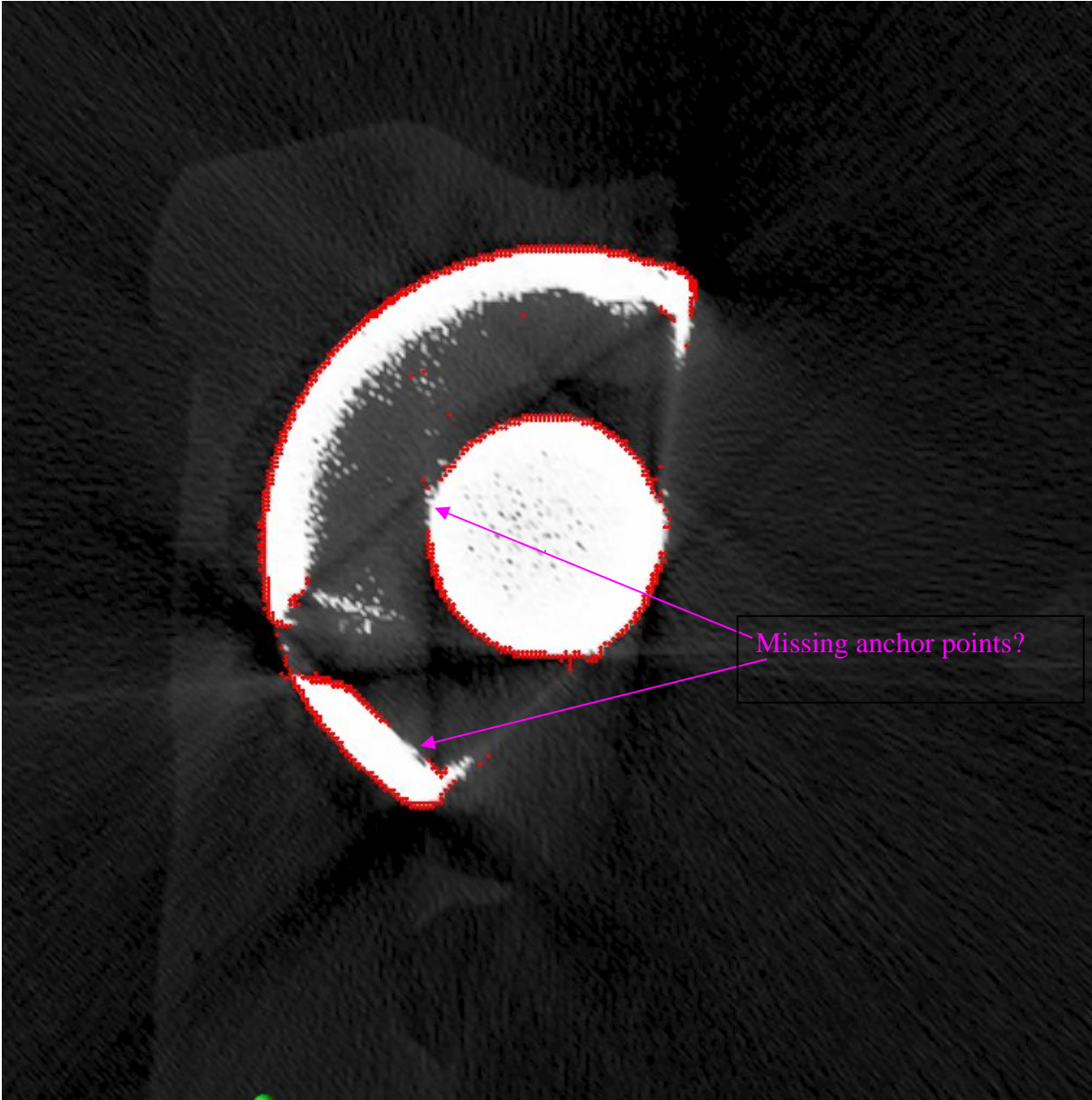
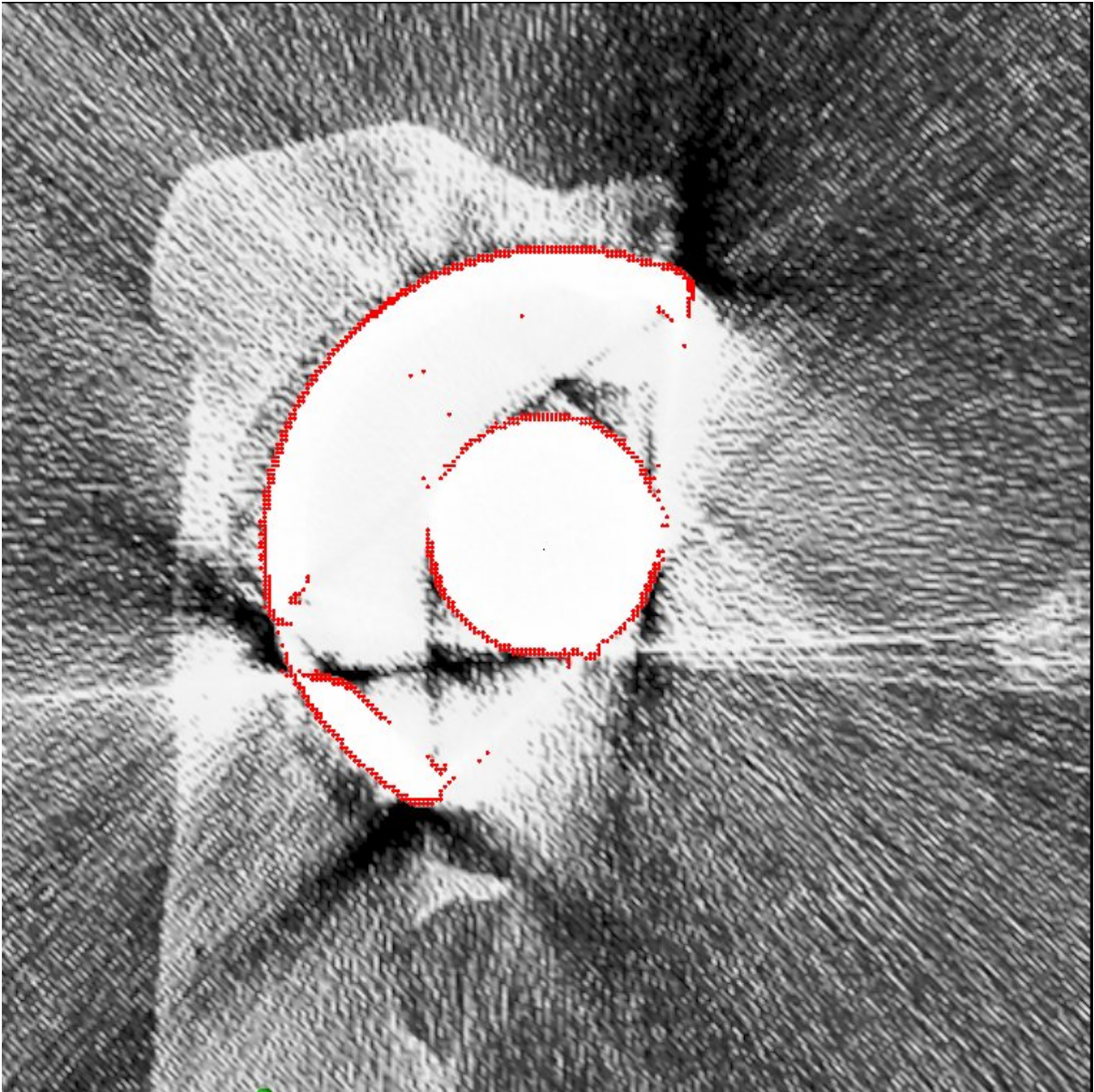
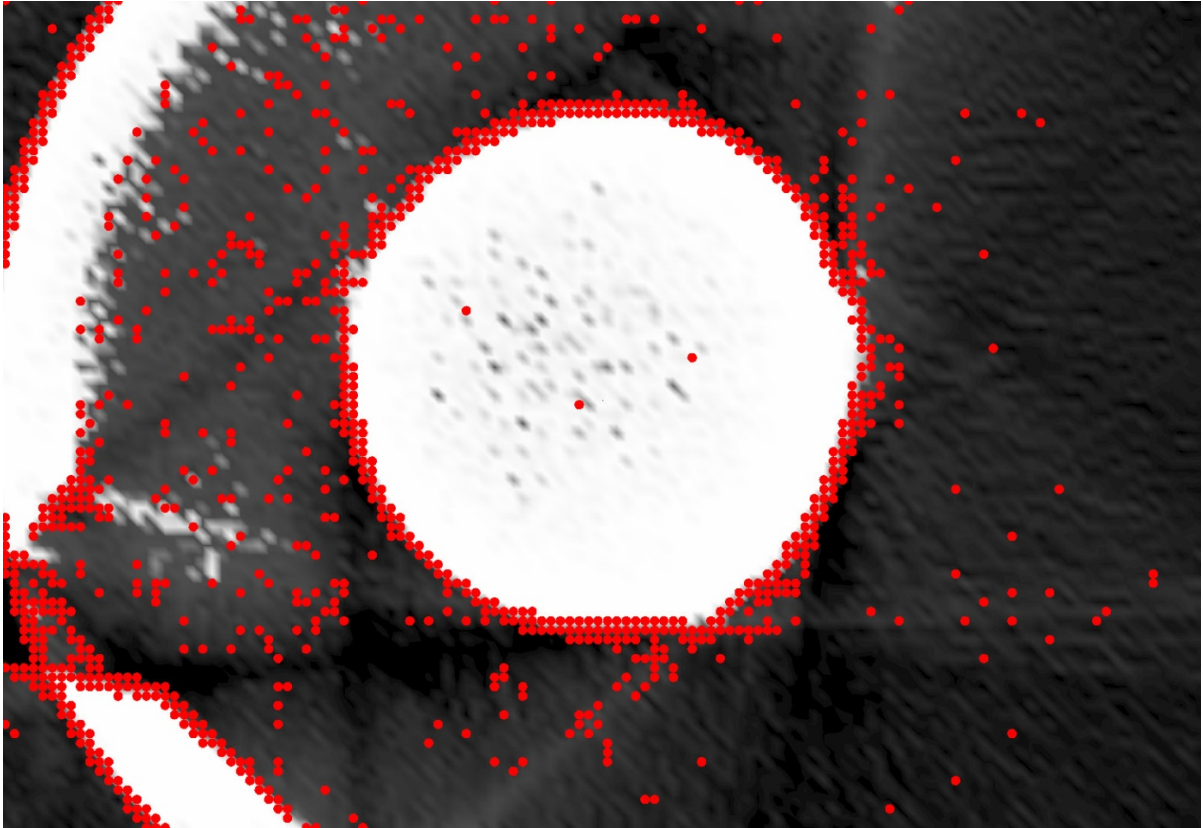


Figure 39: Hip phantom with a threshold value of 0.5



**Figure 40: Hip phantom with a threshold value of 0.5**

Figure 41 and Figure 42 show the 256 phantom hip image in a zoomed-in mode in order to show the anchor points close up.



**Figure 41: Zoomed in hip phantom with a threshold value of 0.3**



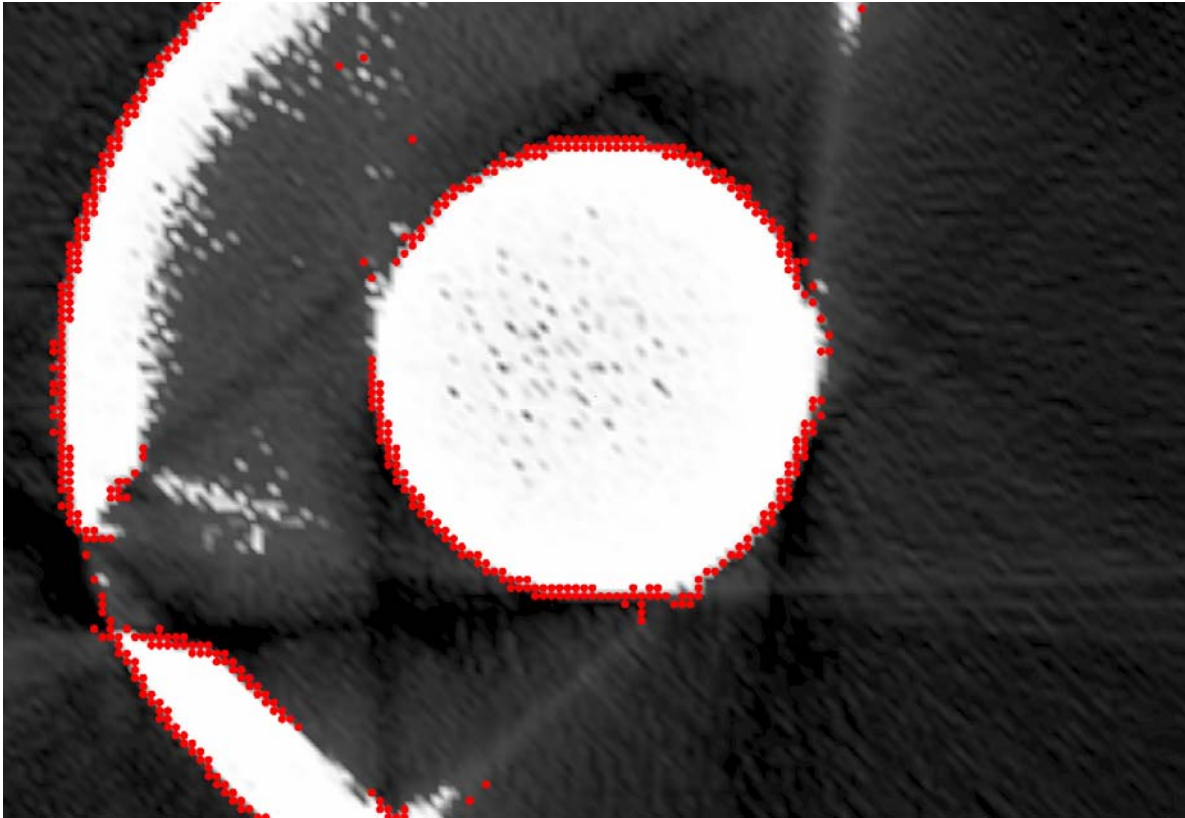


Figure 42: Zoomed in hip phantom with a threshold value of 0.5

### 7.1.2 Summary of force analysis

We started out with two different goals:

1. List and discuss different types of forces.
2. Implement and evaluate the suggested forces.

Goal one was achieved in Chapter 5 - where we listed and discuss a number of different possible forces. Out of the five possible forces presented, we decided to implement two, the spring force (described in section 5.2.3) and the magnetic force (section 5.2.2). In addition to ourselves we have had five other persons using the forces while doing network delay testing (for a total of 7 test users). The results of these network tests are given in section 7.2.1.

