Developing a Multi-Touch Application for Medical Volume Visualization

Bachelorarbeit
zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Christopher Denter  cdenter@uni-koblenz.de

Erstgutachter:  Prof. Dr. Stefan Müller
Institut Computergrafik, Universität Koblenz-Landau

Zweitgutachter:  Dipl.-Inf. Diana Röttger
Institut Computergrafik, Universität Koblenz-Landau

Koblenz, im März 2010
Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ja  Nein

Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.  □  □

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.  □  □

(Ort, Datum)  (Unterschrift)
Aufgabenstellung für die Bachelorarbeit

Christopher Denter (Matr.-Nr. 207110270)

Thema: Developing a Multi-Touch Application for Medical Volume Visualization

The most widely used interface for a computer today is a combination of a keyboard and a mouse. Although the importance of these traditional input devices (especially for the advent of computers in our everyday life) cannot be neglected, it is obvious that the single-click nature of the mouse is limiting. As there is only one mouse, only one person can interact with the computer at a time. Additionally, this person also has to understand the transfer of the hand's movements into on-screen cursor movements.

The term Multi-Touch describes a way of interaction between humans and computers. By leveraging state-of-the-art technology, devices can be built that allow for multiple users to directly manipulate a screen's contents with one or more of their fingers. This leads to much more intuitive and natural user interfaces. In order to fully exploit the potential that these technologies offer, applications must be designed in a way that allows for usage in a multi-touch context.

One area where these novel input methods would be beneficial is the visualization of medical volume data. In diagnostics, a deep understanding of a patient's medical image data (as obtained by a CT or MR system) is crucial to choosing an appropriate treatment. Volume rendering assists here by visualizing the spatial relationships of the patients body. Multi-Touch can further be used to enhance the process of reviewing medical data, e.g. by using gestures to perform certain operations more quickly. Potentially, several people can collaborate more efficiently.

This thesis therefore explores the possibilities of easing the interaction for users of such applications.

On top of already existing libraries for rendering and multi-touch input handling such as VTK and PyMT a medical visualization tool will be developed. Certainly, integration of the required components is an important concern. However, the focus of this thesis is on utilizing multi-touch input methods in the context of visualization tasks and dealing with the challenges they bear.

This comprises the following steps:

1. Explaining the different hardware and software components, how they work and interact.
2. Identifying areas where multi-touch enabled input devices can support the user in a medical volume visualization context.
3. The creation of a multi-touch aware user interface for medical volume visualization tasks. Optimizing the user interface elements with the special capabilities and limitations of multi-touch input devices in mind.
4. Evaluation of the interface with respect to efficient and intuitive usage.
5. Documentation of the results.

Betreuerin: Dipl.-Inf. Diana Röttger
Koblenz, den 12. November 2009

- Christopher Denter -

- Prof. Dr. Stefan Müller -
Acknowledgements

I want to give credit where credit is due. First of all I have to thank Mathieu Virbel and Dr. Mark Batesole for their help and their technical and medical expertise, respectively. Next I would like to thank my family for all the support and especially my father for the invaluable woodworking guidance. Lastly I would like to thank my lovely girlfriend Laura for her patience.
Contents

1 Introduction .................................................................... 1
  1.1 Multi-Touch .............................................................. 1
    1.1.1 Usage Scenarios ................................................... 1
    1.1.2 Gestures ............................................................. 2
  1.2 Volume Rendering .................................................... 3
    1.2.1 Volume Datastructures ........................................... 3
    1.2.2 Marching Cubes .................................................. 4
      Isosurfaces .............................................................. 4
      The Algorithm ........................................................ 5
      Problems ............................................................... 6
    1.2.3 Ray Casting ....................................................... 6
      Transfer Functions .................................................. 6
      Trilinear Interpolation ............................................. 7
      The Algorithm ........................................................ 8
  1.3 Medical Imaging Modalities ...................................... 10
    1.3.1 X-Ray Imaging .................................................. 10
    1.3.2 Computed Tomography ........................................ 11
    1.3.3 Magnetic Resonance Imaging ............................... 12
    1.3.4 Storage of Medical Image Data ............................ 13
    1.3.5 Hounsfield Density Units .................................. 14

2 Hardware ....................................................................... 15
  2.1 Principles of Optical Multi-Touch Systems .................. 15
    2.1.1 Utilization of Infrared Light .................................. 15
    2.1.2 Diffuse Illumination ............................................ 17
    2.1.3 Frustrated Total Internal Reflection ....................... 19
    2.1.4 Diffuse Surface Illumination ............................... 21
  2.2 Documentation of Hardware Assembly ....................... 21
    2.2.1 Hardware Components ....................................... 22
      Illumination .......................................................... 22
      Camera ................................................................. 23
      Image Projection .................................................... 25
      Computer ............................................................. 25
      Calibration .......................................................... 26
      Table Framework .................................................. 27

3 Software ...................................................................... 28
  3.1 Software Dependencies .......................................... 28
    3.1.1 Programming Language: Python .......................... 29
    3.1.2 Multi-Touch Input and User Interface: PyMT .......... 30
    3.1.3 Volume Rendering: VTK ...................................... 31
    3.1.4 Tracking: CCV et al. ......................................... 32
Community Core Vision .......................... 32
reactIVision ...................................... 33
Movid .............................................. 33
3.1.5 Touch Information Transmission: TUIO Protocol . . . 33
3.2 Related Software: OsiriX .......................... 34
User Interface ..................................... 34
Analysis .......................................... 35
Ideas for Improvement ............................. 35
3.3 Widget Overview .................................. 35
3.3.1 Rendering Widget ............................. 37
Rotation .......................................... 37
Scaling ........................................... 37
3.3.2 Transfer Function ............................. 39
Adding & Moving Points .......................... 39
Removing Points ................................... 39
Global Adjustments ................................ 39
Visual Appearance ................................. 40
3.3.3 Color Picker ................................. 41
Realization ........................................ 41
3.3.4 Presets ....................................... 42
3.3.5 Dataset Gallery ............................... 42
3.3.6 Widget Integration ............................ 43
4 Evaluation ......................................... 45
4.1 Hardware Evaluation .............................. 45
4.1.1 Solved Problems .............................. 46
Weak Contrast ..................................... 46
Delayed Response ................................ 47
Image Projection .................................. 47
Jitter Inaccuracy .................................. 47
Lamp Reflection ................................... 48
Double-Tapping Thresholds ........................ 48
4.1.2 Results ....................................... 49
4.1.3 Possible Improvements ....................... 49
Better Camera ..................................... 49
Higher Luminous Intensity ......................... 50
Improved Tracking Algorithm ..................... 50
4.2 Software Evaluation .............................. 51
4.2.1 Solved Problems .............................. 51
Interacting with the VTK Scene .................. 51
Integrating VTK and PyMT ....................... 52
Parallelizing Tasks ............................... 53
Performance Improvements ....................... 54
4.2.2 Results ..................................... 54
Volume Manipulation Gestures . . . . . . . . . . . . . 56
Transfer Function Manipulation Gestures . . . . . . . 56
Color Picker Gestures . . . . . . . . . . . . . . . . . . . . . . . 57
4.2.3 Possible Improvements . . . . . . . . . . . . . . . . . . . 58
Volume Rotation . . . . . . . . . . . . . . . . . . . . . . . . . . 58
Clean Parallelization . . . . . . . . . . . . . . . . . . . . . . . 58
4.2.4 Expert Evaluation . . . . . . . . . . . . . . . . . . . . . . 59
Potential of Multi-Touch . . . . . . . . . . . . . . . . . . . . . 59
BoneTouch-Specific Tasks & Questions . . . . . . . . . . 59
Subjective Impression . . . . . . . . . . . . . . . . . . . . . . 60

5 Conclusion & Further Work 61
Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
Further Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

A Instructions i

B Survey i
  B.1 General Questions . . . . . . . . . . . . . . . . . . . . . . . i
  B.2 Tasks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . i
  B.3 BoneTouch-related Questions . . . . . . . . . . . . . . . . . ii
  B.4 Subjective Impression . . . . . . . . . . . . . . . . . . . . . . iii
Abstract

With the advances in medical imaging, more and different kinds of data are being acquired and different scanning techniques become increasingly important. This is due to the fact that anatomical information about a patient’s body can be obtained without actually cutting into it. Modern scanning devices generate three-dimensional representations of the scanned objects that can be rendered by a computer and presented to the physician interactively. By adjusting complex parameters, it is even possible to alter the visual appearance of different tissues depending on their density, which allows for visual discrimination of interesting structures. The applications utilized for rendering are rather advanced and their usage might be inefficient. It is hypothesised that this is partially due to the limited input bandwidth of traditional interaction devices and techniques. An already existing, mouse-controlled volume rendering application will be analyzed in this thesis. The analysis will be focused on potential shortcomings resulting from the limited nature of the input device. Based on this analysis, it will be proven that user interaction would benefit from another type of control. Multi-Touch, a novel human-computer interaction technique, allows for considerably higher input bandwidth and more natural user interfaces. With multi-touch, the user can utilize potentially many fingers at the same time to interact with a computer through a touch sensing device. Many tasks that formerly required some mouse clicks can now be carried out more efficiently with corresponding gestural commands. This, however, requires a change of thinking with respect to user interface design and usage metaphors.

The applicability of multi-touch in medical volume rendering contexts will be researched in this thesis. To obtain meaningful results, a prototype application with multi-touch user interface will be developed and presented, with emphasis being put on easy and efficient user interaction.

The verification of the aforementioned assumption can only take place when a multi-touch input device is used to evaluate the prototype. Accordingly, such a device will be designed and built in the course of this thesis.

The discussion commences by introducing the reader to all concepts and the terminology required to understand the different aspects that are key to success. An evaluation of both the hardware and software components will conclude the thesis.
Zusammenfassung


1 Introduction

The work presented in this thesis mainly builds upon two developments in computer graphics and human-computer interaction research. In the course of this thesis, a multi-touch input device (hardware part) was built. This device was used to test a volume rendering application (software part) with a multi-touch-enabled user interface that was also developed. The following two sections introduce the reader to the terminology and basic concepts of these two fields. Additionally, section 1.3 explains two different sources commonly used to obtain the data used in medical volume rendering applications.

1.1 Multi-Touch

When a user wants to interact with a computer, an input device is used to express the user’s intent. Many different types of input devices are available, such as mice, keyboards, joysticks, touch screens and even microphones (for speech recognition). All input devices can be characterized by the amount of Degrees of Freedom (DOF) they offer the user. Therefore, with increasing complexity, some devices give the user more control than others.

A relatively new type of input device that is said to support Multi-Touch enables the user to interact with a computer through a surface that directly senses the presence of the user’s fingers when touching the surface. Ideally, there is no hard limit for the maximum number of touches that can be processed at a time. Each touch usually has 2 DOF, namely its x/y position on the input surface. For example, instead of using two mice for $2 \times 2$ DOF, the user can simply use two fingers of the same or different hands. This allows for utilization of gestural commands, such as swiping two fingers from top to bottom to scroll in a document, or moving one finger around the other to rotate an image.

When the surface used for touch input coincides with the computer screen, the user can directly manipulate the on-screen contents. For many basic operations (such as scaling an image), the corresponding multi-touch gesture is assumed to be more natural and impose a lower initial barrier on the user.

1.1.1 Usage Scenarios

Probably the first device that made multi-touch available for the mass market was the Apple iPhone. Since its introduction in 2007, Apple has extended...
its product palette by several other devices that make heavy use of multi-touch. Other vendors such as DELL and HP also offer multi-touch hardware.

In addition to these consumer products, custom-made multi-touch hardware was built for public exhibitions such as the Future Ocean Explorer [FOE], a massive installation to showcase facts about oceanic regions, or the Puddle of Life installation at Coimbra’s Science Museum [PoL]. Constructions of this kind are appealing due to their highly futuristic appearance and the fact that many users can immediately start interacting with the system.

This thesis explores how the utilization of a multi-touch interface for medical volume visualization can aid physicians with their basic visualization tasks. Advantages and drawbacks, as well as efficiency and subjective preference will be analyzed [MT09].

1.1.2 Gestures

When a user interacts with a multi-touch interface, one or more fingers or hands are used to convey the user’s intent. Anything from simple single-tap operations (similar to a single-click) to complex bi-manual actions (such as a shape being drawn with both hands) can be interpreted as commands by the user interface. In the context of this thesis, any such operation is referred to as a gesture.