Before doing the network tests we also asked the users what they thought about the two different forces we had implemented. All users were very positive to the functionality of both the magnetic and the spring force. They also gave us two types of negative feedback:

1. **Movement along lines.** Some of our test users noticed the problem with a not completely smooth movement along lines as discussed I section 8.3.2.
2. **Faulty or missing anchor points.** As described in sections 7.1.1 and 8.3.1, some users noticed that there were anchor points missing at some places or present at places where they should not be.

Even though these two problems still remain, we feel that our second goal concerning forces was met. We successfully construct a new system to decide where forces should be applied, as well as two very well functioning types of forces.

## 7.2 Networked haptics

In this section we describe the tests and our analysis concerning the networked haptics part of this thesis project.

### 7.2.1 Delay and packet loss testing

In order to find out how much delay or packet loss the system can tolerate before being perceived by the user as useless we did some testing. The test consisted of giving a test subject a single task: try to follow and outline a shape in a medical image. The image chosen was the phantom hip in Figure 39.

The user got to test the system for a while without any delay. After each test run we increased the delay or the packet loss and ask the user to evaluate the quality of the haptic performance on a five grade scale. These five grades were:

0. No loss of quality, the system feels as responsive as when no delay was present.
1. A small change in quality is noticed but the system still works well
2. The user can now really feel that the system is affected but can still complete the task without problem
3. The quality is beginning to get really bad and the user may have a slight difficulty in completing the task
4. The system is totally useless

In order to plot graphs with decreasing values indicating worse quality (in order to be comparable to Cole and Rosenbluth's model as shown in Figure 11 on page 16) we decided to reverse the scale. Therefore, in the graphs in this chapter a 4 represents the best quality and 0 represents a totally useless system.

We had two different implementations of the network part of the code as described in section 6.7. Therefore every tester had to do the tests twice, once for each version. One user did some of the tests several times and these tests have their own graphs.

As described in section 2.6.1 we cannot add an exact amount of extra delay due to the functionality of the operating system. On our test system we had between 4 and 8 milliseconds extra delay. Thus if we told Traffic Control to add 1 ms delay, the actual delays varied between 5 and 9 ms. Therefore we will add 6 ms to the delay values shown in our results – in order to show the actual delay introduced. This is the reason why the values in the graphs are 0, 7, 11, 16, 21, ... ms instead of 0, 1, 5, 10, 15, ... ms.

What do these tests tell us? When we started working on this thesis we were not sure how the haptic system would behave over a network link. We did not know in what way the user experience was going to change with worsening conditions over the network link (in terms of higher delay and packet loss).

First a note about the user that did the tests several times, the results got worse for each repetition of the test. This is probably due to the fact that the user became more experienced with the system and therefore changed her view of how to apply the quality scale. It would have been better if all users would have had the possibility to do the tests several times, but unfortunately this was not the case.

## 7.2.2 Test results

Figure 44 and Figure 46 shows the results of the delay testing with the two different implementations. The figures show us that the quality loss seems to be falling linear with increased delay.

A thing that is noticeable is that most users gave the anchor implementation (Figure 44) several scores of **1** and **0.5** before they deemed the system totally useless. The anchor version of the system also maintained a higher quality for quite a bit longer when compared to the force version, see Figure 51. But since these large differences only occur when the quality is lower than **1** on the quality scale – this corresponds to a almost useless system; hence our conclusion is that we can **not** say that one of the two implementations works better than the other.

*User 3*'s scores differ quite a bit from the others in some of the tests. We are unsure if this is due to technical reasons during the tests or if this user just had much higher demands on the system's functionality. A removal of *user 3* changes the anchor system from being worse than the force to being slightly better. Due to *user 3*'s difference in scores from the other users, we decided to include a graph comparing the anchor and force test results without *user 3*'s results. See Figure 52. The case with *user 3* also illustrates that it is not a simple task to create a quality scale that users can apply easy and consistently. Every user has their own experiences, thresholds, and expectations of the system's performance.

As for the system's total tolerance against delay over a network link we found the results a bit disappointing. Quality **2** which represents a still well usable system was reached at as little as 30 milliseconds of delay. Quality **1** was reached around 40 milliseconds, a bit higher for the anchor system at around 45 milliseconds. Our results clearly show that some kind of caching has to be done in order to get this haptic system working with a satisfying quality over network links where high delay might be present.

The packet loss results were much better. The system shows a greater tolerance towards packet loss than delay. We still have what looks like a linear relationship between increased packet loss and falling quality. See Figure 50 for the anchor version and Figure 48 for the force version. We still see quite different results from *user 3* in these tests too, and therefore we decided to do two different comparison graphs, again one with *user 3* and one without.

The system can tolerate somewhere around 25-30 percent packet loss before it reaches a quality of **2** and up to around 45 percent packet loss before the quality is down to **1** (i.e., unusable). We think that this is impressive since a network with 25 percent or more packet loss probably would be seen by most used as very bad.

The greater tolerance against packet loss than delay is explained by the way the system works. A lost packet results in either a missed placement of an anchor (in the anchor version) or a momentarily zero force (in the force version). But since both the detection of an anchor point and the force update is done every callback loop, which runs a thousand times every second, even quite high losses still do not affect the functionality to a point where the user finds it irritating. Note that even at a 45% packet loss rate, if we assume that these losses are roughly uniformly distributed, the user is still getting a correct update at about 450 Hz or once every 2ms! The delay on the other hand still causes all the force output to happen, but too late. This behavior seemed to be noticed very quickly by the test users.

It is interesting to note that *user 3* was less tolerate of delay that the other users in the case of delays with the anchor, while more tolerate than the other users in the case of delays with the forces. The same user was more sensitive to packet loss in the case of forces that in the case of anchors; but consistently much more sensitive to packet loss than the other users.