Although there is no standardized set of gestures, it can be stated that similar operations are carried out similarly in different contexts (figure 1). One common gesture is the selection and translation of an object by touching and dragging it. By touching the object, it is selected. By moving the finger while still touching it it is then translated in the direction of touch movement. This is also often used to scroll in documents and websites. Another well-known gesture is used for scale operations. Two diverging fingers are used to scale an object around a scale center in between the fingers. The distance of the touch positions then determines the scale factor.
This gesture can be modified to allow for more precise definition of the scale center when only one finger moves. The other steady finger then determines the center of scale. Sticking to the website example, this gesture is often used to zoom in on interesting regions of the page or increase the font size. The gesture commonly used to rotate objects is akin to the scaling gesture. Two fingers can be rotated around the same center (which is then interpreted as center of rotation) while maintaining a constant distance to each other. Analogously to the scaling operation, only one finger can be rotated around the other to clearly define the center of rotation. One use case of this gesture is the rotation of an image. It is obvious that these are not the only gestures that can be used. It is the responsibility of the interaction designer to find meaningful and efficient gestures for the current tasks in hand [ML04; WB03].

1.2 Volume Rendering

When a three-dimensional object is projected onto a two-dimensional image plane, depth information is lost. While this is not an issue in many cases, for some applications it is necessary to preserve the information. For example, a physician often wants to examine the positional relationships between different parts of the body, such as the teeth and the skull. A solution to this problem is the generation of a volume dataset representing the part of the patient’s body the physician is interested in. This digital dataset is also only displayed on a two-dimensional screen. However, since it is available in computer memory, it can be rotated and rendered from different perspectives (even in real-time), hence compensating the lack of depth perception [Preim07].

1.2.1 Volume Datastructures

For the computer, an ordinary two-dimensional image is just a two-dimensional matrix or a two-dimensional grid. This Cartesian grid can be understood as a discrete coordinate system with two axes i and j. One pixel of the image represents one bin in the image that can be addressed by its coordinates (i, j). The resolution of the image determines the number of pixels in the i and j direction. Usually, all pixels have the same width w and height h (globally) and w = h. The entire image’s width is then \( W = res_i \times w \) and analogously, the height is \( H = res_j \times h \). In the case of gray level images (to which we will limit ourselves here for the sake of simplicity), every pixel contains a gray level value, usually represented as an integer in the range [0, 255], where 0 is black and 255 is white. The image’s file size is then \( size = res_i \times res_j \times 8 \) Bytes.

---

5The term pixel is an abbreviation for picture element.
In order to represent a three-dimensional object, the aforementioned data structure needs to be augmented. In particular, a third dimension \( k \) must be added. In the context of medical image data, this is usually achieved by creating a stack of two-dimensional images. The index \( k \) then addresses the \( k \)th image in the stack. Every individual image in the stack is known as a slice.\(^6\) It is important to note that a slice in medical data is not infinitesimally thin, but has a certain width.

The three-dimensional analog to a pixel is a voxel.\(^7\) In the three-dimensional grid, which is called a volume, every voxel can be addressed by its coordinates \((i, j, k)\). Depending on the context, a voxel might contain a scalar value, a tensor or something else. In our context, a voxel always contains a scalar value.

Eight adjacent voxels compose a volume cell.\(^8\) There is no need for this volume cell to be a perfect cube. This is due to the fact that the distance between two slices, the slice distance, is not necessarily equal to the horizontal and vertical distances in one slice (which is the pixel distance).

The volume datasets that are used in this thesis were generated by either a CT or MRI scanner (see section 1.3). In this kind of dataset, the voxels’ scalar values represent tissue densities. The patient is placed in the scanner and the grid virtually superimposes the scanned part of the patient’s body. The tissue density of the patient’s body is then sampled at the position of the voxels. The result of the scanning process is the aforementioned volumetric dataset representing tissue densities at discrete spatial positions. This dataset is then subsequently used to visualize the patient’s body [Preim07].

### 1.2.2 Marching Cubes

In order to visualize a volume dataset, different approaches are feasible. One such indirect approach is Marching Cubes. It is indirect because it creates a polygonal representation of the dataset in an intermediate step. This polygonal mesh is then rendered instead of directly rendering the dataset. To contrast both direct and indirect approaches, the Marching Cubes algorithm will be explained below.\(^9\)

**Isosurfaces** The algorithm creates polygons that represent isosurfaces. An isosurface is a contour in the volume that, for a given threshold also known as isovalue, divides the volume into an inside and an outside. All

---

\(^6\)Similar to a slice of bread.

\(^7\)The word voxel is, analogous to pixel, an abbreviation for volume element.

\(^8\)Different definitions of the term voxel are used in the literature. The definitions in this thesis are based on [Preim07].

\(^9\)Although no indirect approach is used in the following, it is still important to know the difference.
Figure 2: One of the fifteen distinct cases of the marching cubes algorithm. Voxels with a black circle are on the inside and voxels without are on the outside. The actual vertices are computed by interpolating along the edges that have one voxel on the inside and one on the outside.

Voxels with a value smaller than the isovalue belong to the inside, while all voxels whose values are greater than the isovalue form the outside. All voxels whose values are equal to (hence $iso$) the isovalue form the isosurface. By carefully choosing the isovalue, structures of interest can be visualized. In medical contexts for example, a surface representing the patient’s skin can be extracted by setting the isovalue to the tissue density of skin.

The Algorithm  The name of the algorithm already suggests that it processes cubes or rather volume cells. Each volume cell is considered individually. For each of the cell’s voxels, a decision is made as to whether it is part of the inside or the outside. Thus, each volume cell’s status indicating which of the 8 voxels are on the inside or outside can be encoded in an 8-bit index number. The $v$th bit’s value indicates whether the $v$th voxel belongs to the inside or outside. It is assumed that each edge is intersected once at most. This assumption leads to $2^8 = 256$ possible combinations of intersection between an isosurface and a volume cell. By taking into account rotation, complement and mirroring transformations this list can be reduced to 15 classes. One such case is depicted in figure 2. For each volume cell, the index determines to which of the 15 classes the volume cell belongs. In order to achieve this, a lookup table that assigns all 256 cases to their class is used. Once the volume cell’s class is determined, the intersected edges are deduced from the table. For each intersected edge the actual point of intersection, the edge vertex, is computed by linearly interpolating between the two voxels that form the edge. Since the lookup table also contains information about the triangulation for each class, the final triangulation for the concrete volume cell can be composed of the edge vertices.
Additionally, if shading is desired, a normal has to be computed for each vertex. The normal is computed by linearly interpolating the adjacent voxels’ normals. After the volume cell has been processed, the algorithm proceeds with (or “marches” to) the next volume cell in memory. Since each volume cell is considered individually, vertices might be duplicated for adjacent cells. These duplicates can be removed by a postprocessing step that merges coincident vertices [Schroeder05]. After all volume cells have been processed (and, eventually, the vertices been merged), the result of the algorithm is a polygonal mesh representing the isosurface of the volume for the chosen isovalue.

Problems Unfortunately there are ambiguities when it comes to distinctively classifying the different volume cell cases. For some cases, different (but still valid) triangulations exist. These ambiguities can lead to holes in the surface. One solution to this problem has been proposed in [Shirley90], where the authors suggest decomposing the volume into tetrahedra instead of cubes. This decomposition results in only four unambiguous cases, but produces about twice as many triangles, which results in higher rendering times [Preim07].

1.2.3 Ray Casting

Another volume rendering technique that directly visualizes the volume dataset is called Ray Casting. This approach does not involve an intermediate step to generate polygonal data. As we will see, the concept of the image plane (i.e., the final image that contains the pixels) is an integral part of the algorithm. Consequently, Ray Casting is referred to as an image-order volume rendering technique. Before presenting the actual idea of the algorithm, two important concepts will be explained.

Transfer Functions Since the scalar tissue values stored in the voxels do not have a visual meaning by themselves, optical properties (e.g. color and opacity) need to be assigned to them. Instead of assigning these properties to each voxel individually, many voxels are assigned the same properties at once based on their scalar value. For this, utilization of so-called transfer functions (TF) for color and opacity is commonplace (figure 3).

---

10 Since no such thing as voxel normals usually exists for volumetric datasets, they have to be approximated. Often, this is achieved with gradients, which are commonly computed as the central difference, intermediate difference or with the Sobel operator on neighboring voxels.

11 For example, this happens when different triangulations are chosen for two adjacent volume cells that share a face where an ambiguous case is present.
Figure 3: In medical direct volume rendering applications, a transfer function is defined by the user to assign certain tissue densities an opacity. Between the defined points, the values are interpolated. TFs for color mapping can be defined analogously. The concept of Hounsfield units is explained in section 1.3.5.

In medical contexts, these are usually piecewise functions. They simply map a range of voxel values to a color and an opacity. This way, regions of interest can be visually accentuated and the appearance of less interesting regions diminished. The focus region, e.g. the teeth, have a higher tissue density than soft tissues such as skin. Thus, to highlight the teeth, the opacity transfer function is adjusted in such a way that the tissue densities of the teeth are mapped to a high opacity, while all other densities are set to a much more transparent value.

A TF consists of a few control points that set opacity and color for a specific tissue density. For any tissue density between two such points, color and opacity are linearly interpolated. This significantly reduces the amount of control points required, therefore easing TF creation. Defining a transfer function that causes the visualization to realistically mimic the real object is a nontrivial task. Due to their importance especially in direct volume rendering techniques such as Ray Casting, transfer functions are an integral element in most volume rendering applications.

**Trilinear Interpolation** The volume data structures that were introduced above are not a continuous representation of the original object, but rather only contain information about it at discrete grid points. However, it is often desirable to access values between these points. In order to provide a reasonable value in these situations, *trilinear interpolation* is commonly employed (figure 4).

The four edges along a given direction (x, y, or z) are considered. For each edge, linear interpolation between its two voxels weighted by the desired point’s coordinate in the edge’s direction is performed. Two faces of the volume cell now contain two interpolated points, respectively. Between
Figure 4: Trilinear Interpolation. First four linear sample points are created. From these, two bilinear points are interpolated. Interpolating these two points again yields the desired trilinear sample point. Image by [Preim07].

those two points of a face, another linear interpolation is performed, again weighted by the desired point’s coordinate. Two points are the result of this bilinear interpolation. In a last step, these two points are again weighted and linearly interpolated, finally yielding the desired point’s value.

While computationally more expensive, this approach produces much better results than naively returning the value of the voxel that is closest to the desired point.

The Algorithm Ray Casting is referred to as a direct visualization approach because it does not require intermediate steps such as a polygon creation phase. Instead, the image resulting from the visualization process is directly deduced from the volume dataset. As suggested by the name, rays are cast from the viewpoint through the pixels of the image plane and through the volume (see figure 5). Along each ray, the scalar values of the dataset are sampled. The distance at which samples are collected for each ray, the sampling distance, can usually be adjusted. Since the sampling points for the rays do not coincide with the voxels of the volume in most cases, the need for trilinear interpolation now becomes evident. A shorter sampling distance therefore leads to more interpolations and increases computational cost, but at the same time reduces staircase artifacts.

The color and opacity that belong to each sample’s scalar value are looked up from the transfer functions. The colors are summarized until an opacity value of 1 is reached. As soon as this happens, early ray termination can take place, meaning no further samples need to be collected for the ray. If an opacity value of 1 cannot be reached until the ray leaves the volume, the background color is used to fill up the remaining opacity.
Figure 5: Rays are cast through the pixels of the image plane. The rays pass through the volume. At certain distances, the volume's values along each ray are sampled. All samples along one ray are then accumulated, resulting in the pixel’s color value.

The amount of rays that are cast per pixel greatly influences the visual quality and rendering performance of the resulting image. While Marching Cubes is usually faster after the polygon creation phase, adjusting the isovalue is cumbersome. With Ray Casting, the user gains fine-grained control of the visual representation of the volume at the cost of constantly computing rays for changing perspectives [Preim07]. Figure 6 compares the results of an indirect and a direct volume rendering algorithm.

Figure 6: Figure a) shows an isosurface rendering. Figure b) depicts the same dataset being rendered by a raycasting algorithm. By choosing an appropriate TF for the raycasting algorithm it is easily possible to visualize other voxel values as well.
1.3 Medical Imaging Modalities

Currently, several different *imaging modalities* exist in medicine. The term *modality* denotes a (hardware) sensor device that is used to create a digital representation of the scanned object. Such modalities (and subsequently, the presentation of the data generated through them) offer the physician a look inside a patient’s body without surgery. A two or three-dimensional representation of the patient’s body can be generated in minutes. The low amount of time required, the quality of the images and the relatively low risks make these modalities a very important tool in diagnostics, therapy planning, interoperative navigation and postoperative monitoring. The following three sections will give an overview of Computed Tomography and Magnetic Resonance Imaging, because both techniques can generate the volume datasets that are needed for volume rendering. To understand Computed Tomography, one has to understand X-ray imaging first. X-ray imaging is therefore explained in section 1.3.1 as well (Further information about these and other medical visualization techniques can be found in [Preim07].).

Godfrey Hounsfield, the inventor of Computed Tomography, was awarded the Nobel Prize for Physiology or Medicine in 1979. Paul Lauterbur and Sir Peter Mansfield were awarded the Nobel Prize for Physiology or Medicine in 2003 for their work that made the development of Magnetic Resonance Imaging possible [Preim07].

This clearly underlines the importance of modern tomographic imaging for medicine and motivates further work in the field.

1.3.1 X-Ray Imaging

![X-Ray Scan](image1)

![CT Scan](image2)

**Figure 7:** Schematic overview of X-ray and CT scanners. Emitters cast X-ray quanta through an object. The radiation is scattered, absorbed or passes through the object onto the detector. For CT, an emitter/detector pair rotates around the object to be scanned.

A very important tool in diagnostics today are *X-ray* scans of a patient’s body. X-ray scans were the first medical image modality that offered the
physician a look inside a patient’s body without surgery. The ‘X’ in the name refers to the fact that the kind of radiation used in the process was unknown at the time when they were discovered by Wilhelm Conrad Röntgen in 1895. X-ray images are based on ionized radiation. X-ray quanta are emitted by a sender into the direction of the object to be scanned (figure 7). The quanta then travel through the object. Depending on the structure and physical composition of the scanned object, the quanta are either absorbed by the object, scattered or pass through the object. The part of the radiation that was not absorbed is then recorded by a film behind the object. In the case of a human being scanned, the different types and densities of tissues, as well as their thickness, result in differently attenuated X-ray beams. The higher the density or the thicker the tissue, the less radiation reaches the film. Less radiation then leads to a brighter region in the respective part of the image (figure 8). In [Preim07] an approximation of the radiation’s attenuation $S$ is defined. In equation 1 $\mu_i$ denotes the attenuation characteristics of the $i$th material of the scanned object. Analogously, $d_i$ denotes the distance the radiation has to travel through that material.

$$S = \sum \mu_i * d_i$$

(1)

Upon completion of the scanning process, a two-dimensional image representing the accumulated tissue densities (in the direction of the emitted radiation) can be examined by the physician. This is also referred to as the intensity profile [Preim07].

1.3.2 Computed Tomography

The invention of Computed Tomography (CT) was the next step in the field of medical imaging. CT scans address a major disadvantage of simple X-Ray scans: An individual X-Ray image cannot be used to correctly comprehend the spatial relationships of the patient’s body. This is due to the fact that an
X-ray image is only a two-dimensional representation of a three-dimensional object and often several structures are indistinguishably superimposed. A CT dataset consists of a series of X-ray images. The emitter and detector rotate around the object to be scanned in a spiral or helical motion. The object is then moved continuously in a direction perpendicular to the direction of the rotation. At the same time, several ordinary X-ray images are taken by the rotating emitter and detector from different positions. The slice images of the volume dataset are then computed from the intensity profiles of a full rotation. Using Radon transform, the slice images of the volume dataset are computed from the intensity profiles of a full rotation. The resulting dataset reveals anatomic information and allows the physician to examine the spatial relationships of tissues (based on their density) more accurately than an ordinary X-Ray image does [Preim07].

1.3.3 Magnetic Resonance Imaging

![Figure 9: Two MR images. The left and right images depict T1 and T2 relaxation times, respectively. Martin Skalej, University of Magdeburg](image)

*Magnetic Resonance Imaging* (MRI) does not expose the scanned object to potentially harmfully radiation like CT does. The patient is placed inside a strong magnetic field. This magnetic field causes the large number of hydrogen atoms inside the human body to align themselves parallel (or anti-parallel\(^{12}\)) to the magnetic field. Inside the magnetic field, the atoms spin arbitrarily. A radio frequency pulse is then turned on. The pulse causes the atoms to rotate into an orientation perpendicular to the magnetic field and spin synchronously (in *phase*) around the same vector. The rotation around 90 degrees leads to a zero z-component of the rotation vector. Since the atoms spin around the same vector, they also spin in the same direction. After the atoms have been stimulated (i.e., aligned to spin uniformly) and the radio frequency pulse has been turned off, they begin to restore their original alignment parallel to the magnetic field. This relaxation happens in

\(^{12}\)Anti-parallel still means parallel, but going into the opposite direction.
two steps, namely the T2 and the T1 step. During the T2 step, the atoms dephase and stop spinning around the same vector. During the T1 step, the atoms also return from their perpendicular position (to the magnetic field) and align themselves parallel to it again. The key to success is the fact that different tissues have different T1 and T2 relaxation times. This is due to the fact that different tissues have different densities of hydrogen atoms. Thus, actually, the density of hydrogen atoms is measured (The difference of T1 and T2 relaxation times is depicted in figure 9.). During the relaxation phase, the atoms release energy that they received when they were stimulated with the radio frequency pulse. By measuring that energy and the relaxation times, different tissues can hence be distinguished.

MRI data acquisition heavily depends on the presence of a sufficient amount of hydrogen atoms in the scanned structures. Therefore, MRI does not provide a good representation of structures such as bones. CT on the other hand cannot properly distinguish between different soft tissue types. CT and MRI should thus not be seen as competitors, but as complementary imaging modalities. Since soft tissues contain high amounts of hydrogen nuclei, MRI is particularly suited for neuroimaging. While MRI does not use harmful radiation, it still may have harmful effects on patients with metal implants or cardiac pacemakers due to the strong magnetic field and the radio frequencies used [Preim07].

1.3.4 Storage of Medical Image Data

In order to govern the exchange of medical image data in such a way that a dataset acquired with any imaging device can be identically displayed on any appropriate output device, a unified storage format is mandatory. The DICOM (Digital Imaging and Communications in Medicine) industry standard specifies such a common format that most image acquisition and display devices in medicine conform to. The DICOM standard is a vast document that regulates many different usage scenarios. For this thesis, however, the only important aspect is the structure of the storage of CT and MRI data. CT and MRI scanners produce a set of DICOM files for each scanning process. Each individual file contains an image that represents one slice of the volume. Additional information such as the patient’s name and age, as well as the ID of the study and parameters of the imaging device are added to these files. In order to generate a three-dimensional volume, the individual slice files that belong together have to be examined. Although the slices of one dataset are usually comfortably stored in the same folder, this

---

13 According to [Preim07], T2 relaxation is in the order of a few milliseconds, while T1 relaxation is in the order of a second.
alone is no sufficient criterion for grouping. Instead, the image parameters have to be identical. This is done so as to avoid confusion of datasets [Preim07]. For the purpose of testing, datasets that were acquired by a CT scanner have been used for this thesis.

1.3.5 Hounsfield Density Units

The volumetric datasets created by the aforementioned modalities represent tissue densities at discrete sampling points. The tissue density is represented by a scalar value, which is normalized into Hounsfield Units. This density scale ranges from $-1000$ for air over 0 for water up to 3000 for bones, with several overlapping regions, e.g., for heart and kidney, in between. For the purpose of storing the value, shifting and scaling operations might be applied\textsuperscript{14} [Preim07].

\textsuperscript{14}For example, VTK stores the values in the range 0.0 to 255.0 as floating point numbers.
2 Hardware

Different technical approaches to building multi-touch hardware exist. For this thesis, a multi-touch table based on optical sensing was built. The advantages of optical sensing over other approaches are that it is relatively easy to build and scales quite well.

In the following, an overview of optical-based solutions will be given. The usual camera image acquisition and finger extraction pipeline will be explained. In particular, information about the multi-touch table built for this thesis will be provided. For the sake of simplicity, the assumption is made in the course of this document that only touches produced by the user’s fingers should be sensed [Rishi09].

2.1 Principles of Optical Multi-Touch Systems

All optical multi-touch systems have several things in common. They almost always use a simple sheet of translucent glass or acrylic as a touch sensing surface. One or more cameras facing the touch surface \(^{15}\) register the objects (usually just fingers) that hover in front of the surface or touch it directly. Since the use of markers to identify fingers is not desirable, other means to better discriminate finger tips from background must be utilized. This is usually achieved with sophisticated means of illumination. The light that illuminates the fingers is then reflected back towards the camera, which is key to success here. The main difference between the three optical solutions explained hereafter is the approach used for illumination.

All of the discussed methods can be employed in tabletop or wall mount scenarios. Even complex forms such as spherical displays can be built [Benko08]. In the following, usage in a tabletop scenario will be assumed without loss of generality [Rishi09].

2.1.1 Utilization of Infrared Light

Although there certainly are situations in which only a multi-touch sensor is required, it is often desired that the computer’s visual response is right at the finger tips. In such situations, two visual components, namely a projector (or computer screen) and a camera, are used. The camera is then adjusted to be sensitive to only a certain spectral band, e.g. human-perceivable red. Light in this spectral range is then used to illuminate only the user’s fingers but not the background, thus leading to easier segmentation of the camera image. It is problematic, however, to use a human-perceivable color for this, since the projector also operates in the visual band. To avoid interference,

\(^{15}\)In some setups, the cameras do not face the actual touch surface, but a mirror that shows the touch surface. This is done so as to decrease the distance between camera and touch surface.
it is therefore desirable to operate them in different spectral bands. Since the projector’s band obviously cannot be changed, infrared (IR) light is usually utilized for illumination. Using IR light has the added benefit of being invisible to the user. The CCD and CMOS sensor chips used in many cameras, however, are in fact sensitive to IR light and can therefore be used\footnote{While the sensors themselves are sensitive to IR light, a filter that blocks the IR range often covers the sensor for noise reduction. This filter needs to be replaced by an IR bandpass filter.}. Accordingly, infrared light is often used for the purpose of illumination [Rishi09].
Figure 10: In rear DI setups, IR light illuminates the user’s fingers from behind of a translucent material. The light is reflected by the finger and captured by the camera. Image courtesy HANSEN, THOMAS

2.1.2 Diffuse Illumination

Diffuse Illumination (DI) is perhaps the easiest method of illumination to understand from a technical point of view. It is used in popular systems such as the reactTable [Kalten06] or the multi-touch Sphere [Benko08].

In its simplest form, light (often just ambient light) shines at the touch surface. When an object advances towards the touch surface, it intersects the path between light source and touch surface, causing it to cast a shadow onto the touch surface. The closer the object comes, the darker the shadow becomes because the ambient light sources cannot illuminate the object anymore. This is the only technique that makes use of shadows instead of direct illumination of the fingers. Furthermore, this front DI setup is particularly well-suited in situations where there is much ambient light. Other solutions (like 2.1.3 and 2.1.4) suffer from ambient light noise, resulting in impaired object recognition. On the other hand, in front DI setups this only improves the contrast\textsuperscript{17}.

Another DI approach uses IR radiators to illuminate objects in close proximity of the touch surface (figure 10). Usually, the light is shone through the touch surface from behind it and reflected towards the camera by the fingers. As with front DI, the closer an object comes to the surface, the easier it is to distinguish it from the background\textsuperscript{18}. The main difference to front DI is that no shadows but bright objects are extracted. This way, not only fingers but also markers can be detected and identified, even when they lie directly

\textsuperscript{17}As long as the light is diffuse enough.
\textsuperscript{18}This is due to the higher amount of light illuminating it.
on the table. There are two major concerns with DI. Since not only fingers actually touching the surface, but also fingers hovering above it reflect the IR light, it is difficult to distinguish between touching and hovering. In applications that do not take hovering objects into account (as is the case for the vast majority of applications), this results in many false detections and hence confusing behaviour of the application. While the number of false detections can be reduced by careful configuration of the image processing pipeline, it is difficult to completely eliminate them. Additionally, even (i.e. diffuse) illumination is difficult to achieve due to the non-planar radiation of the light sources. This is undesirable since it produces differently bright objects. To increase diffusion of the light emitted by the radiators, a diffuse material (e.g. thin translucent paper) is often added between the fingers and the radiators. Additional calibration steps can further aid in achieving equally bright objects at any position on the surface [MT09; Rishi09].
Figure 11: Due to the effect of total internal reflection, the light is trapped inside the acrylic unless another material frustrates the TIR effect, causing FTIR. Image courtesy Hansen, Thomas

2.1.3 Frustrated Total Internal Reflection

In [Han05], the author suggests a low-cost solution for building multi-touch surfaces. The well-known physical effect of total internal reflection (TIR) is exploited to highlight objects that come in direct contact with the touch surface (figure 11). These bright luminescent objects (BLOBs) can then be extracted from the camera image after it has undergone several image processing steps.