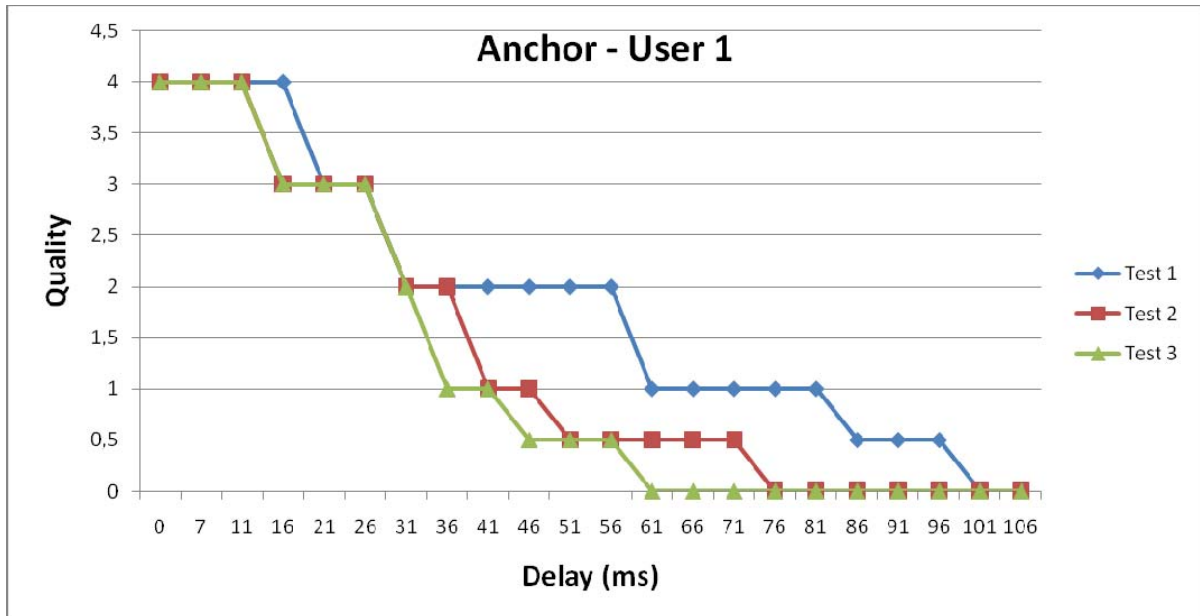


Figure 43: Anchor version – User 1 – Delay

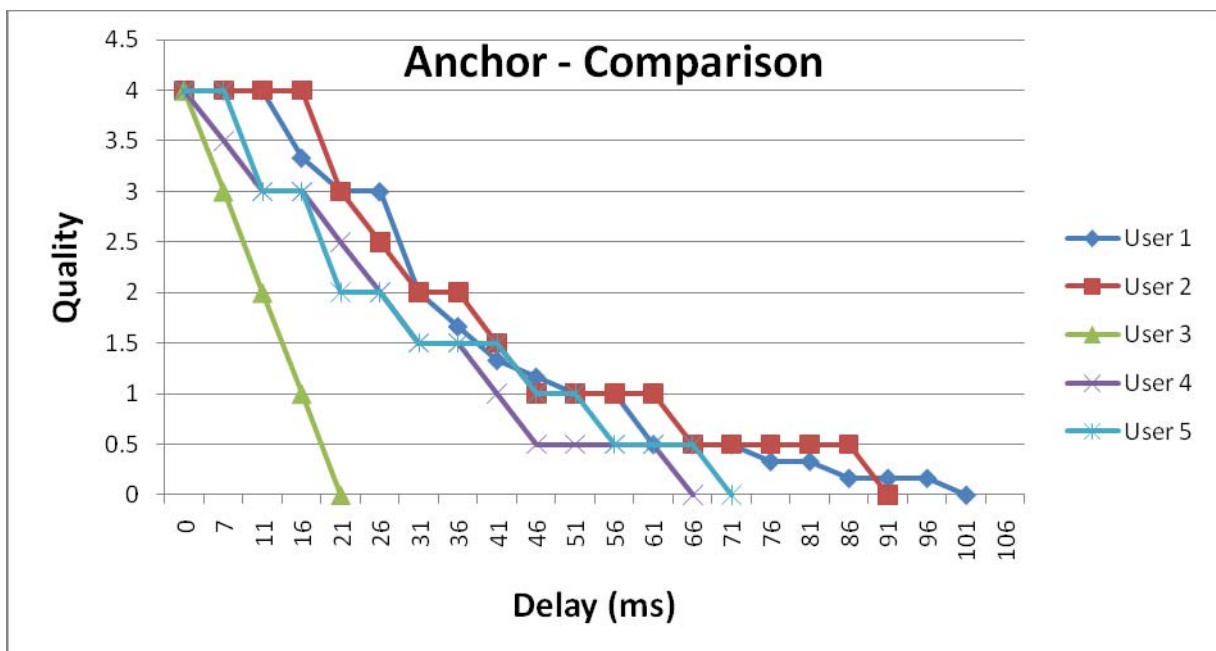


Figure 44: Anchor version – All users comparison – Delay

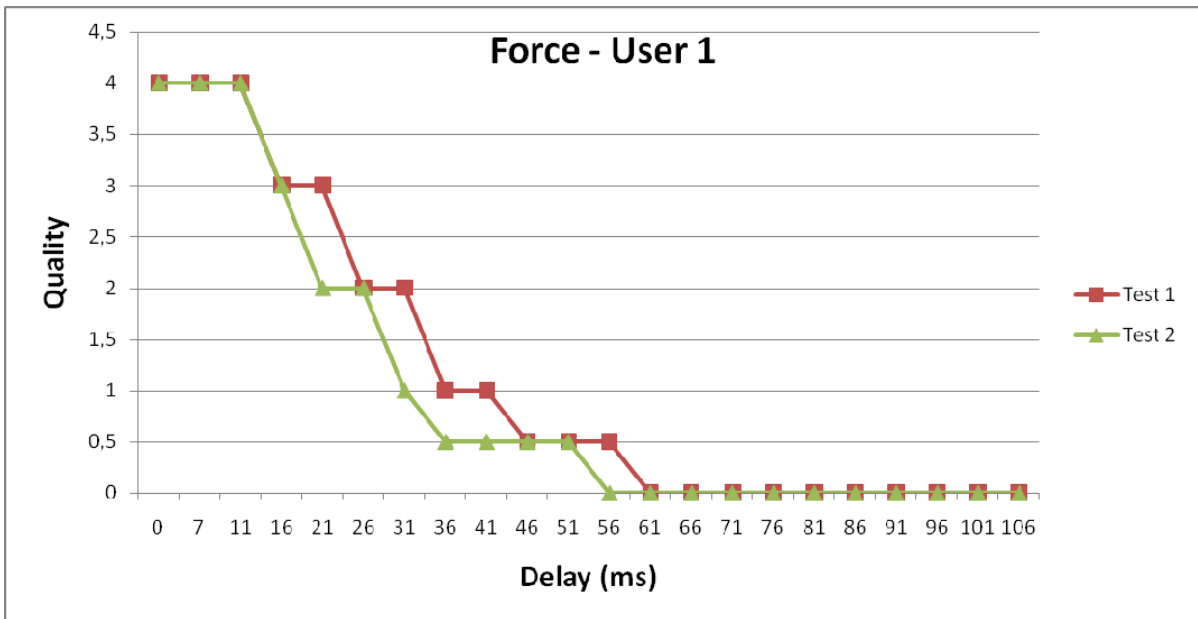


Figure 45: Force version – User 1 – Delay

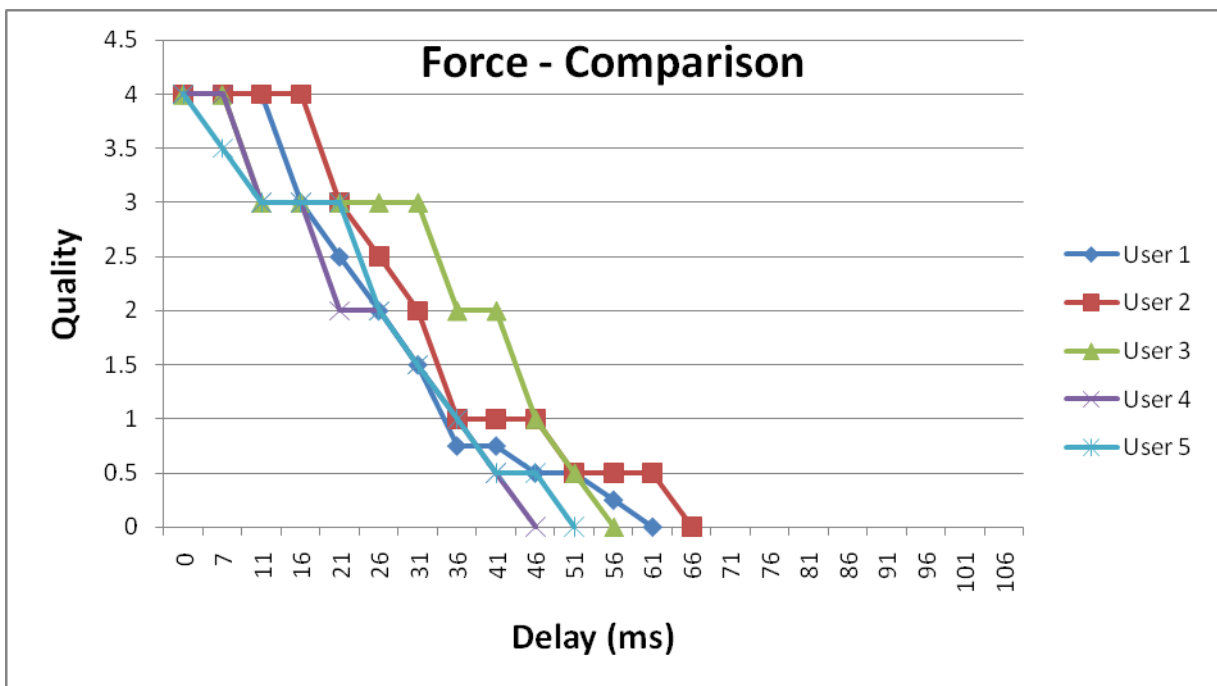


Figure 46: Force version – All users comparison – Delay



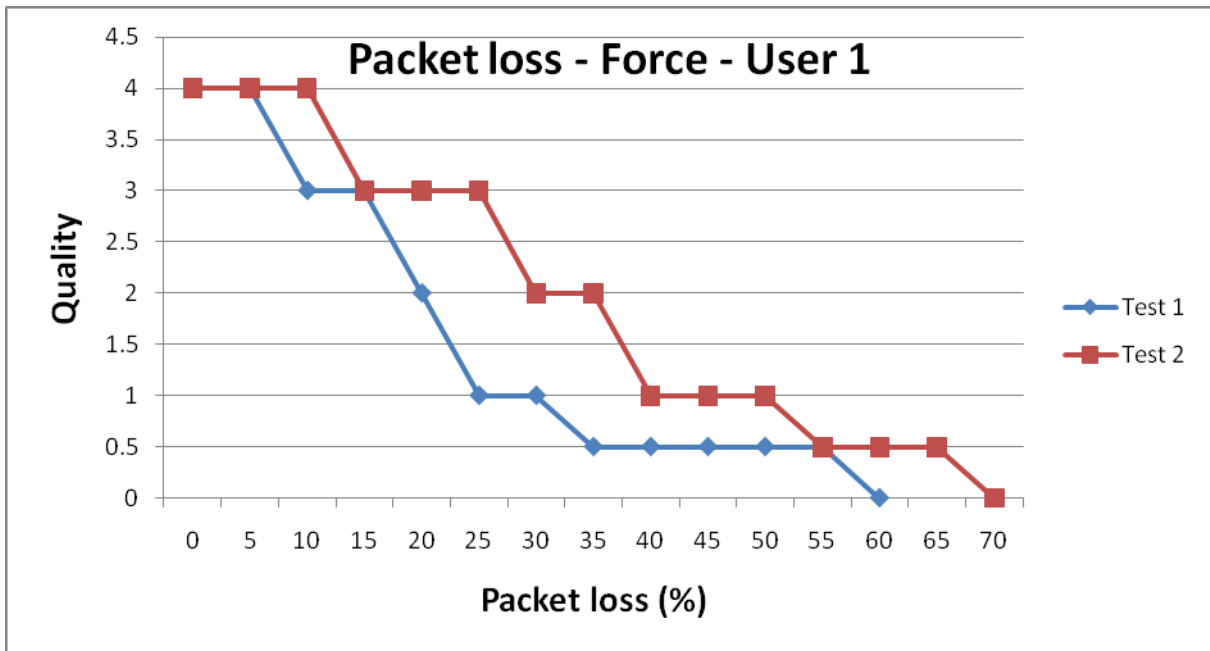


Figure 47: Force version – User 1 – Packet loss

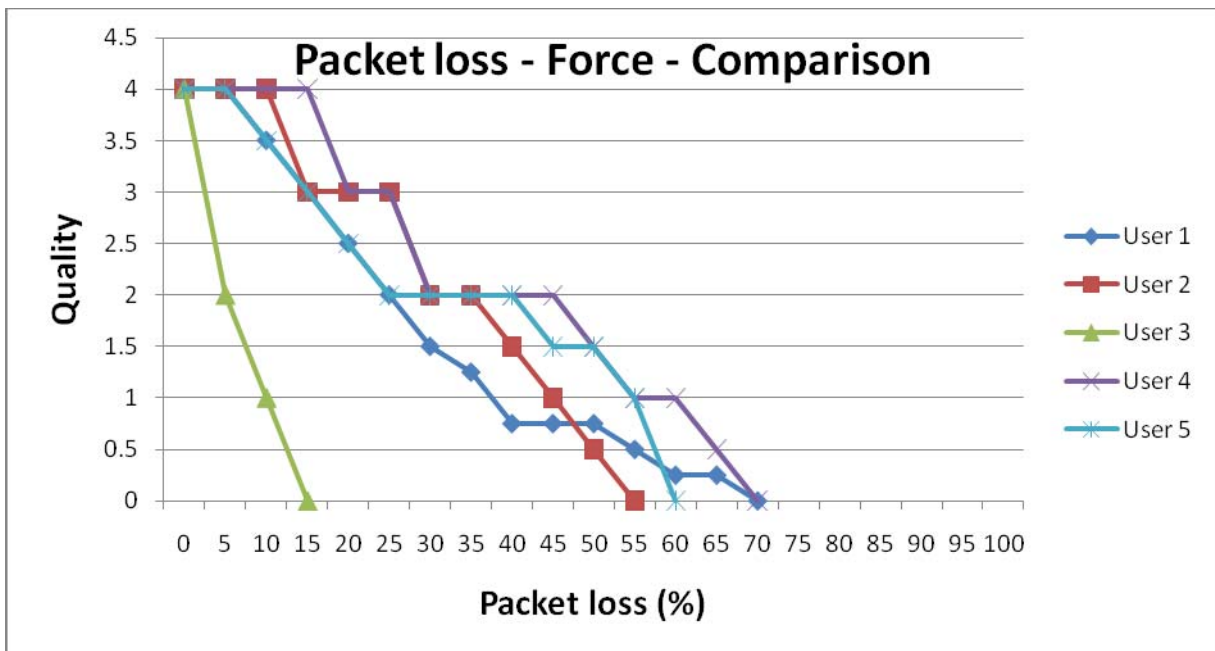


Figure 48: Force version – Comparison – Packet loss

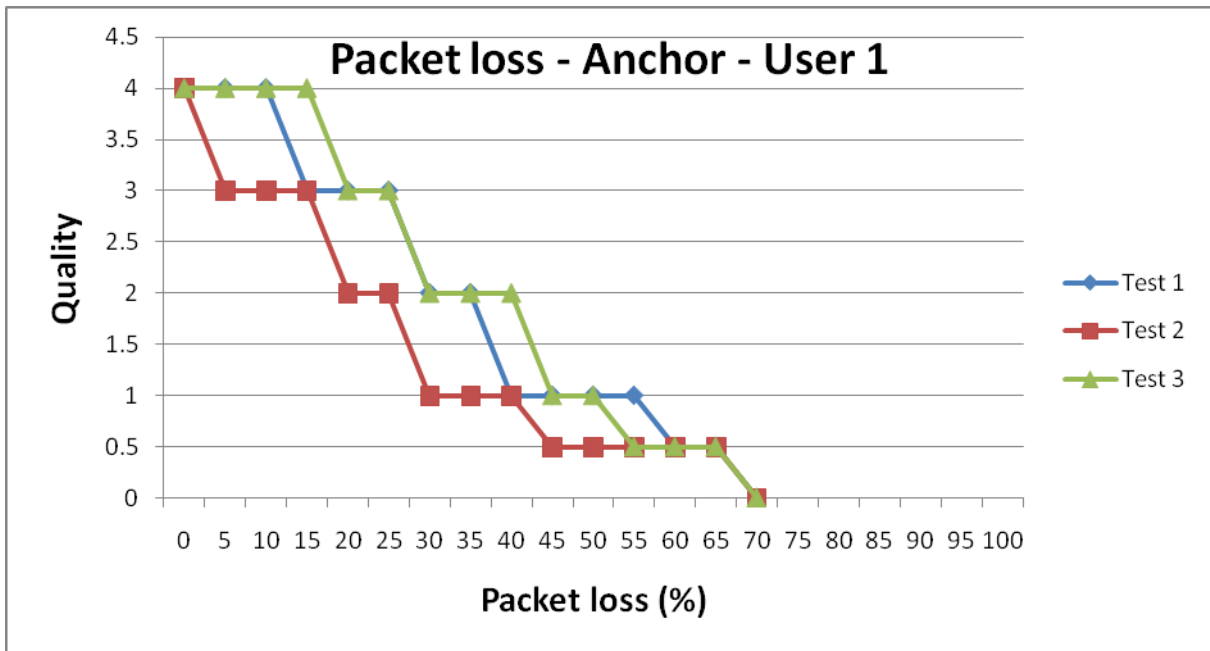


Figure 49: Anchor version – User 1 – Packet loss

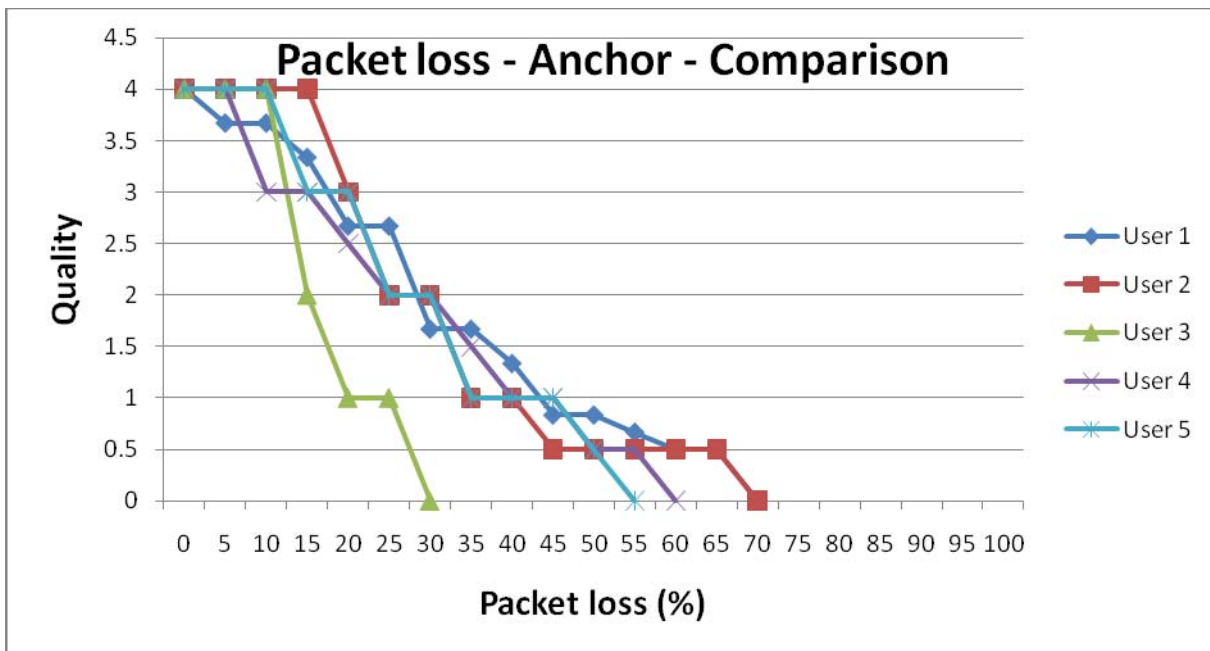


Figure 50: Anchor version – Comparison – Packet loss

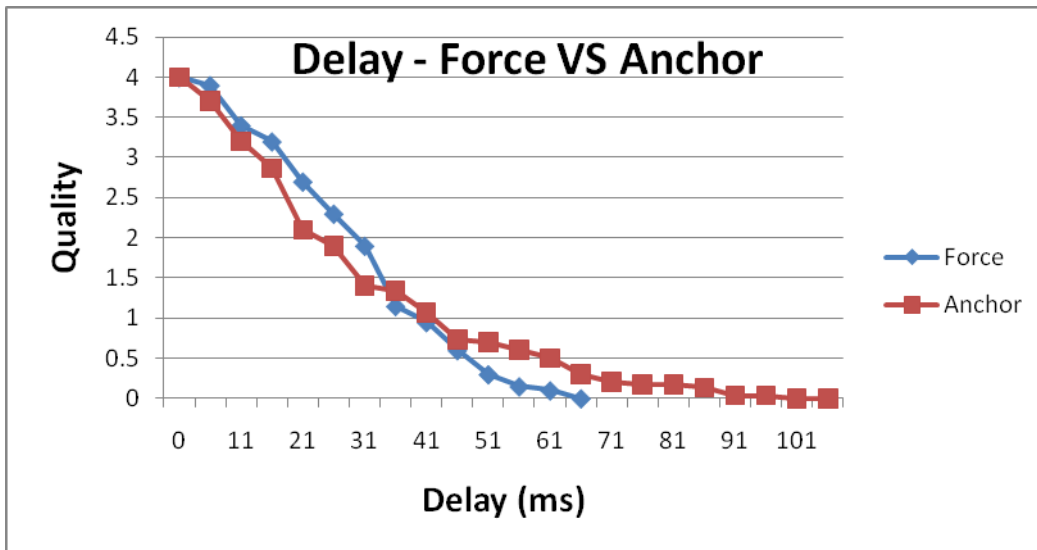


Figure 51: Force VS anchor – Delay

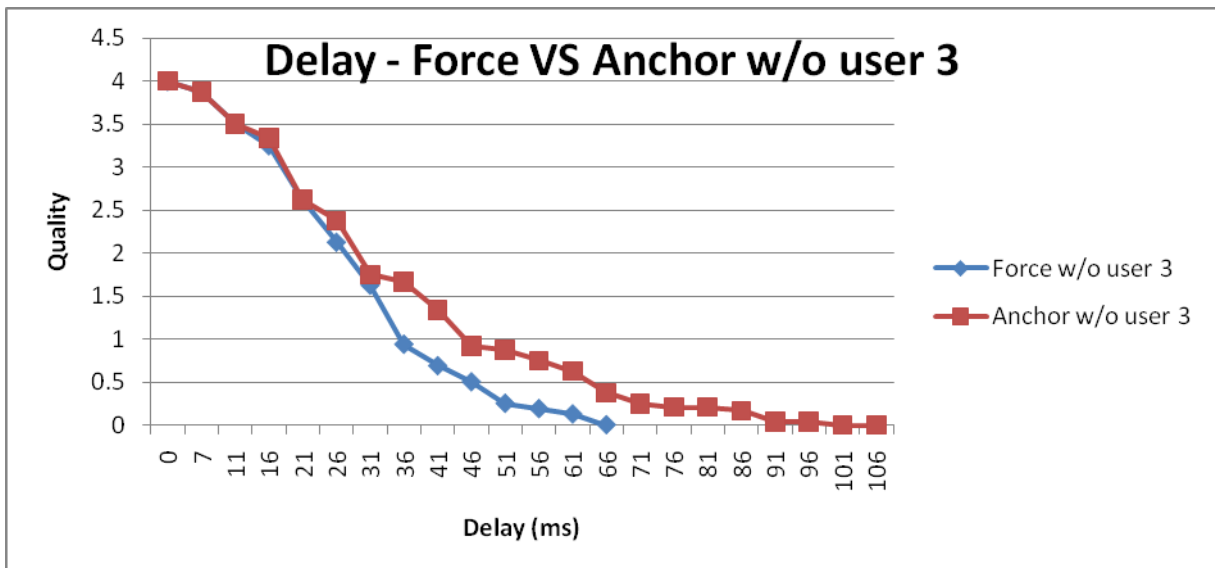


Figure 52: Force VS anchor without user 3 – Delay

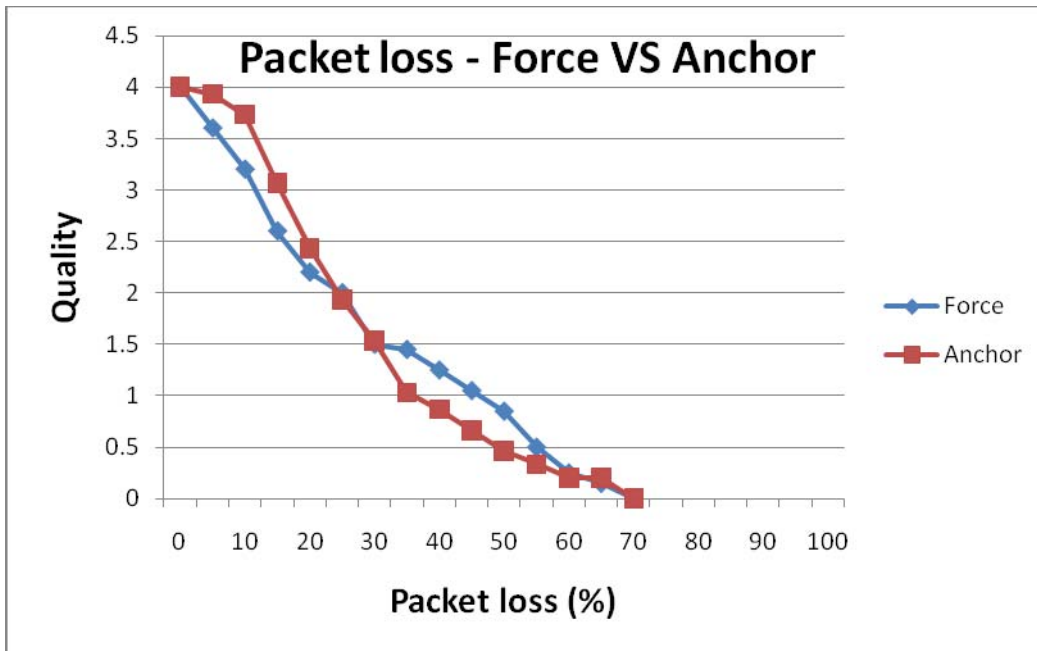


Figure 53: Force VS anchor – Packet loss

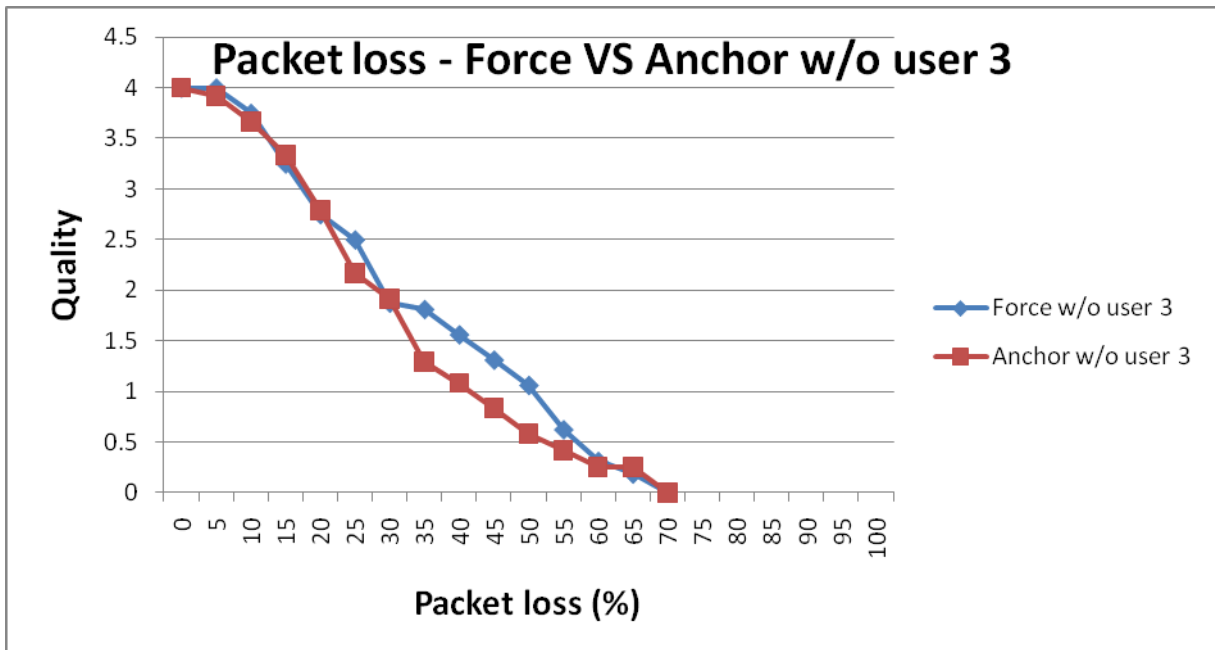


Figure 54: Force VS anchor without user 3 – Packet loss

### 7.2.3 Network caching

A caching system was mentioned in section 6.5.4 as a possible approach to mitigate the impact of delay while working with a networked haptics system. We did not have time to *implement* such a caching system, but we collected data, evaluated it, and analyzed it with the intention to find a formula giving the probability of a cache hit as a function of bandwidth, number of data points, and time.

In order to do some calculations concerning the potential effect of caching, we collected data from four different images, by using the OpenDX network to count anchor points and generate log files. We collected data from 100 slices per image. We did this twice per image; once with a 0.3 gradient threshold and once with a 0.5 gradient threshold. The number of anchor points for the different images and parameters can be seen in Table 3.

**Table 3: Number of anchor points in different images and with different thresholds**

<b>Image</b>	<b>Size</b>	<b>Threshold</b>	<b>Min points</b>	<b>Median</b>	<b>Mean</b>	<b>Max points</b>
Phantom hip	256*256	0.3	966	1530	1528	2736
Phantom hip	512*512	0.3	4820	14260	14020	23250
Hip patient	512*512	0.3	620	1058	1807	10130
Abdomen	512*512	0.3	49	8167	8922	23860
Phantom hip	256*256	0.5	575	707	721	956
Phantom hip	512*512	0.5	1229	2404	2398	4276
Hip patient	512*512	0.5	96	336	448	2296
Abdomen	512*512	0.5	24	3808	3835	13610

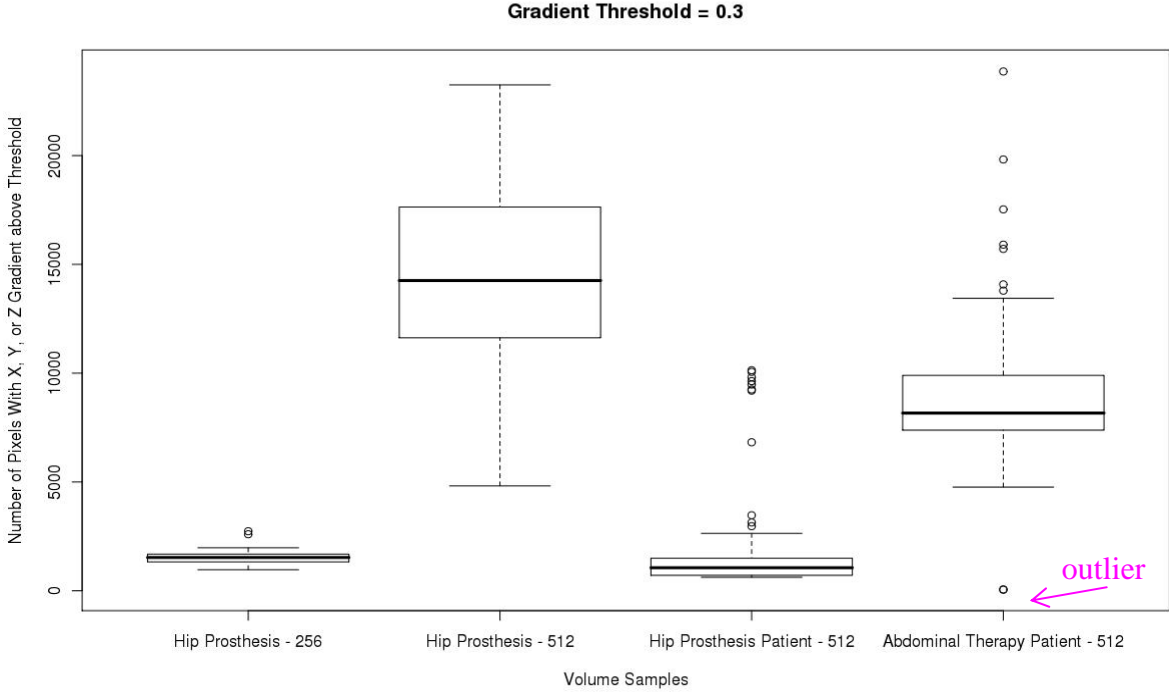
Some inconsistencies can be noted in the table, for example the minimum number of anchor points in a slice for the abdomen image is 49 and 24 for the 0.3 and 0.5 gradient thresholds respectively. After we examined this more we realized it was because those slices with such an exceptionally low number of anchor points had radiation therapy markers in them. Such markers are very bright and have a pixel intensity value that is very high. Since the gradients for each slice are normalized with regards to the highest pixel intensity value in the slice, all other gradients apart from the ones at the marker positions were scaled down much lower than in the other slices. The result was that the number of gradients that were above the threshold was much lower than in slices without these radiation therapy markers. One possible solution to this problem is to **not** normalize the gradients and instead use the original gradient values when comparing to the threshold. The reason for normalizing the gradients in the first place, was to make sure that the forces were not too high in the original system. However, since we did not use the gradients as forces in our system, we should not need to normalize them. However, we did not have time to further evaluate this idea in our project.

Another discrepancy we noticed in the data logs was that if we took a subset of the 100 slices and collected data only from this subset, the first and/or last slices' values differ from the values in the full data set. We realized that this is due to the way the gradients were calculated. Since for each pixel the gradient is computed based upon the 26 surrounding voxels' intensity values in three dimensions, hence have an edge effect where the gradient values are computed as if there is a "ghost" slice before and after the slice data set where all

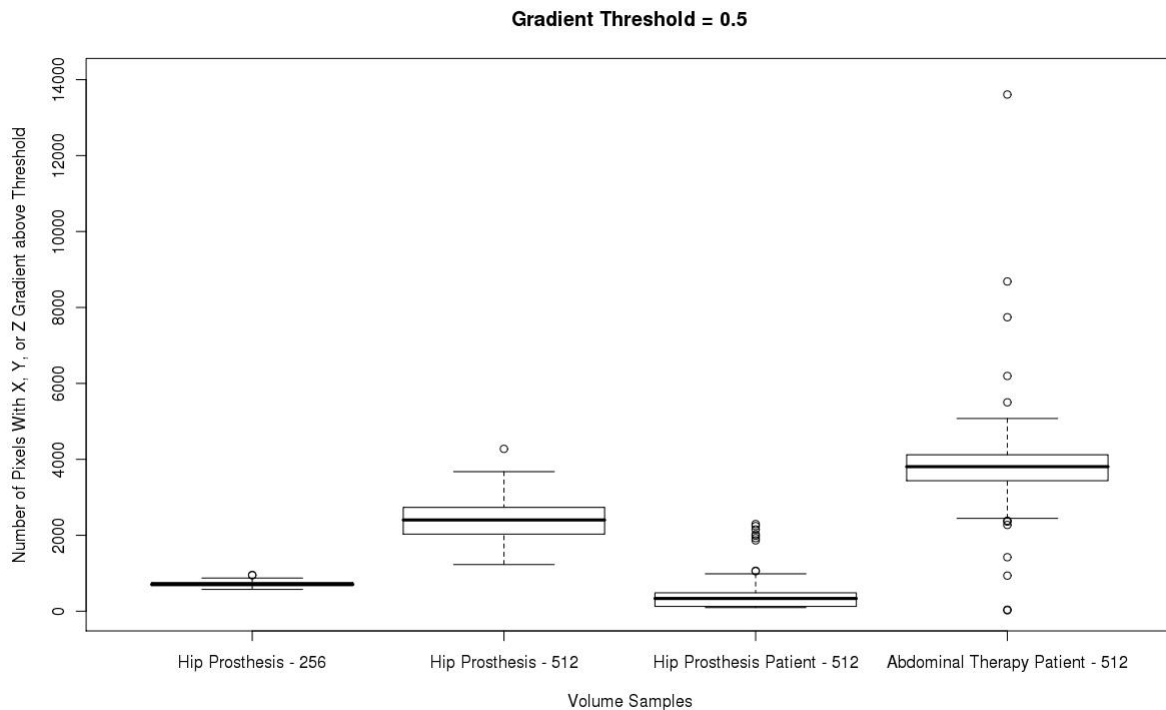
intensity values are zero. This is what makes the first and last slices' gradients different from what they should be. Our solution to this problem was to remove the first and last slices' values from our collected data set.

After we completed our data collection and made the above adjustments to our collected data set, we created box-plot graphs using this data. Figure 55 to Figure 57 shows the box-plots that we drew. Note that the columns are in the same order as the rows in each half of Table 3; thus the first two columns are for a phantom and the last two columns for patients.