TIR is an optical phenomenon. It occurs when light from one medium (e.g. glass, acrylic) attempts to enter another medium (e.g. air) with a lower refractive index. Depending on the light’s angle of incident, it is refracted to a certain degree. If the angle of incident goes beyond a critical angle, it is even totally reflected back into the original medium (i.e., reflected internally, hence the name). Thus, the light is effectively trapped inside the medium (which is effectively an optical waveguide) it tries to leave.

When another material, such as a finger, makes contact with the medium the light is trapped in, the light at the area of contact is no longer subject to TIR. The object is then said to frustrate the effect of TIR (FTIR). Due to the frustration, the light leaves the medium and illuminates the finger (or any other object that can trigger the TIR effect).

Using IR LEDs, the touch surface is edge-lit with IR light. When the user triggers the FTIR effect, light bounces off the finger tips and is reflected back towards the camera. Since the light only leaves the waveguide in the area of frustration, the fingers appear much brighter than the background and can be segmented relatively easily. Objects in close proximity of the surface (but without actual contact) do not cause the FTIR effect, thus almost no false detections occur.
The finger recognition quality of FTIR heavily depends on how wet one’s fingers are. The greasier the finger tips, the better the recognition. It is difficult to make good contact with the touch surface with dry fingers. In order to overcome this problem, the author suggests placing another \textit{compliant} layer on top. This surface has to have a higher refractive index than the glass or acrylic. It also needs to have good coupling properties with the waveguide when pressed. Research for a perfect compliant material is still ongoing [Han05; Rishi09].
2.1.4 Diffuse Surface Illumination

By using a special acrylic with the product name *Endlighten*, the FTIR and DI approaches can be combined (figure 12). This hybrid solution is very similar to FTIR in that it also makes use of an edge-lit sheet of acrylic. However, tiny particles have been embedded in the acrylic. The particles can’t be seen by the human eye. They act like mirrors and cause some of the IR light to leave the acrylic, hence partially disturbing the TIR effect. Depending on the amount of particles and light, this can be exploited to achieve an evenly distributed and diffuse illumination of the surface (DSI). While the FTIR effect is still present in DSI setups (since not all light is scattered), DSI also allows usage of markers that would not couple well enough with FTIR-only surfaces. On the other hand, the consequence is lower contrast of touches. In addition to this technical concern, higher costs have to be taken into account due to the relatively high price of Endlighten [Rishi09].

2.2 Documentation of Hardware Assembly

The primary objective of this thesis is the implementation of a multi-touch volume rendering application. In order to properly test the program, multi-touch capable hardware is required. Due to the lack of affordable ready-made devices and the interesting challenge, it was decided to build custom hardware. In the following, the components required for the installation will be presented. Final results will be provided in section 4.1.2.
Figure 13: Ninety IR LEDs were put through holes that had been drilled into an L-shaped aluminium frame. After soldering them, they illuminate the sheet of acrylic from the edges.

2.2.1 Hardware Components

After evaluating the options, the DSI approach was chosen due to the possibility of tracking markers\(^{19}\) and a considerably lower rate of false detections. Exploiting the scalability of the approach while still remaining within the realms of possibility, the pure diagonal of the touch surface was set to about 40 inches. In the following, the components as well as the build process will be presented.

**Illumination**  
Achieving a good illumination is the first of several crucial steps towards a usable device. The special acrylic (Endlighten XL) needed for the DSI approach was purchased with a size of \(800\text{mm} \times 600\text{mm} \times 8\text{mm}\). Four L-shaped aluminium bars were screwed together to form a frame that holds the acrylic on four of its faces. Ninety holes were drilled into the remaining four faces. Ninety IR LEDs (OSRAM SFH-485) at a wavelength of \(880\text{nm}\) were put into these holes and were soldered with resistors to form a closed electrical circuit (figure 13). Using a power supply unit, the circuit can be connected to a socket. While ready-made IR LED strips are available, this self-made solution is much cheaper.

\(^{19}\)While marker tracking is not required for the volume rendering application, it might be interesting for further projects that go beyond this thesis.
At the very end of the entire table’s construction process, another four aluminium bars were added on top, covering the LEDs. These prevent the light leaving the side of the LEDs to illuminate objects at the borders or in close proximity of the surface of the table, such as the user’s palms. This addition led to a much lower rate of falsely detected objects.

**Camera**  In order to track the IR BLOBs, a camera is required. Due to the comparatively low price and satisfactory performance, the Sony Playstation 3 Eye is a popular choice among hobbyists. In theory, the camera is able to capture up to 120 frames per second (fps) at a resolution of $320 \times 240$ pixels, or 60fps at $640 \times 480$ pixels. It is connected to the computer via USB and employs a CCD chip as a sensor. An IR blocking filter and a two-stage lens is mounted in front of the CCD.

No official software drivers are available from Sony, which is a major disadvantage of this device. Unofficial drivers, however, are freely available for Linux, Mac OS X and Windows. While the Windows driver does support all update rates and resolutions, its code is kept secret and crashes are common at the time of writing. On the other hand, the Linux and Mac OS X drivers currently suffer from being limited to a lower frame rate.

The IR blocking filter that sits between the lens and the CCD has to be removed so as to not block the IR BLOBs, because they are of particular interest for the subsequent processing steps. Since the only wavelength of interest is that of the IR LEDs, a bandpass filter can be used to weaken all other wavelengths. As a result, only the wavelength of the IR LEDs $\text{20}$ (in a range of tolerance) reaches the sensor, which simplifies image processing.

Replacing the filter involves a full disassembly of the camera, which voids the warranty.

The bandpass filter is no perfect replacement for the IR blocking filter due to its size and shape. This negatively impacts the camera image in that it becomes rather blurry.

To solve this issue, the original lens has to be replaced as well. A standard M12 mount was screwed onto the camera’s circuit board and a fitting M12 lens inserted. The bandpass filter also fits into the mount between the lens and the CCD chip. Since the M12 lens and mount are threaded, they allow for continuous focus adjustments as opposed to the original lens’ two-step focus.

---

$\text{20}$In addition to the light emitted by the LEDs, ambient light with the same wavelength is sensed as noise as well. For example, the light emitted by the sun also emits that wavelength, as well as several types of indoor illumination such as ordinary light bulbs and halogen lamps.
Figure 14: Camera and projector. Both are held by self-made mounts and face upwards.

Figure 15: A raw frame as captured by the camera. Only the camera’s brightness and contrast were adjusted. Ten fingers touch the surface.
Another drawback of the camera at the time of order of the components was the lack of official technical documentation. In particular, no sensor response curve indicating the sensor’s sensitivity for certain wavelengths was available\textsuperscript{21}. Nevertheless, for hobbyist constructions, where cost rather than quality is a concern, this camera including all modifications is still cheaper than a ready-made solution from special-purpose vendors.

As the camera is supposed to be used in front of the users rather than below them, the camera cannot be oriented to view upwards by itself. A mount that supports this has been built and proven to greatly simplify orientation of the camera (figure 14). After the illumination was functional and the camera modified, a first test confirmed the basic feasibility. With the naked eye, the illumination could not be seen. The camera image, however, revealed the reflection of infrared light by the fingers while touching the acrylic (figure 15).

**Image Projection** With the illumination in place, another layer for displaying the screen image can be added. This can be achieved by either disassembly of a flat-panel screen or utilization of a projector and projection material. The latter approach was chosen for this thesis (figure 14). To avoid shadows, the image is projected from the rear (i.e., the projector is placed inside the table). When using a projector, a projection layer to project the image onto is required. The acrylic itself cannot be used as projection surface due to its light transmission rate of 91\%\textsuperscript{22}. As projection material, a rear projection foil (Rosco Gray) was used. The acrylic is not as easily damaged as the foil. The acrylic was hence put above the projection material, with another layer of glass below the foil for stabilization.

**Computer** Any combination of computer hardware and operating system can be used to process the video stream from the camera as long as performance is sufficient. Fast graphics hardware is additionally required for many client applications. For this thesis, development was mostly done on a laptop (Intel Core 2 Duo, 2.53GHz, NVIDIA 9400M graphics chip, Mac OS X). A dedicated computer has also been set up (Intel Core 2 Duo 2, 2.13GHz, NVIDIA GeForce 8600 GT, Windows 7).

A functioning computer concludes the list of essential components for the multi-touch screen. What follows hereafter is either software deployment or optional hardware improvements.

\textsuperscript{21}By now, unofficial charts are available, indicating that the sensitivity of the CCD for light at 880nm is not ideal.

\textsuperscript{22}By using only the acrylic, most of the light would just pass through to the ceiling.
Calibration 

Touch input can then be sensed and the computer image projected. It is not yet possible, however, to tell what part of the projected image the user is touching. To achieve this, a correlation between sensed touch points and projection must first be established. One practical procedure involves the projection of a grid of calibration points (figure 16). One after another, the individual calibration points of the grid are highlighted by the projector. The user then solely touches the highlighted point. The active grid point’s on-screen coordinates and the touch position sensed for this touch are stored and the system proceeds with the next grid point. After all points have been processed, the grid is triangulated. When a touch occurs inside a triangle, its screen coordinates are computed by interpolating between the three known points of the respective triangle. Every touch outside the calibration grid (i.e., not within a triangle) is then usually discarded. It is therefore desirable to maximally exploit the resolution of the camera’s CCD chip. After the calibration process, touch positions in camera coordinates can be converted to screen coordinates which can be understood by the client application.
Figure 17: The result of the hardware construction phase. The table’s dimensions are $1200\text{mm} \times 800\text{mm} \times 900\text{mm}$ and the touch surface has a size of $800\text{mm} \times 600\text{mm}$.

Table Framework In order to stabilize the construction, a wooden table framework was built and the aforementioned components embedded (figure 17). The full size of the tabletop is $1200\text{mm} \times 800\text{mm}$, of which $800\text{mm} \times 600\text{mm}$ are dedicated to the touch surface. At a height of $900\text{mm}$, the table can be used comfortably while standing. On the front face, room for the user’s feet was left. Enclosing a projector in such a table produces heat which has to be dissipated. This was achieved by working two fans into the left and right faces, respectively. The fan’s speed can be adjusted using a slide control. To be on the safe side, a thermometer constantly monitors the temperature inside the table. All of the lateral covers (except the back cover) can be easily removed with a knob to allow for further internal adjustments should the need arise [Rishi09].
3 Software

In the course of this thesis, a volume rendering application with multi-touch user interface (working title: BoneTouch) was designed and implemented. The application builds upon already existing software libraries and toolkits. In order to function properly, the application needs to operate in conjunction with an additional set of applications and protocols. On top of these software dependencies, the BoneTouch user interface was implemented. After giving an overview of the software libraries BoneTouch depends on, the actual design and implementation will be discussed.

3.1 Software Dependencies

BoneTouch is written in the Python programming language 3.1.1. Figure 20 depicts the correlation of BoneTouch and its dependencies. The PyMT 3.1.2 toolkit is used for the creation of the user interface. The VTK 3.1.3 library is responsible for reading DICOM files from disk and rendering them. A tracking software (CCV or Movid) 3.1.4 segments finger tips from background in the camera video stream. The location, size and shape of these segmentation results is then sent to BoneTouch over the TUIO protocol 3.1.5. PyMT interprets accepts this TUIO network stream and transforms it into input events for the application, which then, potentially, changes its state (and that of the VTK rendering) and renders its output. The graphical output is presented to the user on the multi-touch surface and allows interaction again, hence performing a cycle. An introduction to all required components will be given in the following.
3.1.1 Programming Language: Python

For BoneTouch, the decision to use the Python programming language was made because all required software libraries can be interfaced through Python Application Programming Interfaces (APIs).

Python is a popular interpreted programming language. It is freely available under the terms of an open source license. Its interpreter is available for all major platforms. The language allows for object-oriented programming. Due to its relatively clear syntax, dynamic typing and interpreted nature it can be used for rapid prototyping.

Many applications and libraries (even those written in C or C++) offer Python bindings, which makes them useful for Python applications. Parts of a program that are critical with respect to execution speed can hence still be implemented in C or C++, without giving up the benefits of a dynamic language for the rest of the application.

It is noteworthy that, strictly speaking, the name Python only refers to the language, not the interpreter. Several Python interpreters exist, with CPython (the reference implementation written in C) being the most widely used option [Python].
Figure 19: The PyMT eventloop consists of three steps. First, input events are collected from all input providers. These are then dispatched as touch events to the root node of the scene graph and either propagated or consumed. Lastly, all widgets are redrawn.