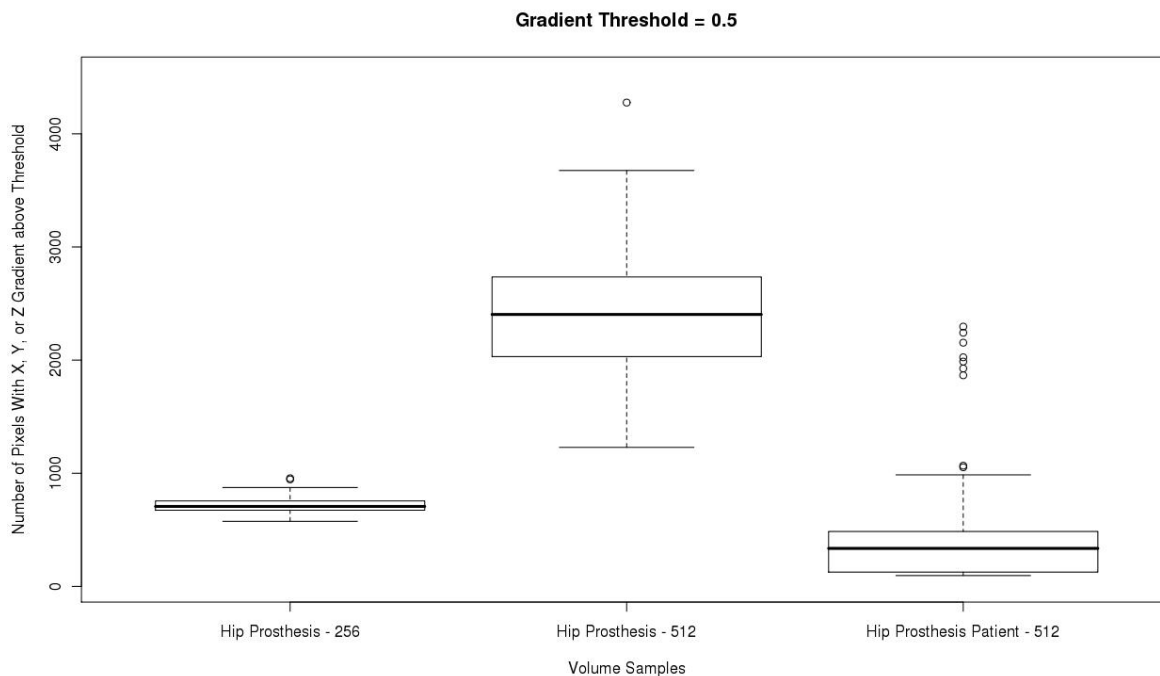
A box-plot is a graph where you can see the minimum and maximum values, in addition to the first quartile, the median, and the third quartile, i.e. you can see at what points there are 25%, 50%, and 75% collected values. It also shows possible outliers, i.e. observations that are numerically distant from the rest of the data. An example of an outlier is in the abdomen data set. For the 0.3 threshold abdomen data set the minimum anchor points was 49 as can be seen in Table 3; while the median was more than 8000. The result of this can be seen in our box-plots as an outlier at the bottom of Figure 55.



**Figure 55: Box-plot of the four images with 0.3 gradient threshold**



**Figure 56: Box-plot of the four images with 0.5 gradient threshold**



**Figure 57: Box-plot of three images with 0.5 gradient threshold**

The difference between Figure 56 and Figure 57 are that we removed the abdominal therapy patient from the box-plot to make it a little easier to see the quartiles and medians of the other three box-plots. As a result we can see that the image with the hip prosthesis in an actual patient at 512\*512 resolution had a lower first quartile, median, and third quartile than

an image of a hip prosthesis in a phantom at the same resolution. This suggests that evaluating algorithms using these anchor positions using such a hip phantom will have different results than evaluating the same algorithms with patient data. Thus we should note that our network tests used a test image consisting of the hip phantom data at 256\*256 resolution.

Our network tests in sections 7.2.1 and 7.2.2 showed that the system was quite vulnerable to delay and also to high rates of packet loss. In order to make the system more tolerant of such network impairments, it was suggested that a cache could be implemented. For a given image the potential anchor points are constant, while the forces at a given position will vary for different choices of anchor point. This makes the anchor points easy to cache for an image. This could easily be used to augment the anchor implementation of the networked system which communicates only the anchor points and computes the haptic forces locally at the client machine as described in section 6.4.

Based upon the data in Table 3 we can compute that with a threshold value of 0.5 we get between 0.88% and 5.19% anchor points out of the total number of pixels. The resulting small amount of anchor points makes it possible to do caching of these anchor point since we only need to send and store the anchor points rather than keeping a full matrix of forces for each point in the image at the client (this would require large amounts of bandwidth and memory).

Table 4 and Table 5 show the amount of memory required for the different numbers of anchor points both for 1 slice, and for 3D with 100 slices. The (median) required memory is computed as the **(median number of points)\*3\*sizeof(double)**. We multiply by 3 since an anchor point is a gradient which has the form of **(x, y, z)** where each variable is a double. The size of a double is 8 bytes.

**Table 4: kB of data for 1 slice**

Image	Size	Threshold	Min	Median	Mean	Max
Phantom hip	256*256	0.5	13.477	16.570	16.898	22.406
Phantom hip	512*512	0.5	28.805	56.344	56.203	100.219
Hip patient	512*512	0.5	2.250	7.875	10.500	53.813
Abdomen	512*512	0.5	0.563	89.250	89.883	318.984

**Table 5: MB of data for 100 slices**

Image	Size	Threshold	Min	Median	Mean	Max
Phantom hip	256*256	0.5	1.3	1.6	1.7	2.2
Phantom hip	512*512	0.5	2.8	5.5	5.5	9.8
Hip patient	512*512	0.5	0.2	0.8	1.0	5.3
Abdomen	512*512	0.5	0.1	8.7	8.8	31.2

In order to show the bandwidth required for a certain probability of getting a cache hit after 1, 15, 30, and 60 seconds we have drawn graphs. The choice of these time intervals was to show that given the relatively small amount of memory required to store the anchor points, even with 100 slices for a 3D dataset, this data can be fully cached in a reasonable time.

The algorithm for calculating the probability is simplistic. We made two assumptions:

1. The points are evenly spread.
2. The user can (and would like) to move to any part of the image instantly.



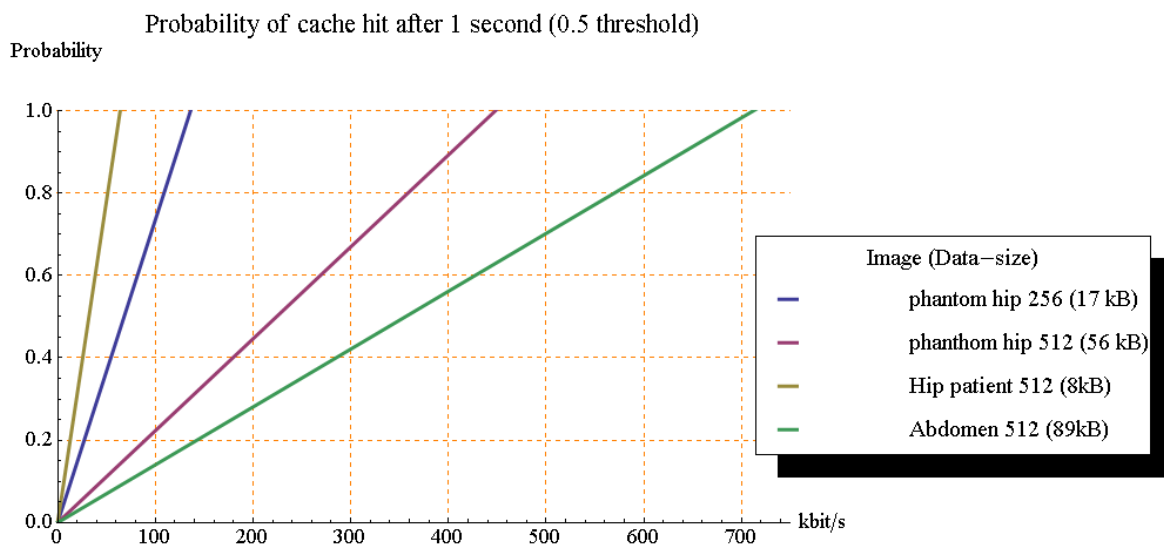
The probability of a hit is calculated by: **(available bandwidth \* time) / total number of anchor points**.

It should be possible to extend this model in various ways. Some ideas are:

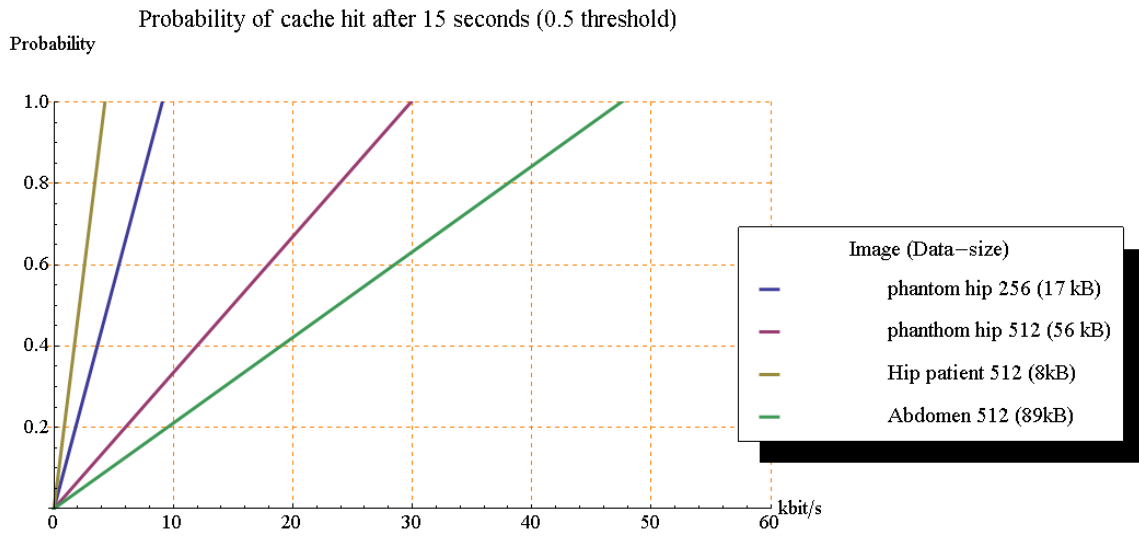
1. Take into account that the anchor points seem not to be evenly spread through the image. Several large parts of the image contain no regions of interests for the user. Therefore the anchor point density will probably be higher at the places where the users actually will move around.
2. Take into account the user's movement patterns, e.g. when outlining a shape the user will probably not move very fast or very far. Therefore it should be possible to shrink the area needing to be cached.
3. Does the user often zoom in the image? This will further reduce the area in need of caching.

A more advanced algorithm is outside the scope of this thesis, but could be a topic for future work.

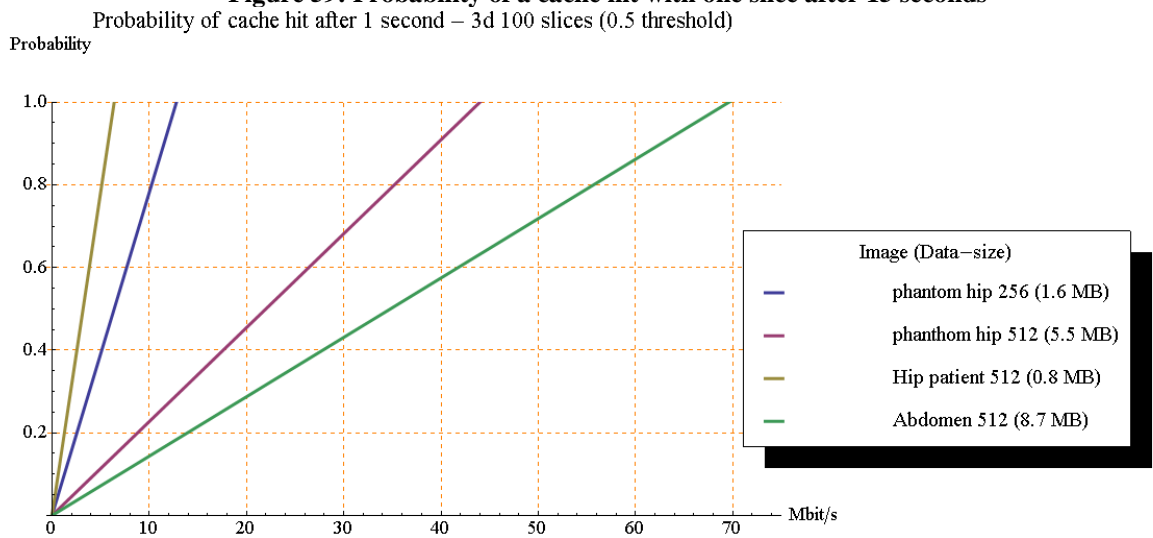
How to deal with the time variable is the next problem we consider. When drawing the graphs we set 60 seconds as the upper limit of time before achieving a 100% cache probability. We selected this value, because we thought that if cache misses still occur after one minute, users will probably end up irritated. Secondly, the amount of data that needs to be cached can be completely transferred in one minute even on slow links. The small amount of data required for a single slice makes it possible to transfer a full cache even via very low bandwidths as shown in Figure 58 and Figure 59. Even for 100 slices, no more than 1.2 Mbit/s is necessary in order to fully cache the anchor points for the image with the largest number of anchor points in 60 seconds, as can be seen in Figure 60 to Figure 63.



**Figure 58: Probability of a cache hit with one slice after 1 second**

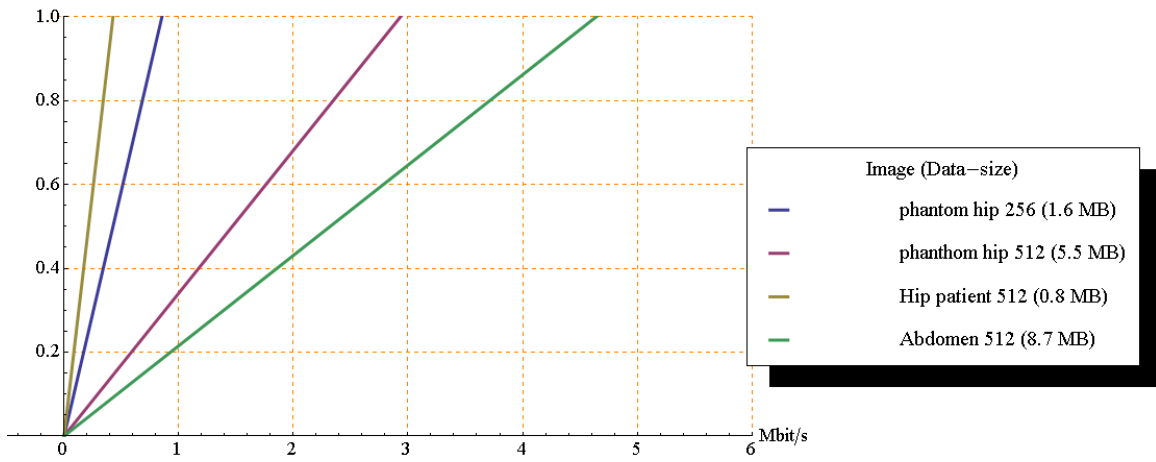


**Figure 59: Probability of a cache hit with one slice after 15 seconds**



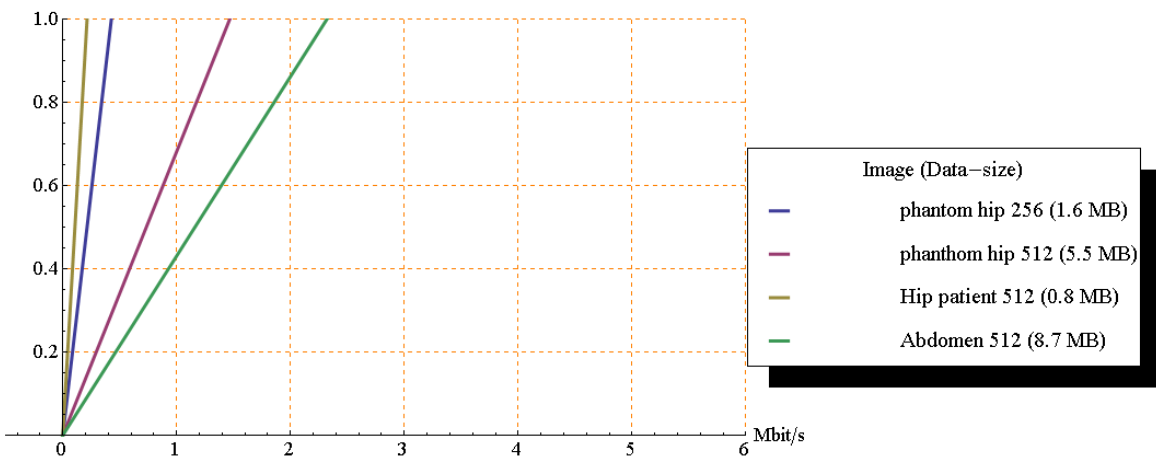
**Figure 60: Probability of a cache hit with 100 slices after 1 second**