3.1.2 Multi-Touch Input and User Interface: PyMT

Most software libraries for user interface construction are based on the WIMP (Window, Icon, Menu, Pointing Device) metaphor. For multi-touch applications, this metaphor is not really adequate due to the indefinite number of touches and their much lower pointing precision.

A toolkit explicitly tailored to the requirements and potential of multi-touch interfaces is PyMT. PyMT is available freely for all major operating systems under the terms of the LGPL open source license. As the name suggests, PyMT is written in Python and allows the creation of multi-touch applications.

When a PyMT application starts, a window is opened. The application can draw into this window using standard OpenGL calls. User interface elements are so-called MTWidgets, which are multi-touch enabled versions of the widgets known from other user interface toolkits. The widgets are arranged in a scene graph with one widget being the root node. PyMT supports several different input devices, such as a mouse, a Wiimote or input via the TUIO protocol. All of these input providers implement an abstract base class TouchProvider and emit objects of a subclass of the abstract base class Touch. All Touch objects contain several attributes such as touch position \((x, y)\), touch state (down, move, up) and a unique ID. Depending on the input provider, additional attributes such as shape and rotation might be defined.

When the PyMT main loop begins, Touch objects are collected from all in-
put providers (figure 19). The touches are then dispatched to the root of the scene graph. Depending on the state of the touch, the object is passed to the widget’s on_touch_down(), on_touch_move() or on_touch_up() method. If a widget does not consume the touch (by returning True), the touch is propagated to the widget’s child widgets using depth first traversal. After all input events have been processed, the main loop issues draw events to the scene graph. Every widget can now draw itself using OpenGL. After the drawing phase, the current iteration of the main loop ends and a new iteration commences.

For BoneTouch, PyMT is used to draw the user interface and accept user input [Hansen09].

3.1.3 Volume Rendering: VTK

Volume rendering is a nontrivial task. It was therefore decided to use an already available solution. The Visualization Toolkit (VTK) offers a plethora of modules ranging from data readers over image processing classes to three-dimensional volume rendering algorithms. VTK is freely available as open source and runs on all major operating systems. All of the data reading and volume rendering algorithms required for BoneTouch are already implemented for VTK. Fortunately, VTK also supports scripting with Python and uses OpenGL for graphical rendering.
3.1.4 Tracking: CCV et al.

In order to get meaningful information about the user’s touches, it is necessary to process the video stream from the camera sensor. To do this, a special software is employed. The purpose of this tracking application is to differentiate between finger touches and background in the input image. This is achieved by passing the raw input image to an image processing pipeline. The output of the tracking process, after interpreting the processed image, is a list of touch positions. A typical image processing pipeline for this involves thresholding, smoothing, amplification and background subtraction. In addition to finding BLOBs in a single image, the tracking application needs to track (hence the name) the found BLOBs over subsequent frames of the video stream. Since the BLOBs found by the tracking application directly translate into input events for the client application, the quality and performance of the segmentation and tracking algorithms greatly affect the responsiveness of the user interface.

Tracking BLOBs over several frames until they vanish allows interpreting them as single touches (similar to a single-click), touch movements (useful for dragging operations for example) and even gestures. In particular, the tracking application has to decide if a BLOB in frame $n$ and another BLOB, potentially at a different position, in frame $n+1$ is still the same BLOB or a different one.

Several different tracking applications are discussed here.

**Community Core Vision**  
*Community Core Vision (CCV)* is a popular choice for hobbyist multi-touch installations. It is available freely for all major platforms. CCV employs a fixed image processing pipeline. In the user interface, parameters for the filters of the pipeline can be controlled with sliders. Filters for (dynamic) background subtraction, high-pass and low-pass, as well as signal amplification are supported.

Due to the fixed pipeline the user only has to adjust the parameters of the filters. This makes CCV easy to use even for image processing novices. For

---

23Potentially, additional information such as shape, size and orientation can be added.
users familiar with image processing, the fixed nature of the pipeline may be too limiting. CCV does not offer marker tracking [CCV].

**reactTIVision** For some applications it is necessary to not only track fingers, but fiducial markers as well. The *reactTIVision* tracking application has built-in support for marker tracking. It is, for example, used to track the individual tone-generating objects used within the reacTable project. As CCV, *reactTIVision* does not support altering the image processing pipeline. Furthermore, no real graphical user interface is available. Since the application’s focus is marker tracking, only rudimentary support for finger tracking is available [Kalten07].

**Movid** In an effort to overcome the limitations of the already available tracking solutions, the *Movid* (Modular Open Vision Interaction Daemon) project was initiated. In contrast to CCV and *reactTIVision*, Movid gives the user fine-grained control over the image processing pipeline. Input, output and filter modules can be connected and their parameters configured via a graphical front-end. Movid offers finger and marker tracking. Movid is still under active development and no stable version has yet been officially released [Movid].

### 3.1.5 Touch Information Transmission: TUIO Protocol

The *TUIO* (Tangible User Interface Objects) protocol has become the de facto standard for communication between the multi-touch input layer (tracking application) and the application layer. All of the aforementioned tracking applications support the TUIO protocol. By sending the tracking results over the network, the protocol makes it possible to separate tracking and client application. TUIO data messages are based on the *Open Sound Control* format and can be sent over protocols like TCP and UDP. Potentially, multiple TUIO streams can be multiplexed at the client side. TUIO uses different types of messages to inform the client about the tracking results. A unique ID is assigned to every object found by the tracker. **ALIVE** messages inform the client about all objects currently present on the touch surface by sending the IDs of the objects. **SET** messages inform the client about a specific object’s state, such as its position in normalized coordinates, velocity and acceleration. An optional **SOURCE** message can be used to identify the host and application the message was sent from. These messages are packed into a bundle and sent to the client application. Additionally, each bundle is given a sequence number in order to uniquely tag the bundle. According to

---

24. A marker is a uniquely identifiable symbol attached to a physical object in order to locate and identify that object in the image more easily.

25. Although the author has been involved in the application’s creation, the fact that it is unrelated to this thesis is explicitly stressed.
the specification, a typical TUIO stream consists of many message bundles of the following form \(^{26}\):

```plaintext
/tuio/2Dcur source application@address
/tuio/2Dcur alive s_id0 ... s_idN
/tuio/2Dcur set s_id x_pos y_pos x_vel y_vel m_accel
/tuio/2Dcur fseq f_id
```

The client application then interprets the TUIO stream and deduces information about the currently active objects and their properties [Kalten05].

### 3.2 Related Software: OsiriX

Applications that read and visualize DICOM datasets already exist. However, they are controlled with a mouse. While utilization of a mouse is not a bad thing per se, the hypothesis is that some nontrivial key operations can be carried out more easily with corresponding multi-touch gestures. The open source software OsiriX is used to evaluate this hypothesis hereinafter.

**User Interface** OsiriX employs a database-driven approach to store patient profiles. The application starts with a window that allows adding new or opening already existing patients. A patient profile is opened by double-clicking it. The images of the dataset can now be examined. Advanced visualization techniques such as surface rendering and direct volume rendering (cf. sections 1.2.2 and 1.2.3) are available as well. The following discussion will be based on the user interface design of the direct volume rendering capabilities.

When the application enters direct volume rendering mode, the user interface initially consists of two parts. The upper part of the screen displays the available features, such as rotation, scale and cropping. The space that remains below is covered by the actual rendering viewport. All operations that can be used to directly alter the volume follow the same principle: The current mouse button function has to be set to one of several states, such as rotation or scale. The cursor then has to be positioned on the rendering widget and by moving it\(^{27}\), the user determines the rate at which the visualization is changed while the mouse button is pressed. In order to carry out any operation on the volume, the user first has to set the aforementioned mouse button state appropriately.

A transfer function widget for color and opacity, which is not shown by default, can be displayed at the bottom of the window by pressing a button. The widget allows for manipulation of individual points of the TF and adjustments of the entire function. The abscissa represents the tissue density

\(^{26}\)This is of course only a human-readable representation. The actual stream is encoded into a binary format.

\(^{27}\)Some states distinguish between top to bottom and left to right movements.
values in Hounsfield units, while the ordinate indicates the opacity of the points. In addition to the opacity, each point also represents a RGB color.

**Analysis**  The OsiriX user interface leaves room for improvement. Since the exact same mouse gesture is used for every operation, there is a higher risk of performing an unwanted operation when the wrong state is set erroneously. Considering that navigational and manipulative operations are carried out alike, this can even lead to changes that are difficult to undo. For example, the *window level* mouse function adjusts the width (left/right movement) and the position (top/bottom) of the TF. This is a useful feature to quickly sketch a basic TF, followed by more exact adjustments of the individual points. However, exploration of the new visualization result is likely desired after the TF has been adjusted and care must be taken to not forget changing the mouse button function. Additionally, it might be confusing that the mouse gesture has to be performed on the rendering widget instead of the widget displaying the TF, which it actually modifies.

**Ideas for Improvement**  Based on the analysis given above, several user interface principles and specific implementation ideas are specified hereinafter. OsiriX’ user interface has to overload the same gesture for several different operations due to the limited number of the input device’s DOF. By reducing the amount of features to a reasonable minimum and assigning each a dedicated gesture, the operations can be defined more distinctively. Put differently, the meaning of a gesture should not depend on a state that has to be set prior. Furthermore, gestures should be limited to the widget they thematically belong to (i.e., it should not be possible to adjust the window level with a gesture on the rendering widget). These ideas will be directly reflected by the concept of the widgets presented in the upcoming section [OsiriX].

**3.3 Widget Overview**

The centerpiece of the application implemented for this thesis is a volume rendering screen. Three widgets are essential for it to operate as intended. Several other widgets make interaction and usage of the application easier. Every widget has been implemented in its own module and organized in a logical hierarchy. Most widgets are independent of each other so as to maximize modularity.
Figure 21: The rendering widget after a rotation operation. At the top of the widget, three buttons can be pressed to set the volume’s orientation to axial, sagittal or coronal. Additionally, the current rendering settings can be saved as a preset.

Figure 22: The different states of the rendering widget. When there is one moving touch, it is interpreted as rotation operation. Two touches (of which at least one has to move) scale the volume. More than two touches ignored.
### 3.3.1 Rendering Widget

First and foremost, a rendering widget is required in which the resulting image from the VTK visualization pipeline is drawn (figure 21). Since the visualization of the volume is what the user is most interested in, this widget spans most of the available screen space. The operations that directly manipulate the volume representation (i.e., scale and rotation) are implemented as gestures for this widget (figure 22).

**Rotation** Rotation is probably the most frequently used operation and should therefore also be the easiest. A single moving touch is used to rotate the volume. The local direction vector of touch movement is determined and the volume rotated around a vector perpendicular to it. The rotation is carried out in the direction of the movement. This results in an experience resembling the gyration of a globe. The user can reset the rotation of the volume by pressing one of three buttons that change the orientation to axial, sagittal or coronal. These medical terms refer to three orthogonal imaginary planes where each divides the patient in half. The sagittal plane separates left from right, so it passes through the nose and the navel. Accordingly, sagittal orientation refers to looking at the patient from the side. The coronal plane separates back and front (i.e., the plane intersects the middle of both ears) and coronal orientation makes the physician look at the patient’s face from the front. Lastly, the axial plane separates upper and lower parts of the body (i.e., it is parallel to the ground if the patient stands upright and passes through the entire stomach). Thus, axial orientation allows the physician to *look down the spine* [Preim07].

**Scaling** To view some part of the volume in more detail, the user may want to zoom in on it. Such scaling operations are executed with two diverging touches, referred to as primary and secondary touch hereinafter. For both touches, the initial position vector is kept in memory (old positions). As soon as one or two fingers move, vectors to their new positions are created. The distance between the two old points is computed and the same is done for the two new points. The scale factor is then given as the ratio of the old and new distance and can be multiplied by the volume’s current size to enlarge or shrink it.
Figure 23: The TF widget. In the background, the dataset’s histogram is drawn. In the foreground, the actual TF is displayed. The dashed circle represents the pincher widget that can be used to drag and scale the TF.

Figure 24: The TF widget is more complicated than the rendering widget. This flowchart expresses its functionality. For the sake of simplicity, touches are considered independently.
3.3.2 Transfer Function

The rendering widget allows for interaction with the geometrical representation of the volume. It does not, however, support altering appearance properties such as color and opacity. A transfer function widget providing this functionality is therefore required (figure 23). The widget spans all of the space available horizontally. Considering that adjusting a TF is a more advanced operation and the fact that the TF is only used as a means to manipulate the volume’s appearance, it seems appropriate to allow for hiding the widget. Accordingly, it can be slid in and out on demand. In this section it is assumed that the TF widget is visible. Several gestures were implemented to allow interaction with the TF (figure 24).

Adding & Moving Points  The most basic operation supported by the widget is the addition of points to the TF. A point is added by simply touching a free area of the widget and can be repositioned by moving the finger. The point is automatically inserted at the appropriate position in the TF and its color either interpolated (if the point is added between two existing points) or replicated from the closest point (if the new point only has one neighbor, i.e., it’s to the left or right of the function). If the user touched an already existing point (or somewhere really close to one) that point can be moved freely. When a point is touched, another widget (see section 3.3.3) that offers color selection is shown next to it.

Removing Points  In contrast to OsiriX, no limitations are imposed on the user as to where a point can be moved. Instead, the function’s mathematical validity\textsuperscript{31} is verified after a point has been moved. Invalid points are removed from the function should the need arise, making sure that the function is mathematically valid at all times. This can also be used to quickly remove an entire section of points. To remove an individual point from the function, it can just be dragged out of the widget. The widget then automatically refits the function.

Global Adjustments  It is often desirable to move or scale the entire function, but moving all points individually is cumbersome for all nontrivial TFS. For every TF consisting of at least two points, an additional widget, the pincher, is shown in the middle of the TF. This widget is drawn as a

\textsuperscript{28}Accordingly, the widget actually represents a five-dimensional function, but since a five-dimensional representation is difficult to visualize and control, this complexity is hidden in a two-dimensional representation.

\textsuperscript{29}As we are only dealing with heads here, translation is not really a requirement but could be added easily.

\textsuperscript{30}Alternatively, the position of the camera along the look-at vector could be shifted.

\textsuperscript{31}I.e., any x value must be assigned one y value at most.
Figure 25: A user interacts with the color chooser. The widget is shown as long as the primary touch (marked with a circle in the selected color) is present. Another finger can be used to select a color. The operation can be carried out with one or two hands. The visualization is updated immediately.

stippled circle. When the user touches inside this circle, the entire TF can be dragged by simply moving the finger. If a second finger touches the circle, this new touch can be used to scale the TF horizontally with a pinching (or stretching) gesture. Moving the finger to the right shrinks the TF, while a movement to the left enlarges it. This simplifies adjustments that would otherwise require repositioning each individual point.

Visual Appearance The widget draws the points as circles with their respective color and draws all points as a connected entity. Inspired by OsiriX’ representation of the TF, the area below the function is colored dyed so as to reflect the color at the respective position. Since all points are to be sent to VTK as soon as the TF changes (and, potentially, the function is refit), the widget has to mimic the way VTK interprets the points and their colors. In particular, this means that no function is drawn left of the leftmost and right of the rightmost point (i.e., clamping is disabled). Furthermore, between points the color is interpolated linearly. Next to the actual TF, the widget also draws a histogram of the current volume dataset’s tissue densities. This provides the physician with further information about the dataset being examined.
3.3.3 Color Picker

As was already mentioned, a color picker is shown whenever a point of the TF is touched (figure 25). The idea of the picker is to offer quick color selection while not disturbing the user’s workflow. It is therefore only shown while a point is being touched (i.e., when a new point is added or an existing point is moved). The widget is represented by a semicircular color gradient close to one side of the touch.\footnote{By default it looks like the inverse character “C”.} Assuming the user touches the point with the index finger, the middle finger can then be used to choose a color. As soon as the primary finger (in this case, the index finger) does not touch the point anymore, the color picker also vanishes. To make it more obvious what color was chosen, an additional ring is drawn around the primary finger in the selected color. Color adjustments are immediately sent to VTK. This is done so as to allow for easy and efficient selection of colors.

Realization  The colors that one might want to assign to tissue densities heavily depend on the context in which the application is used. For example, dentists might require colors for skin, bones and teeth while a pathologist may want to tint the heart red. Due to the fact that all colors should be reachable with the secondary finger, there is only limited space available for showing the color palette. Furthermore, the resolution of the palette has to be relatively low bearing in mind the lack of precision due to the finger’s size.

In order to make the selectable colors as well as their order and spacing adjustable easily (esp. without requiring a change of code), a simple trick is employed. The widget loads the color palette as an image from disk. This file can easily be adjusted in an image editing program. To make the shape of the palette adjustable as well, the alpha channel of the image file can be used. When a touch event is propagated to the widget that is inside the widget’s bounding box, one of two things happens. If the user did not touch any color (i.e., the finger points at a pixel with an alpha value of zero), the event is just discarded \footnote{No functionality was available in PyMT to get the color being touched. A patch was hence contributed that adds support for reading a color from a specified coordinate using glReadPixels().}. If the alpha value is not equal to zero, a real color was touched. This color is then sent to the VTK pipeline immediately to instantly reflect the new color’s influence on the volume’s appearance. This is a very efficient way of avoiding complex collision tests while allowing arbitrary shapes for the palette.
3.3.4 Presets

The tasks a physician faces are often recurring. To only show the patient’s teeth, for example, a TF with just a few white points at the appropriate positions suffices in most cases. To make switching between several TFs for different purposes easy, the current TF can be saved as a preset. To do this, the user simply touches a button and enters a descriptive name in the virtual on-screen keyboard that is then shown. The TF is then stored as a preset that can be used across datasets. When a dataset is opened, it is first rendered into a framebuffer object in a small size with all available presets. These renderings are then used as icons next to each presets’ description in a scrollable list of all presets. When a preset from the list is tapped twice, the current TF is replaced with the one stored in the preset. All presets are stored on disk and are available between sessions.

3.3.5 Dataset Gallery

In order to visualize a patient’s dataset, it is first necessary to load it from disk. A file browser to locate and select datasets is therefore required. When a dataset is selected, a profile is created for it. The profile stores all information required by the application, such as the path to the dataset, the date it was added and cache data. All profiles are stored in a database on the hard disk so that the user does not have to browse the file system every time a specific dataset has to be visualized. Several datasets can be added at once and no dataset is loaded automatically because this is a costly operation. Data such as the patient’s name and gender are extracted from the DICOM files and added to the profile. Additionally, after a dataset has been opened, a snapshot of the rendering is taken to simplify recognition of datasets that were already examined. The last TF that was specified for the dataset is also stored in a normalized form and reloaded the next time the dataset is opened. All profiles are persistently stored in a folder that can be specified by the user. A profile is also just an ordinary widget that can be rendered on the screen. A profile’s representation includes the snapshot of the volume (if available, otherwise a dummy image is used) and some of the information it stores, e.g. the patient’s name. The benefit of this is that all profiles can be added to a scrollable list, the dataset gallery, which is presented to the user. From this gallery, the user can choose the profile to be opened.

While PyMT already incorporates a file browser, a series of modifications was required for it to work properly.
Figure 26: Rendering mode: The rendering widget spans most of the available screen space from the left. The TF widget can be slid in and out on demand and covers all the horizontal screen space. In the space that remains to the right, the presets are shown.

3.3.6 Widget Integration

All of the aforementioned widgets must be combined and laid out across the screen (figure 26 shows the result of this layout process). A stage widget that is not visible by itself is used to govern the addition and removal of the basic widgets when changing from the dataset gallery to rendering mode and vice versa. It is the only object that is added to the PyMT window directly and forms the topmost node (from the client application) of the scene graph\textsuperscript{35}. The stage widget also performs all the computations that are needed in order to adjust the other widgets to the size of the screen. Following the singleton design pattern, there is only one stage object at any given time.

Profile objects are passed around between widgets instead of the raw information. For example, when a profile from the dataset gallery is double-tapped, it calls the stage’s \texttt{open\_profile()} method and passes itself as

\footnote{It is not really the root of the scene graph, since that is by design the PyMT window object itself.}
argument. The stage then shows a progress bar and passes the profile object to a loader, which extracts the path to the dataset from it. The loader triggers a callback of the stage when loading is complete. The stage then creates all widgets for the rendering mode, removes all other widgets and adds the newly created widgets to itself as children at the appropriate positions.

Thus, all information and functionality is nicely encapsulated in modular objects. No tangling relations exist between thematically independent widgets.
4 Evaluation

The success of the two main objectives of this thesis (hardware and software) will be discussed in the following. Decisions that were made will be summarized and explained. Problems and their solutions will be discussed and some advanced ideas for improvement are provided. Furthermore, an expert evaluated the software component and provided valuable feedback which will be analyzed as well.

4.1 Hardware Evaluation

Many approaches to building a multi-touch device are feasible. Before the hardware could be built, all possibilities had to be considered.

The most fundamental question is that of the sensing technique. Among the different options (such as optical, capacitive, resistive, etc.), the utilization of optics and computer vision were the most appealing because they’re by far the easiest to work with in hobbyist scenarios.

The question that arises next is how large one wants the touch surface to be. To allow for interaction with two or more users simultaneously while still remaining affordable, a surface diagonal of about 40" seemed reasonable. This size decision mandates the use of a projector due to its scalability\textsuperscript{36}. When using a projector, it is desirable to project the image from behind the projection screen to avoid shadows cast by the user’s limbs. Hence, the construction of a vertical setup (i.e., the device is mounted on the wall) is impractical due to space limitations. Accordingly, the decision was made to build a table box with the projector inside (a horizontal tabletop setup).

The image was initially projected onto an architect’s drawing paper which led to a rather poor image quality. It was later replaced by a proper rear-projection material. This greatly improved the visual quality of the image. Among the different means of illumination (DI, FTIR, DSI), the DSI approach promised the most versatile set of capabilities. It combines the advantages of both DI (fiducial marker support) and FTIR (high contrast) at the expense of higher cost for the required acrylic. The additional cost can be neglected considering the benefits, which led to the decision of implementing the DSI approach. Although ready-made IR LED strips are available, they are very expensive. In order to save money, an aluminium frame that holds 90 IR LEDs was built and the LEDs soldered together in order to illuminate the acrylic placed inside the frame.

A camera was required to record the user’s interaction with the touch surface. A camera that operates at a resolution of $640 \times 480$ pixels and 60 FPS was purchased. After replacing its IR blocking filter by exposed negative

\textsuperscript{36}The disassembly of a 40" LCD or Plasma panel would have been too risky by far.
film\textsuperscript{37}, BLOBs could be detected. However, the quality of the image segmentation algorithm’s results suffered from the high amount of ambient light that still reached the camera sensor. After replacing the negative film by a proper IR bandpass filter, only light at the wavelength of the IR LEDs was allowed to pass through to the sensor, which produced considerably better results. Additionally, the camera’s original lens was replaced by another lens whose focal length allowed focusing the touch surface more precisely. Since it was very difficult to orient the camera in such a way that it faced the touch surface above it, a custom-made mount for the camera was employed. This stand significantly simplified positioning the camera. Lastly, the actual table itself was built. As main material, wood was chosen over metal since it is less expensive and much easier to work with. Three of the four side panels can be easily removed. Besides improving robustness, this also has the benefit of protecting the components. Furthermore, the internal complexity remains hidden.

4.1.1 Solved Problems

Several problems worsened the quality of the interaction and the touch input sent by the tracking application. In this section, their cause will be analyzed and the solutions explained.

\textbf{Weak Contrast} The quality of the BLOBs that were produced by the DSI approach were not satisfying initially. Bad coupling of the fingers with the acrylic triggered the FTIR effect only under heavy pressure. This presented a problem especially for dragging operations, where tracking of a moving finger was interrupted rather quickly. To improve coupling, a thin film of silicone was sprayed on top of the surface. The film is hardly seen and only slightly noticed when touching the surface. The application of silicone greatly improves touch sensing and even produces BLOBs under very weak pressure. Dragging operations also benefit from the improved coupling and can be carried out much more easily. After heavy usage, the surface has to be cleaned and the silicone reapplied. Additionally, the image processing filters supported by the tracking application were fine-tuned. The importance of a properly adjusted image processing pipeline cannot be overstated.

\textsuperscript{37}This is a simple but effective way to weaken the incoming light and hence darken the camera image.
Delayed Response  On the development system (OSX) the interface suffered from notable delay. The delay was caused by the camera driver’s low frame rate on OSX. By installing Windows 7 on another dedicated system and using an unofficial driver only available for that platform, the frame rate could be increased from 30 to 60 FPS at the same resolution. This greatly improved user interaction and reduced the delay of the response considerably.

Image Projection  To show the computer image directly below the touch surface, one has to choose between using a projector or disassembling a monitor of the appropriate size. While 40” monitors are available, one cannot be sure beforehand as to whether the particular device at hand can be disassembled properly. Due to the fact that said disassembly certainly voids warranty and the potential risks involved with taking the device apart, it was decided to use a projector instead.