Probability of cache hit after 15 seconds – 3d 100 slices (0.5 threshold)  
Probability



**Figure 61: Probability of a cache hit with 100 slices after 15 seconds**

Probability of cache hit after 30 seconds – 3d 100 slices (0.5 threshold)  
Probability



**Figure 62: Probability of a cache hit with 100 slices after 30 seconds**

Probability of cache hit after 60 seconds – 3d 100 slices (0.5 threshold)  
Probability

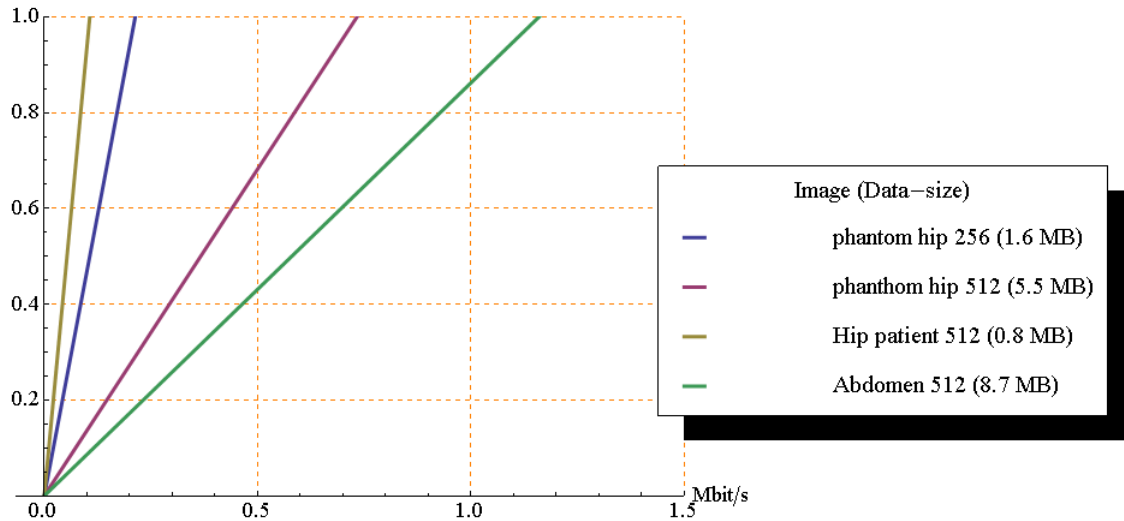


Figure 63: Probability of a cache hit with 100 slices after 60 seconds

## 7.2.4 Network analysis

As mentioned before, we had two different goals for the network part of this thesis:

1. Analyze how delay and packet loss affects the user's perception of the haptic system.
2. If delay negatively affects the user's perception of the haptic force, then find a solution that allows the haptic system to work despite high delay.

We constructed a test setup that allowed us to collect data with the help of test users. This data helped us describe the system's behavior and to analyze how the user's perception of the haptic system was affected when delay and packet loss were (separately) introduced. The only shortcoming in reaching our first goal was the relatively small number of users we had performing the test.

The second goal was met with the description of a cache system as described in section 7.2.3. We described a possible solution to allow the system to work during high delay, but unfortunately we did not have time to do an implementation, tests, and evaluation of this implementation.

## Chapter 8 - Conclusions and Future Work

This chapter briefly summarizes our conclusions and suggests future work for potential subsequent projects.

### 8.1 Conclusions

We believe that we meet our goals. More details about this are given in sections 7.1.2 and 7.2.4. While we did not establish as clear a model for the decrease in perceived quality of haptic feedback as a function of increased delay, plots of our experimental data for delay are similar to that of Cole and Rosenbluth's model of decreasing voice over IP quality as a function of delay, but with a much faster decrease in quality as a function of delay. As there is not necessarily a numeric correspondence between the quality values that we used and the ITU MOS quality values, we cannot make a numeric comparison between our data and the Cole and Rosenbluth model, but our data would seem to suggest that there is **not** an initial linear decrease in quality for low delays and that the decrease in perceived quality was **not** as fast as one might expect considering simply the ratios of the voice packet rate (typically 50 Hz) and the 1000 Hz rate of the haptic feedback loop. The rate of decrease in quality seems to only be about one half of what you might expect based upon the ratio of these packet rates (i.e., a factor of 10x rather than 20x).

### 8.2 Suggestions for others working in this area

#### 8.2.1 Examine how much computation actually needs to be done each time the haptic loop runs

During our tests we had access to machines with dual and quad core CPUs as well as 8 gigabytes or more of RAM. Therefore, our code is probably not optimized as much as possible. One thing that should be considered is which parts of the code need to be executed in every loop of the callback code, which currently runs 1000 times a second. For example, a user will probably not be able to move the cursor to different positions a thousand times each second.

#### 8.2.2 Usage of another haptic device

We only used the SensAble PHANTOM Omni haptic device in our project so our results might not be completely accurate for similar devices. However, the big differences are probably in the way the programmer communicates with the different brands of haptic devices, i.e. the different API, thus our theories and results should be fairly the same regardless of what device is being used. Although, a different device might have handled the velocity parameter better and thereby would have opened up the possibility of an implementation of a viscous force.

#### 8.2.3 If we had to do it again, what would we have done differently?

The things that caused us the most trouble were related to the fact that we were two people doing this thesis together.

We had two different versions of Microsoft Word which caused much trouble when the older version opened files from the newer one (.docx), and then saved them with loss of formatting and functions such as automatic figure numbering as a result. The solution to this problem is to keep the document in the older format at all times.

Google Wave[26] says it will allow several people to edit the same documents simultaneously and we think that a function like that would really help when writing one report together.

We also felt that we would have benefitted from using some sort of versioning system when doing the actual programming. We stuck to the plan with manually saving and naming different versions of the files and it became a real hassle to know what we had done in each file after a while.

Another problematic aspect was all the movement of files (e.g. logs, screenshots, data, texts, manuals, the actual thesis, and so on). We had two different machines at the Karolinska Institute, one laptop each, and several machines at home for testing and writing. We used email and an USB-memory stick to move and share files, but this was really annoying and did not give us instant access to each other's up-to-date files. Some sort central storage of the files would have been useful. For example we found Dropbox[27], but discovered it much too late into the project.

Write every day! You cannot get this advice too many times (or too often)! It is way too easy to get caught up with something you are currently doing, and think "I will just finish this and write tomorrow instead". Then tomorrow you say "tomorrow" again and soon you will lose the flow of writing and end up with days or even weeks of material that needs to be put into the report.

## **8.3 Future work**

### **8.3.1 Better ways to calculate anchor points**

The resulting quality of our idea of snapping the cursor to anchor points relies on how you process your image and how you decide which points should be anchor points. If you do this badly, you will end up with way too many anchor points that should not have been there and the resulting user experience will be bad.

For this thesis we used the OpenDX network and code previously developed by Professors Marilyn E. Noz & Gerald Q. Maguire Jr. This system computes the gradient at every voxel of the image, and stores this into a force matrix. The system originally used these gradients as the force sent to the haptic device. Therefore every gradient was normalized by the maximum gradient of its slice in the image in order to stay within the range of force suggested for the haptic device.

Based upon our experience we believe that it would be better to skip the normalization and leave the gradients as they are. This could prove to be a better way to set the threshold, hence choosing better anchor points.

Another idea, proposed by Marilyn E. Noz, is to take the points around the current one into consideration. By doing this, it should be possible to find and exclude single points that are unsuitable as anchor points.

### **8.3.2 Improve movement along lines**

As discussed in our results, we are not completely satisfied with the forces when moving along lines. As you can see in Figure 39, several shapes produce more or less a single line of anchor points. This requires the user to move exactly over the points for a smooth movement. Any movement outside these points would result in a force trying to get the user back on the line.

Our attempts to implement a feature to make it easier to move along a line were not successful, see section 5.9.6.

### **8.3.3 Cached network haptics**

Our original intent was to make the system tolerant to high latency by implementing the cache described in section 6.5. This part is still left undone, but we think that we have documented our work and theories sufficiently for someone else to continue this work.

### **8.3.4 Experimental measurements with very low added delays**

The absence of the initial linear portion of the Cole and Rosenbluth model might simply be missed because this would correspond to very short delays and there were only a very limited number of test users. This effect is unlikely to have any practical effect on network induced delays beyond a campus local area network – as the network delay will be large enough that the next portion of the Cole and Rosenbluth model would seem to apply. However, measurements with much shorter delays might be an interesting task for future work.

## **8.4 Work that we have not completed**

### **8.4.1 Introduce a way to disable the forces**

During our tests and implementation we could not temporarily disable the force output. We also think that a user of the finished system would find it useful to be able to temporarily disable the forces. One reason that this might be desirable is that they find that they are not able to reach certain parts of the image or get stuck at a point or in a region.

### **8.4.2 Extending the work and experiments to 3D**

We did not get a chance to implement our forces in 3 dimensions. We feel that our idea with anchor points works very well in two dimensions and we cannot think of any circumstances that would prevent it from working in three dimensions.

### **8.4.3 Add a viscous force and wall force**

Due to the fact that the velocity parameter of the device did not work as we expected it to, we did not have a chance to try our idea for a viscous force, see section 5.2.5.

After finishing the implementation and evaluation of the spring force and the magnetic force we decided that these forces worked sufficiently well to allow us to skip the wall force, as we thought that our time was better spent completing some of the network part.

#### **8.4.4 Completely split the code for the network part**

We only implemented a bouncer solution as mentioned in section 6.4. In order to add the functionality to use the system in a real world scenario, i.e., between two different locations a complete split of the code is necessary; this is described in section “Proposed method 2”.

#### **8.5 The most obvious next steps**

For the forces part support should be added for 3D. This has been our most requested feature. As for the network part, a complete split of the code and support for caching is needed.

#### **8.6 Hints for someone who is going to follow up this work**

We really learned a lot and got a good understanding of how the haptic device worked by reading and trying the simple applications. The sample applications that SensAble provided with their API were really good. Thus we strongly recommend that anyone who continues with this work try the basics first, before trying more sophisticated haptic programming.

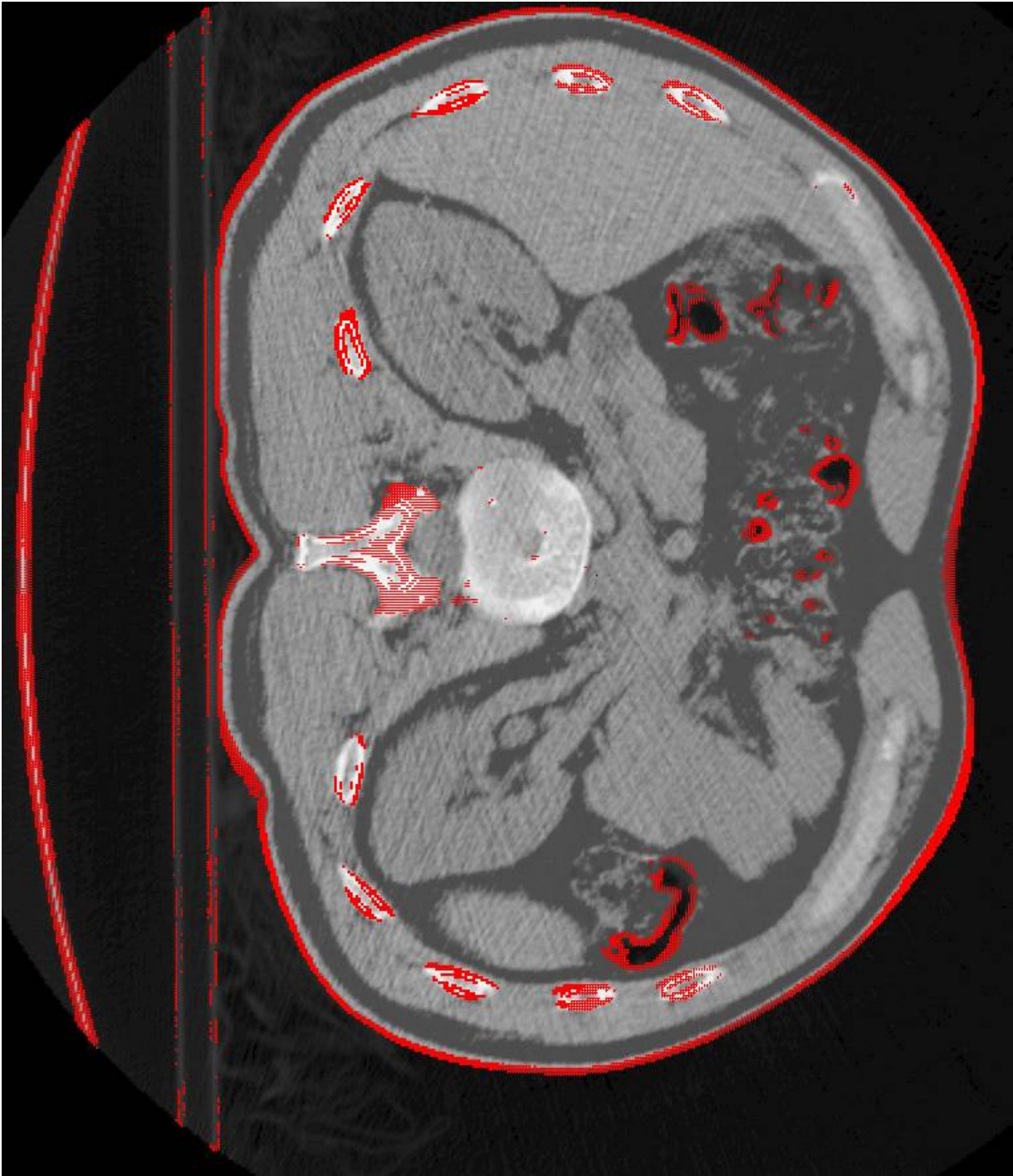


## References

- [1] E. Anderlind, *Haptic Feedback for Medical Imaging and Treatment Planning*, Master Thesis, School of Electrical Engineering, Royal Institute of Technology, Stockholm, June 2006.  
[http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2006/rapporter06/anderlind\\_eva\\_06077.pdf](http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2006/rapporter06/anderlind_eva_06077.pdf)
- [2] K M van Mensvoort, *What You See is What You Feel: On the simulation of touch in graphical user interfaces*, Dissertation, Eindhoven University of Technology, Eindhoven, April 2009, ISBN: 978-90-386-1672-8.  
[http://www.powercursor.com/download/what\\_you\\_see\\_is\\_what\\_you\\_feel.pdf](http://www.powercursor.com/download/what_you_see_is_what_you_feel.pdf)
- [3] K. Lundin Palmerius, *Direct Volume Haptics for Visualization*, Dissertation, Department of Science and Technology, Linköping University, Norrköping, May 2007.  
<http://liu.diva-portal.org/smash/record.jsf?pid=diva2:23459>
- [4] B. Riedel and A. M. Burton, *Perception of Gradient in Haptic Graphs: a Comparison of Virtual and Physical Stimuli*. In Eurohaptics 2001, pp. 90–92, Birmingham, UK, 2001.  
<http://www.dcs.gla.ac.uk/~rayu/Publications/eurohaptics2001.BR.pdf>
- [5] SensAble Technologies, Inc. Homepage, *PHANTOM Omni® Haptic Device product info*, last accessed 21/8-2009.  
<http://www.sensable.com/haptic-phantom-omni.htm>
- [6] SensAble Technologies, Inc., *Sensable Open Haptics Toolkit version 3.0 Programmer's Guide*, last accessed 21/8-2009.  
[http://www.sensable.com/documents/documents/OpenHaptics\\_ProgGuide.pdf](http://www.sensable.com/documents/documents/OpenHaptics_ProgGuide.pdf)
- [7] Belkin International, Inc. Homepage, *Belkin Nostramo n30 Game Mouse product sheet*, last accessed 21/8-2009.  
<http://www.belkin.com/pbs/123033.pdf>
- [8] Immersion TouchWare software download page, last accessed 21/8-2009.  
[http://www.ifeelpixel.com/download/drivers/immersion/immersion\\_touchware\\_desktop.htm](http://www.ifeelpixel.com/download/drivers/immersion/immersion_touchware_desktop.htm)
- [9] iFeelPixel software description page, last accessed 21/8-2009.  
<http://www.ifeelpixel.com/description/>
- [10] Amazon.com Logitech iFeel Mouse product information, last accessed 21/8-2009.  
<http://www.amazon.com/exec/obidos/asin/b00005asfk/ifeelpixel-20>
- [11] Amazon.com Logitech WingMan Force Feedback Mouse product information, last accessed 21/8-2009.  
<http://www.amazon.com/exec/obidos/asin/b00001w01z/ifeelpixel-20>
- [12] Kensington Orbit® 3D Trackball product information, last accessed 21/8-2009.  
<http://web.archive.org/web/20050204225648/www.kensington.com/html/1402.html>
- [13] Kensington Orbit® 3D Trackball product sheet, last accessed 21/8-2009.  
<http://web.archive.org/web/20040728083028/app.kensington.com/images/prodSheets/K64232PS.pdf>
- [14] Amazon.com Saitek W07 Touch Force Gaming Mouse product information, last accessed 21/8-2009.  
<http://www.amazon.com/exec/obidos/asin/b000063df9/ifeelpixel-20>

- [15] HP Force Feedback Web Mouse product information, last accessed 21/8-2009.  
<http://web.archive.org/web/20061021230046/http://www.rca.com/product/viewdetail/0,2588,PI45864-CI100102,00.html>
- [16] OpenDX.org Homepage, last accessed 21/8-2009.  
<http://www.opendx.org/>
- [17] J. Postel, *User Datagram Protocol*, Internet Engineering Task Force, RFC 768, 28 August 1980.  
<http://www.ietf.org/rfc/rfc0768.txt>
- [18] Man page for Traffic Control, last accessed 21/8-2009.  
<http://linuxreviews.org/man/tc/>
- [19] Linux Foundation Homepage, *Net:Iproute2 information*, last accessed 21/8-2009.  
<http://www.linuxfoundation.org/en/Net:Iproute2>
- [20] Linux Foundation Homepage, *Net:Iproute2 examples*, last accessed 21/8-2009.  
<http://devresources.linux-foundation.org/shemminger/netem/example.html>
- [21] Linux Foundation Homepage, *Net:Netem information*, last accessed 21/8-2009.  
<http://www.linuxfoundation.org/en/Net:Netem>
- [22] R. M. Traylor, D. Wilhelm, B. D. Adelstein, and H. Z. Tan, *Design Considerations for Stand-alone Haptic Interfaces Communicating via UDP Protocol*, in Proceedings of the 1st Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC '05), pp. 563–564, Pisa, Italy, March 2005. Digital Object Identifier: 10.1109/WHC.2005.31  
[http://www.ece.purdue.edu/~hongtan/pubs/PDFfiles/C56\\_Traylor\\_WHC2005.pdf](http://www.ece.purdue.edu/~hongtan/pubs/PDFfiles/C56_Traylor_WHC2005.pdf)
- [23] R.G. Cole and J.H. Rosenbluth, *Voice Over IP Performance Monitoring*, Computer Communication Review, a publication of ACM SIGCOMM, volume 31, number 2, pp. 9–24, April 2001.  
<http://www.acm.org/sigcomm/ccr/archive/2001/apr01/ccr-200104-cole.html>
- [24] D. Halliday, R. Resnick, and J. Walker, *Fundamentals of Physics*, Sixth Edition, John Wiley & Sons, Inc., New York, 2001, ISBN 0-471-39222-7.
- [25] Wikimedia, *Image of a right-handed three-dimensional Cartesian coordinate system*, last accessed 21/8-2009.  
[http://commons.wikimedia.org/wiki/File:3D\\_coordinate\\_system.svg](http://commons.wikimedia.org/wiki/File:3D_coordinate_system.svg)
- [26] Google Homepage, *Wave project information*, last accessed 21/8-2009.  
<http://wave.google.com/>
- [27] Dropbox Homepage, *Filesharing application*, last accessed 21/8-2009.  
<http://www.getdropbox.com/>