When using a projector, the challenge is to adjust it in such a way that the projected image covers the entire projection surface with minimal distortion. This is made much more difficult as a result of the limited space available inside the table. The shorter the distance that the projector’s light has to travel, the smaller the resulting image is. Therefore, in order to maintain reasonable measures (esp. height) for the table, one could use an ordinary projector and one or more mirrors to lengthen the distance the light has to travel. Tests with an early prototype revealed that the utilization of an ordinary projector in combination with mirrors is very cumbersome to set up and easily affected by vibration (e.g. caused by users leaning against the table). As a consequence, a special projector with a so-called short-throw lens was purchased. Due to the short distance this projector needs to produce an image with 40” diagonal, it was even possible to use it without any mirrors. By positioning the projector vertically and projecting directly at the surface, the desired result could be achieved much more easily. In order to orient the projector vertically, a custom mount that allows to rotate the projector had to be built. This mount has proven to be invaluable for properly positioning and adjusting the projector. Using it, the image can be projected onto the projection foil with very little distortion.

Jitter Inaccuracy  Due to the quality of the camera and the amplification filter necessary to find BLOBs, the touches sent to PyMT suffer from jitter artifacts. This means that the tracker sends touches that perform slight jitter movements although the fingers producing the BLOBs do not move at all. This inaccuracy causes confusing rotations or movements of the objects being touched, such as the volume being rendered.

\footnote{Care must be taken in this case, because not all projectors allow being oriented like this.}
The problem was solved by implementing a postprocessing module for PyMT. This module simply discards all touch movement events that move the touch by less than a given threshold. The module can be activated for only a subset of the available input devices so as not to discard any events caused by devices that do not suffer from jitter (e.g., a mouse). A simple calibration tool that measures the amount of jitter and updates the configuration for the respective hardware has also been implemented. This module solves the problem of BLOB jitter in a satisfactory manner. Unwanted touch movements are suppressed very reliably, resulting in an improved user experience. The module can be used for all PyMT applications and is independent of the tracking application.

**Lamp Reflection** One major problem was the occasional occurrence of a false BLOB at always the same position. This led to severe confusion as some applications started behaving oddly. At first, the BLOB seemed to appear under random circumstances, which made analyzing it particularly difficult. After extensively searching for the cause of the problem, it became obvious that a reflection of the projector’s lamp is captured by the camera. However, this reflection is only picked up when a certain region of the screen becomes sufficiently bright, which led to the seemingly random appearance of the BLOB. Additionally, the touch surface bends very slightly under pressure, but still enough to shift the false BLOB by a tiny distance. Therefore, basic background subtraction in the tracking application does not yield satisfactory results.

Since the region where the false BLOB occurs is quite small, a reasonable solution to the problem is to just ignore `on_touch_down` events in that area. PyMT already comes with a module to ignore arbitrary regions, but defining these regions optimally is a very cumbersome task. A calibration tool to automatically find and ignore such areas was hence implemented. The calibration program simply divides the screen into small bins and colors them white individually, while the background remains black. The user is instructed not to touch the screen while calibration is in progress. This way, every touch event generated during the calibration phase is interpreted as being caused by a false BLOB and the region where it appeared is automatically added to the ignore list. By making use of this module, the erroneously generated touch events can effectively be discarded.

**Double-Tapping Thresholds** A famous operation that one can perform with a mouse is the double-click, which is often associated with opening the entity that was clicked. Due to the fact that almost every user is familiar with this operation, its adaption to multi-touch scenarios is desirable and

---

39 Due to the small size of the region, it is very unlikely that a user wants to touch exactly there. It is still possible to move a touch through the ignored region.
has led to the double-tap. In BoneTouch, a double-tap is used to open a dataset for example. In PyMT, a double-tap is defined as two subsequent touches with different IDs at almost exactly the same position in a short time. Thresholds for the maximum amount of time that may pass between the two touches as well as for their distance to each other can be defined by the user.

However, finding good values for these two variables is difficult and they very much depend on the multi-touch system being used (e.g., frame rate of the camera). Another calibration tool was added that measures the distance and time between a series of double-taps performed by the user. These values can then be set in the user’s configuration file. This greatly simplifies configuration of double-tap parameters [Hansen09].

4.1.2 Results

The results achieved with the hardware installation (cf. section 2.2) are quite satisfactory. All that one has to do is connect a computer (with the required software installed) via the cables that come out of the side of the table. The entire device (including the projector, IR illumination and fans) can then be turned on by plugging a cable into a power outlet. After calibration 2.2.1, several example applications that come with PyMT can be run and used nicely. The camera operates at a resolution of $640 \times 480$ pixels with 60 FPS and allows for smooth interaction with the surface.

While the installation does function in situations where ambient IR light is present, it obviously works best in dark environments or such with only low ambient IR radiation.

4.1.3 Possible Improvements

Although the installation does work as intended, there is always room for improvement. Some ideas that might positively influence the performance of the device are listed below.

**Better Camera**  The main problem of the construction is the camera. A resolution of $640 \times 480$ pixels is used to capture the entire touch surface\footnote{Not even taking into account borders that need to be cropped.}. Due to the large size of the surface and the projector’s higher resolution of $1024 \times 768$ pixels, this is not always sufficient to discriminate different touches. Therefore, care has been taken to make user interface elements sufficiently big. Ideally, a camera with higher resolution should be used or images from several cameras stitched together.
**Higher Luminous Intensity** Sophisticated illumination is key to success in the aforementioned optical multi-touch setups. The LEDs used here are operated at a conservative intensity so as to minimize the risk of burning them through. Their intensity could be maximized by utilization of different resistors. Furthermore, the borders of the acrylic have been sawed by the manufacturer, resulting in a rough interface. They could be grinded and polished in order minimize scattering of light at the entry point. According to the manufacturer, this might improve light entrance by up to 6%.

**Improved Tracking Algorithm** At the time of writing, CCV is still used as tracking software. The tracking algorithm employed by the software leaves room for improvement. In particular, when two or more fingers are quickly moved parallel to each other in the same direction, the tracked fingers are sometimes confused which leads to confusing jumps of the detected touch events. The tracking algorithm should therefore be improved\(^4\). 

\(^4\)E.g. by also taking into account the local movement vector of each touch.
4.2 Software Evaluation

At the beginning of the design phase, several possible software combinations were evaluated. Only reactTIVision and CCV were available at the time. Due to reactTIVision’s focus on fiducial marker tracking and CCV’s comparatively broad support for finger tracking, CCV clearly became the tool of choice. In the future it will likely be replaced by Movid, which is more stable and allows for better configuration of the image processing pipeline. It then became obvious that a toolkit to handle the TUIO stream sent by CCV would be required. PyMT was chosen among other options such as MT4j [MT4j] because it seemed the most advanced, was actively developed and, most importantly, provides a mouse simulator. OpenGL, which is supported by PyMT, seemed the most appropriate for drawing the user interface since it offers great freedom with respect to user interface design\footnote{I.e., one is not bound to the set of widgets of a particular user interface toolkit.}.

For volume rendering VTK and Voreen [Voreen] were possible candidates. VTK was the most attractive due to its Python support, extensive documentation and good reputation.

4.2.1 Solved Problems

Three main problems were faced when it came to integrating PyMT and VTK. In the following two paragraphs, different approaches will be described and the final solutions explained.

Interacting with the VTK Scene  Among the image processing and visualization modules, VTK also supports several input devices such as mouse and keyboard. Using these, VTK allows interaction with the scene that is drawn. While specific input devices (e.g. mouse, keyboard, joystick) are encapsulated by an interaction abstraction layer, it soon became obvious that transforming the multi-touch input events to VTK on this level would be cumbersome. This is due to the fact that the abstract base class for input devices was designed with traditional input devices in mind. For example, in order to add support for a new multi-touch input device, one would have to implement methods such as `onLeftButtonDown()`, which is clearly not desirable. As a consequence, another approach was taken. The interaction facilities VTK offers are bypassed and not used at all. Instead, multi-touch input is interpreted by BoneTouch and used to directly manipulate the VTK objects (e.g., volume, camera). This allows for easier implementation and more complex manipulations of the VTK objects.
Integrating VTK and PyMT  The rendering VTK produces and the rest of the BoneTouch interface should be combined (i.e., it is not desirable to accept multi-touch input in one window and show the rendering in another window). Three alternative approaches to bring rendering and user interface together were evaluated.

VTK could have been used to draw the user interface directly. That, however, meant that the user interface (in VTK) would have to be decoupled from the actual input (from PyMT). Hence, this option was quickly discarded. Instead, the visualization from VTK should be drawn in the PyMT application.

The first approach to achieve this is based on exporting the rendering from VTK as a texture. The exported texture can then be drawn on a rectangle in BoneTouch. While this works well in theory, practical limitations rendered this approach impracticable and, on some systems, even useless. Firstly, this approach requires an additional window containing only the VTK visualization to be opened\(^{43}\). Secondly, platform specific code is required with this approach\(^{44}\). While proof of concept implementations were functional for Windows and Ubuntu Linux, a modification of the VTK source code was necessary to extract the ID of the background rendering window. With this modification, the VTK rendering was finally drawn in the PyMT application on OSX as well. Unfortunately, it was still impossible to use this rendering in an advanced manner since it was not extracted as a texture, but only drawn as an overlay. Due to this and the significant amount of platform-specific code required, this approach could only be a temporary solution. A reasonable solution to the actual problem, namely VTK not being able to render into already existing OpenGl contexts, was therefore strongly desired. This would also eliminate the need for any platform-specific code.

After extensive research, the VTK source code was changed so as to allow exactly this. With this modification, VTK is now able to render into another OpenGl context by skipping all initialization steps\(^{45}\). In order to make more advanced transformations of the visualization (such as scaling or rotating) possible without altering the actual VTK scene, the rendering from VTK can be drawn into a framebuffer object (FBO), which in turn is projected onto a textured rectangle. This approach has the advantage of decoupling the rendering’s resolution (which directly affects rendering speed) from the size at which the rendering is drawn. BoneTouch makes use of this by always keeping the rendering widget the same size, but adjusting the quality (i.e., the resolution) of the rendering adaptively to allow for interactive frame rates [VTKUG04].

\(^{43}\)While this is not a technical problem, it still is confusing behaviour.

\(^{44}\)The situation is even worse. For example, on OS X different frameworks such as Cocoa and Carbon do again require specific code.

\(^{45}\)Since this might be interesting for other applications as well, the patch will be contributed back to the VTK project. Credit goes to Mathieu Virbel for the implementation.
Parallelizing Tasks  In order to maintain a responsive user interface regardless of the tasks that might be performed in the background, it is often mandatory to use threads or subprocesses. Certain architectural limitations of the Python interpreter made this very difficult, though. The matter affects things like parallelizing user interface and volume rendering code and other things that require concurrency. To outline the problem, the issues that were faced when implementing something as simple as a progress bar will be explained.

While loading a dataset is simply a matter of passing the correct path to the VTK reader object, it is much more difficult to maintain a responsive user interface while loading is in progress. Ideally, the part of the program that draws the user interface would run in its own thread and the loading operation would run in another thread. Due to a well-known problem\(^46\) of the CPython interpreter, it is not possible to use threads with this interpreter to improve efficiency, unfortunately. A global interpreter lock (GIL)\(^47\) prevents the CPython interpreter from executing bytecode from more than one thread at a time.

Due to the fact that VTK (and some of PyMT’s dependencies) is only available for CPython, other interpreters that do not have a GIL are not an option. Using processes instead of threads is not an alternative either, since the VTK reader object cannot be serialized which would be necessary to pass it from one process to the other. It would still be possible to use another process to separate the interface from loading the dataset, but that requires performing all of the visualization in that process as well. This, however, does not only require serialization of the rendering result, but also keeping another window with its own OpenGL context open and rendering in that window. Although reading a dataset might take up to half a minute or more, an animated progress bar is not worth that much effort.

Instead, a workaround is employed to overcome the problem. The actual problem is that control is handed to VTK as soon as it starts loading the dataset and there is no chance for the user interface to be redrawn. Control of the process is handed back to the PyMT application only after the dataset has been loaded completely. In order to overcome this problem, a facility of VTK is used. The base class of all VTK objects, \texttt{vtkObject}, implements the observer design pattern. This means that callback functions can be registered with any VTK object. These functions are then called when a specific event occurs. By loading the dataset in a different thread and registering a callback with the reader’s \texttt{ProgressEvent}, VTK hands control over to the main thread for a very short time. To make sure that the PyMT main loop has enough time to redraw the user interface, the callback just makes the

\(^{46}\)Actually, this is one of the biggest problems of the CPython interpreter.