**Appendix A: Medical images with anchor points**



**Figure 64: Abdomen with a threshold value of 0.3**

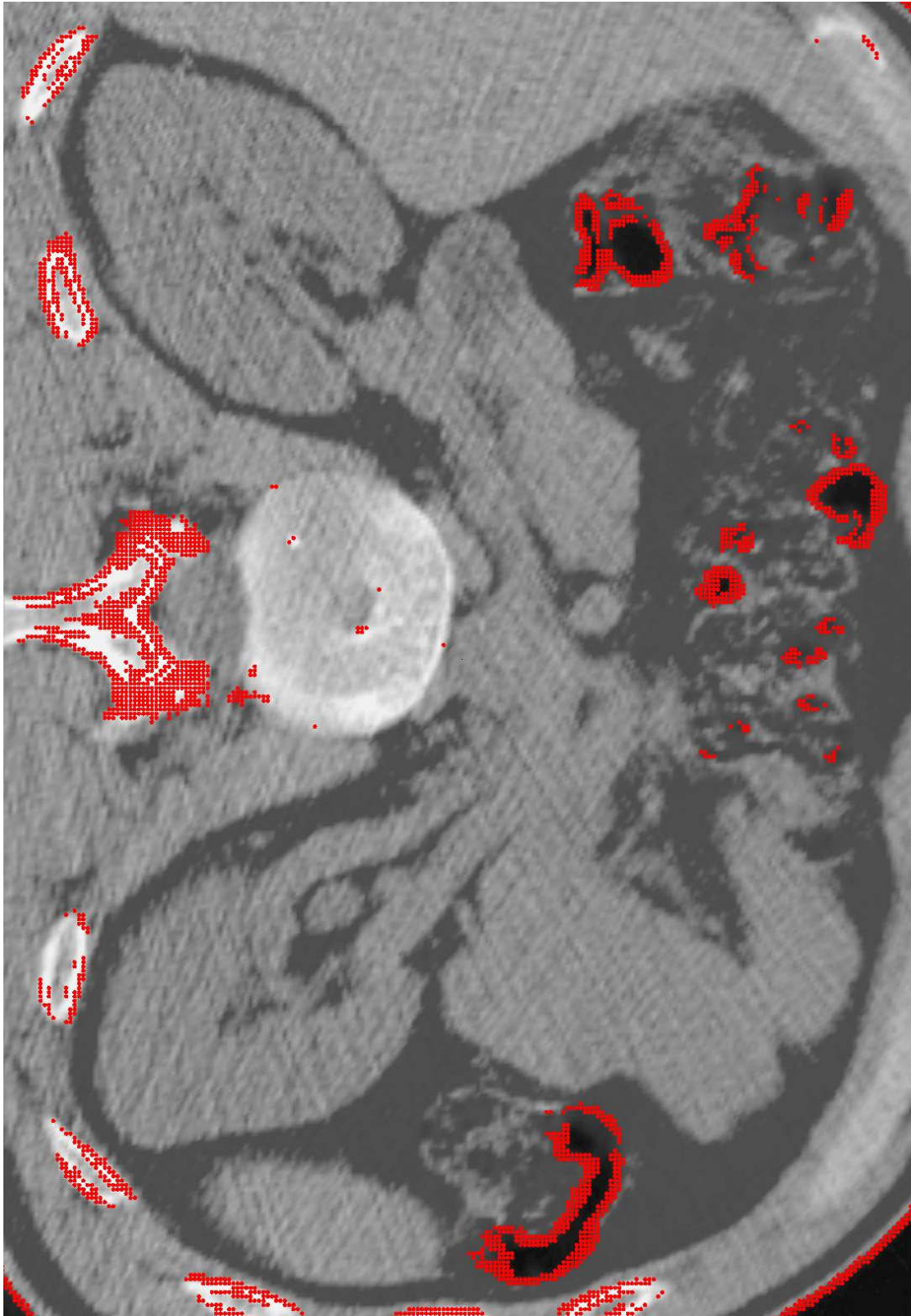


Figure 65: Zoomed in abdomen with a threshold value of 0.3



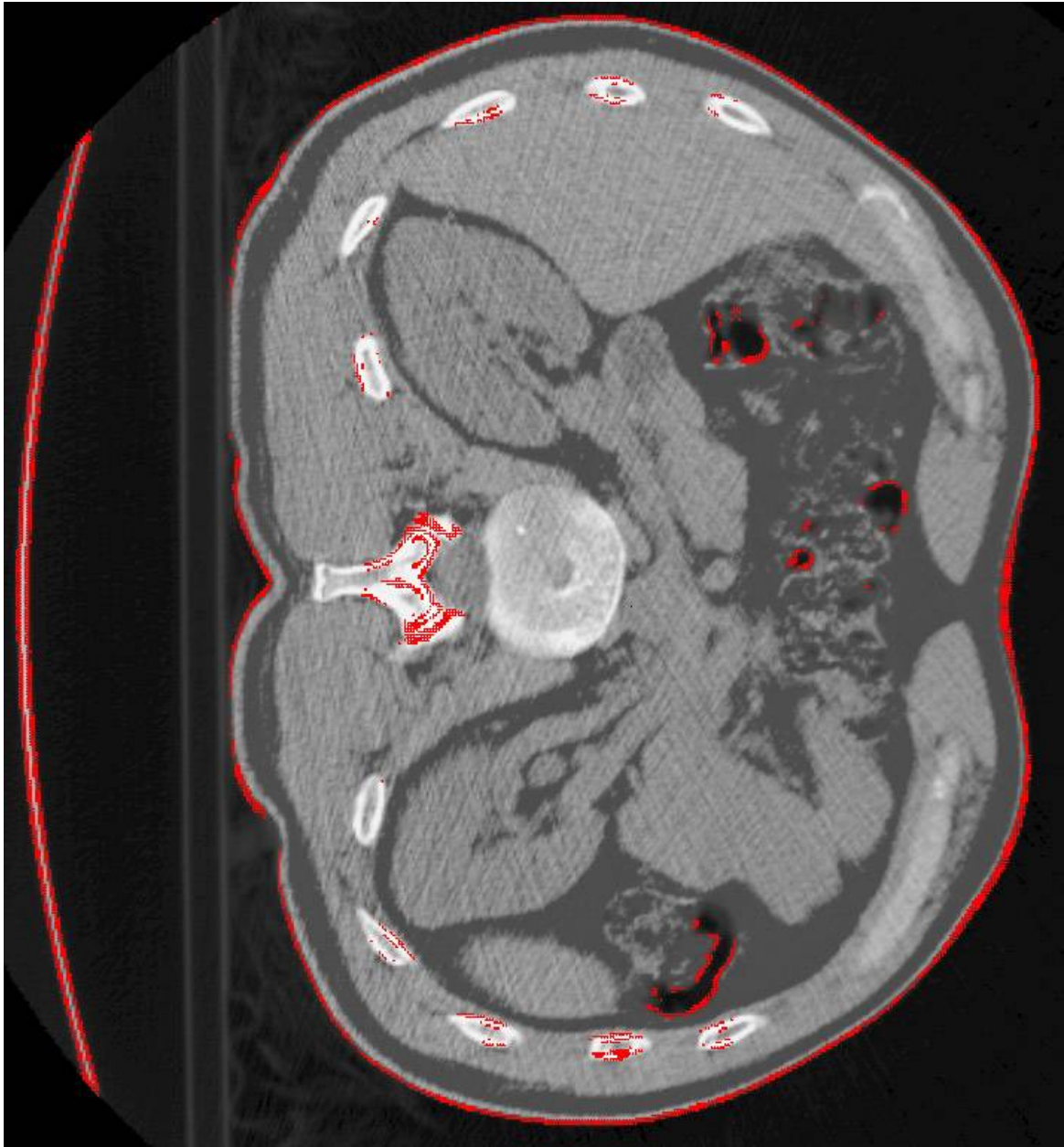


Figure 66: Abdomen with a threshold value of 0.5

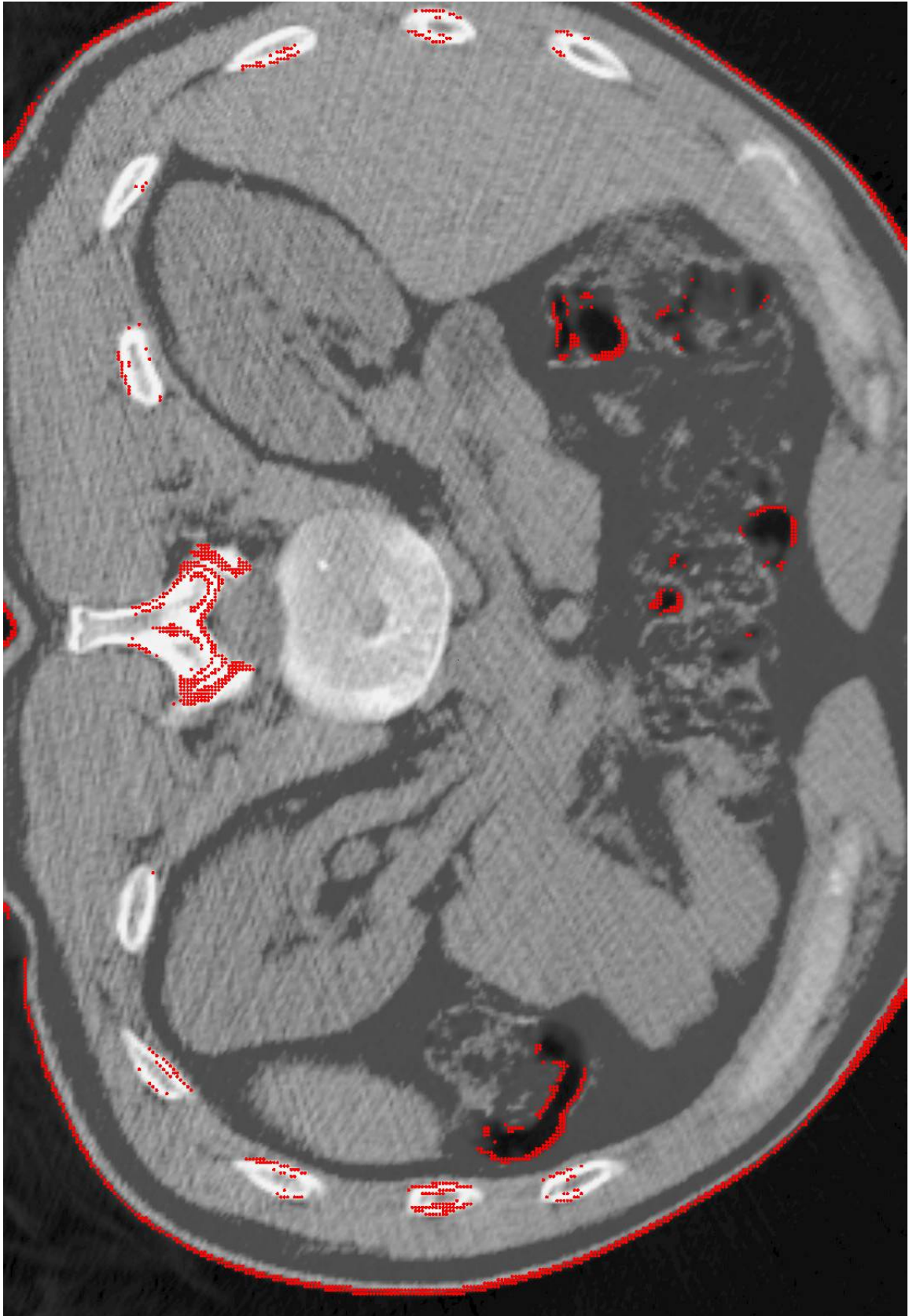
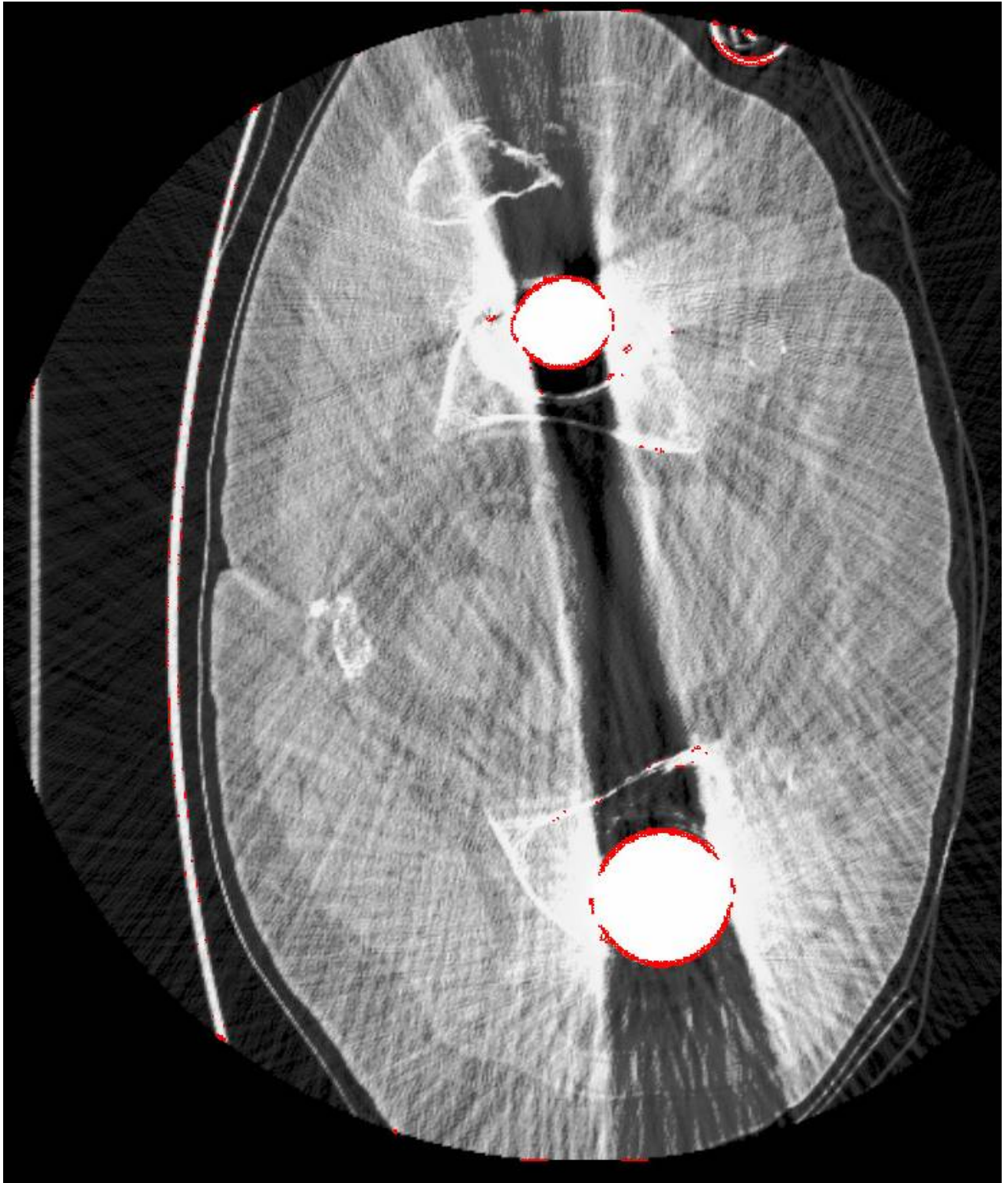


Figure 67: Zoomed in abdomen with a threshold value of 0.5



**Figure 68: Hip patient with a threshold value of 0.3**



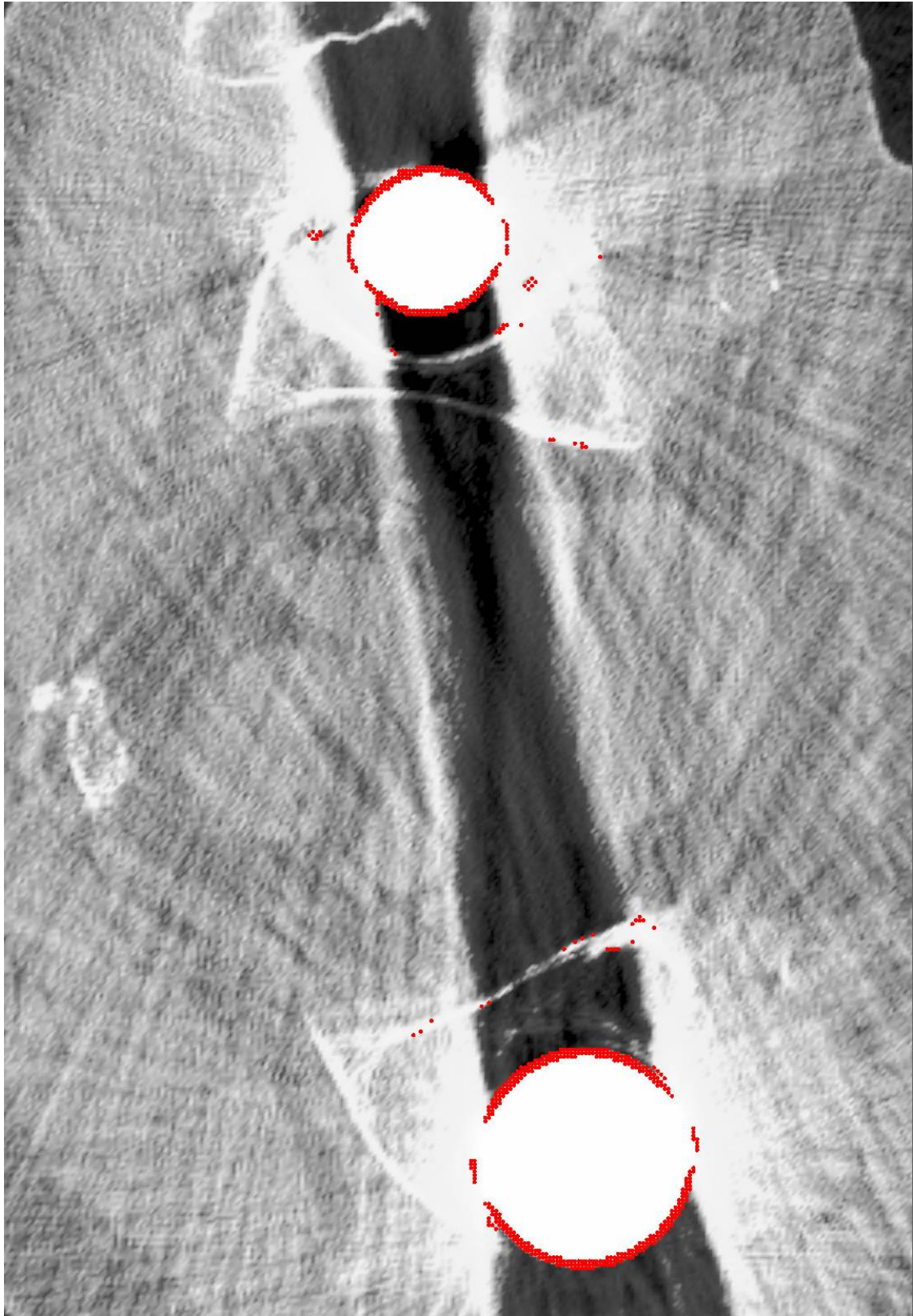
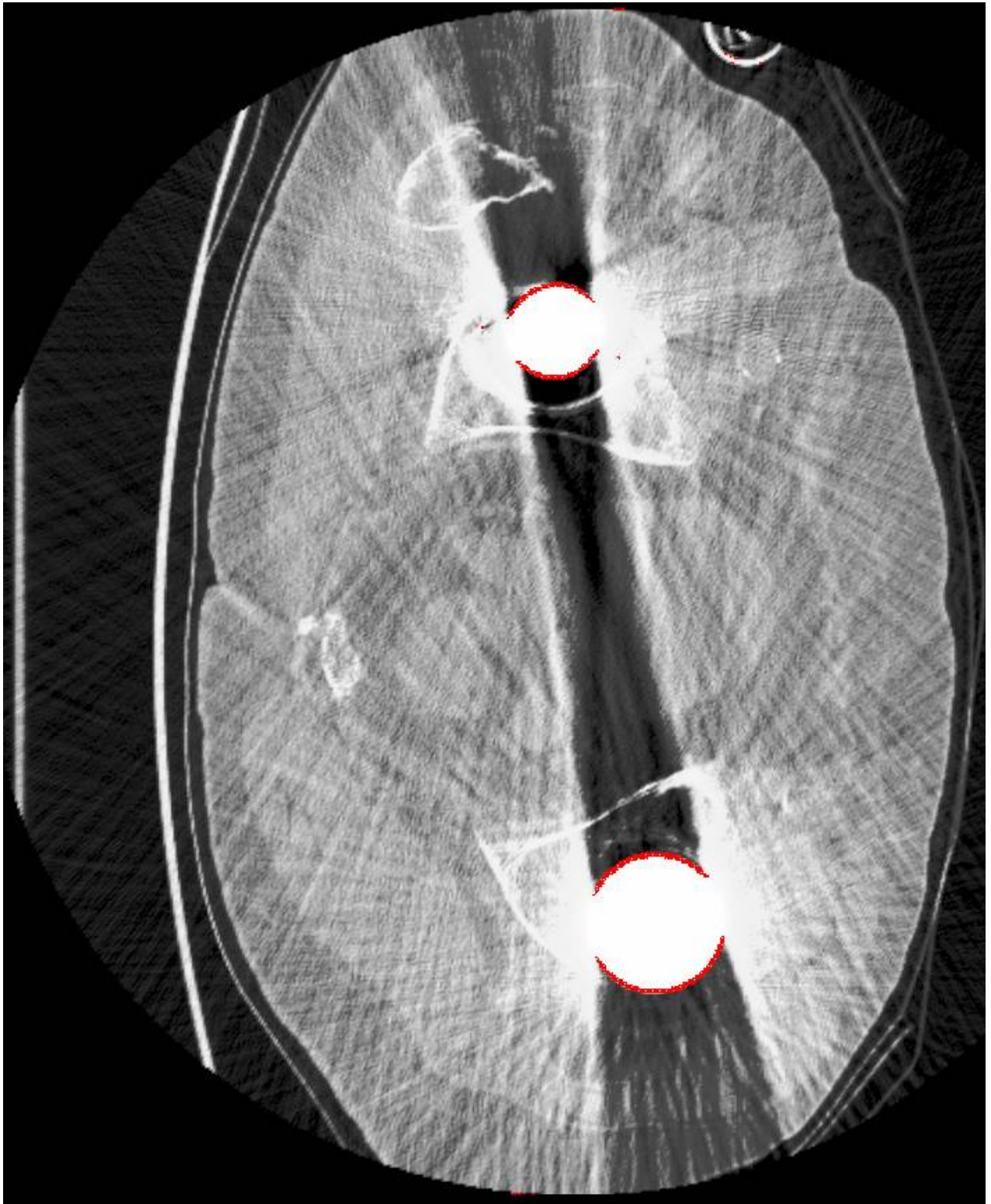


Figure 69: Zoomed in hip patient with a threshold value of 0.3





**Figure 70: Hip patient with a threshold value of 0.5**



**Figure 71: Zoomed in hip patient with a threshold value of 0.5**