\(^{47}\)The GIL is a very controversial topic in the Python community and a discussion of the advantages and disadvantages is far beyond the scope of this thesis.
loader thread sleep for a millisecond. This makes it possible to inform the user about the progress of the loading operation, which is important given that the application usually runs in full screen mode and would otherwise seem to hang for half a minute or more. A clean solution to this problem either requires future versions of Python to drop the GIL or a complete rewrite of the entire application in MT4j [Python; Schroeder05; MT4j]. It is doubtful whether a progress bar is worth the effort of a complete reimplementation with a different framework.

**Performance Improvements** The time it takes to render the volume directly affects the entire program’s responsiveness. Since it is desirable to keep the user interface responsive at all times, several performance optimizations have been implemented. The resolution of the image directly dictates the time it takes VTK to render the volume and at the same time indicates the quality of the visualization. A slider widget has been implemented that can be used to change the resolution and hence trade quality for speed and vice versa. However, it is cumbersome to adjust the resolution manually. Therefore, the resolution is automatically adjusted adaptively to stay above a certain frame rate when interacting with the user interface. As soon as the interaction ends, the volume is rendered at the usual resolution. Additionally, the rendering is cached and the visualization pipeline only executed when the volumes appearance is changed (i.e., when adjusting the TF, rotating or scaling it). These adjustments have led to a boost in performance and overall responsiveness.

### 4.2.2 Results

After the initial problems concerning the integration of VTK and PyMT were resolved, all planned features and some additional ones were implemented successfully. The application is robust and operates as planned on the hardware device (figure 27). DICOM datasets can be loaded and visualized without problems. The transfer function widget can be used to adjust the appearance of the volume, while the rendering widget allows for gestural transformations of the volume. Presets can be used to quickly switch between different visualization options. In the following, the efficiency and simplicity of the implemented gestures after intensive usage are discussed. Furthermore, an expert was sought to share his opinion about the gestures and the application in general. This expert evaluation will be discussed in section 4.2.4.
Figure 27: A user interacts with the volume rendering widget. A scaling operation is about to be performed with two fingers. The application is used on the self-made multi-touch table.
Volume Manipulation Gestures  Rotation and scale gestures were implemented for the volume rendering widget. No gesture was added to translate the volume. This was done on purpose, because only heads were visualized for the sake of simplicity and no translation operation is necessary for this relatively spherical kind of volume. An additional gesture for translation should be implemented as soon as entire bodies have to be visualized. Since rotations are probably carried out the most often, the corresponding gesture was kept as simple as possible and can be carried out with only one finger. To improve usability for users already familiar with OsiriX, the rotation algorithm behaves the same. This gesture has proven to be an efficient way to rotate the volume.

Scaling operations are often required to zoom in on structures of interest. A scale gesture was therefore implemented. Taking into account what users may already know, the implementation is akin to the scaling gesture found in other contexts. The changing distance between two fingers is measured and used as a scale factor. In order not to lose track of the volume, the center of the scaling operation is always the center of the volume. After extensive usage, it can be stated that the operation is easy and efficient to perform.

Transfer Function Manipulation Gestures  The most important widget in terms of modification of visualization parameters is certainly the TF widget. Adding new points should be as easy as possible. To add a new point, the user simply touches a free area of the TF widget. A new point is then added to the TF and the volume rendering updated. As expected, this is an extremely efficient way of quickly sketching a TF. To lower the rate at which new points are added erroneously, one might consider only adding new points after a double-tap, though.

Selecting and moving a point is also a common operation. It is therefore kept simple and can be performed by touching and dragging a point. Since moving a point left of its predecessor or right of its successor, this operation has a high potential of leading to an invalid function. Two approaches to prevent this were evaluated. The first possibility is to limit the area in which a point can move (i.e., it must not move left of a preceding or right of a succeeding point) and is akin to OsiriX’ behaviour. This turned out to be rather confusing since a point being moved suddenly stops following the user’s finger. Consequently, the implementation was changed so as to allow movement of a point inside the entire widget. This requires automatic refitting of the entire TF (by removing invalidated points) after a point has been moved. Fortunately, this change has led to a much more pleasant and efficient user experience. Additionally, this implementation also introduces a method to delete an entire segment of points.

Moving the entire TF can easily be accomplished by dragging the TF’s
pincher with one finger. This operation is both easy to perform and easy to understand.

The two finger scaling operation requires the user to first touch the TF’s pincher with both fingers and then moving one finger to the left or right to scale the TF, while keeping the other in place. The initial rationale for this was the idea of supporting the addition of points (potentially by other users) even while the TF is being scaled. With hindsight this was a bad idea since the TF is almost always changed by only one person at a time. Furthermore, the initiation of the gesture is unnecessarily complicated. The implementation should therefore be adjusted.

Removing a point from the function is particularly easy. The point just has to be moved out of the TF widget. Although the gesture cannot be understood by just looking at the application without introduction, it is very easy and efficient to carry out once one is aware of how it can be done.

**Color Picker Gestures** Changing a point’s color is an important operation as it heavily affects the appearance of the visualization. The necessary should therefore be easy and efficient as well. However, the widget required to choose a color should not disturb the user’s workflow. It was therefore decided to show a palette whenever the user touches a point. To make one-handed color picking possible, this widget is shown in close proximity of the point being touched. As soon as the point is no longer manipulated by the user (i.e., the finger does not touch the point anymore) the widget should automatically be removed.

With these requirements in mind, the widget was implemented. Since changes to the points’ colors are immediately reflected by the visualization, the user can instantly see the effect of a change. Accordingly, changing a point’s color is a matter of two touches and can be achieved very easily and efficiently.
4.2.3 Possible Improvements

The result of the software development is quite satisfactory already. Until the software product can really be used to showcase the potential of volume rendering and multi-touch, some improvements should be realized, however.

**Volume Rotation** At the time of writing, the volume rendering widget allows rotations to take place around all axes of the object’s coordinate system. Sometimes, this leaves the volume in a confusing orientation (e.g. upside down). Attempts to limit the possible rotations to only two axes were successful, but led to even more confusing behaviour and were hence discarded. Therefore, finding the most intuitive rotation operation remains open for research.

However, it is questionable whether it should be changed at all, considering that the operation is equal to the one found in OsiriX and changing it might be confusing for users of both systems.

**Clean Parallelization** As was already explained, CPython’s GIL presents a problem when it comes to parallelizing tasks. While the issue was solved for the animated progress bar when loading a dataset, it is much more difficult to separate the entire volume rendering from the user interface logic. This, however, would be necessary to completely eliminate the slight stuttering effects that occur every time the volume has to be rendered. At the moment this issue is worked against by minimizing the time it takes VTK to complete a render pass. This is done by making sure that no redundant computations are performed (i.e., the rendering is cached and only rendered again when something changed). Additionally, the resolution of the volume is changed adaptively, since lower resolution leads to lower rendering times.
4.2.4 Expert Evaluation

An orthodontist with extensive experience in medical volume rendering (with OsiriX) was asked to perform a series of tasks with the application and answer several related questions. The survey consisted of four parts that had to be answered in order. Firstly, some general questions about the potential of multi-touch were listed. Secondly, a series of consecutive usage tasks with BoneTouch had to be carried out\footnote{The user’s interaction with the system was recorded by a camera to ease analysis.}. Thirdly, a question had to be answered for each task that was performed. Lastly, the subjective impression of the test person was obtained.

It is important to note that the test person used his own multi-touch hardware to test the application\footnote{This is simply due to the fact that the test person lives on the other side of the globe.}. The main difference to the system presented before is that it is wall-mounted, does not use a projector and has the camera mounted on the ceiling.

**Potential of Multi-Touch** In the first section, questions about the disadvantages of using OsiriX with a mouse were asked as well as the user’s opinion about the advantages and drawbacks of multi-touch in general.

The hypothesis that switching back and forth between different mouse function states can be cumbersome and confusing was verified. Furthermore, the test person expressed the opinion that different multi-touch gestures can be used to offer a more seamless interaction with the volume rendering application. This confirms the analysis of some of the shortcomings from section 3.2. When asked about potential disadvantages of multi-touch, the test person answered that the lower accuracy of a finger touching a sensor (as opposed to the pixel accuracy of the mouse) might become a problem for tasks that require precise input.

**BoneTouch-Specific Tasks & Questions** In order to gain an insight into the way a user interacts with the system and learn about potential problems and difficulties, a series of tasks had to be carried out and corresponding questions had to be answered for each task.

The following tasks had to be performed in order: Rotation and scaling of the volume, creation of a new TF, basic and advanced adjustments of the TF, basic and advanced color and opacity adjustments and finally, removal of individual points and entire segments from the TF.

The rotation and scaling operations for the volume were easily understood and performed by the test person. The same is true for creating a new TF and moving it.
Scaling the function, however, presented a problem. The expert criticized that the scaling gestures are different for the volume and the TF widget. The TF is scaled by placing two fingers inside the TF’s pincher and dragging one of them to the left or right. Since the ring that represents the pincher is rather small, placing two fingers inside it is problematic if resolution is too low. For the volume on the other hand, the relative distance of two freely movable fingers to each other is used as scale parameter, independent of the direction of their movement. This inconsistency, as well as a problem with the resolution of the camera image in the area of the TF widget made it difficult for the test person to perform the scaling operation. This has to be fixed by adapting the volume rendering widget’s scale gesture to the TF widget. To resolve the resolution issue, a secondary touch anywhere inside the TF widget should be used for the scale gesture as long as a single primary touch is present inside the circle.

Another problem caused by the low resolution of the expert’s hardware was exactly positioning a point at zero or full opacity at the very bottom or top of the TF widget. Snapping areas should therefore be added that allow for more tolerance in these regions.

Despite these issues, the test person was able to interact with the TF as expected. Addition and removal of points in particular were easily carried out. The idea of the color picker widget was well received, but the test person pointed out that the default colors are not suitable for orthodontics and should be changed.

**Subjective Impression**  Overall the test person’s feedback was positive and the opinion was expressed that multi-touch can certainly ease interaction in medical image exploration. Some interactions, such as the removal of points by simply dragging them out of the TF widget or the selection of a color turned out to be good ideas.

In its current state, however, the application cannot yet be used on a day-to-day basis. Thus, the aforementioned problems have to be resolved before it can be showcased.

---

50 This is due to the fact that the test person’s camera is mounted on the ceiling, facing the touch surface from quite some distance, which causes trapezoidal distortion that is especially severe in the bottom of the screen, which is where the TF widget is by default.
5 Conclusion & Further Work

Conclusion In this thesis, the possibilities of combining a medical volume rendering application with novel multi-touch input devices have been explored. Fundamentals of both medical imaging and multi-touch were explained.

By analyzing an already existing mouse-controlled volume rendering program, several disadvantages caused by the limited nature of the mouse were encountered. A solution to these drawbacks was proposed in the form of a multi-touch volume rendering application. In contrast to the analyzed program, this application does not overload the same input operation with several different meanings depending on the program’s state. Instead, every operation one can perform with the application is accessible via a unique multi-touch gesture. User confusion (which is common for the analyzed program where operations are often performed in a wrong state) is therefore minimized. For the implementation, already available software libraries for multi-touch user interface creation and volume rendering were utilized. Much emphasis was put on writing clean, readable, styleguide-conform, well-documented and well-designed program code. Where functionality wasn’t available in the external libraries, patches have been produced and contributed back.

In order to test the application, multi-touch hardware was required. Since ready-made solutions were scarce and expensive at the beginning of the thesis, a self-made multi-touch input device was built. The construction of the device documented.

An evaluation of the hardware and software, as well as an expert evaluation conclude this thesis.

Further Work The expert evaluation of the application provided valuable feedback. The remaining problems that were discovered should be solved. In particular, the scaling operation of the transfer function has to be redesigned. To completely eliminate stuttering effects of the user interface, a way to properly separate user interface logic from the visualization pipeline has to be found.

For the hardware part, more than one camera should be used. Further improvements regarding the coupling between finger and acrylic should be researched. The tracking application should be replaced by a more configurable program that provides fine-grained control about the image processing pipeline. Furthermore, the tracking algorithm should be changed so as to minimize confusion of tracked fingers.

In general, the usefulness of novel interaction methods such as multi-touch in medical contexts should be researched in more detail.
A Instructions

The following survey is comprised of four sections. When answering questions, please be as precise and elaborate as possible. Please record yourself while using the application (Screenrecording/External Camera or both). The questions and tasks of the survey have to be answered in order. Thank you for participating.

B Survey

B.1 General Questions

1. What are the problems of using an application like OsiriX with just a mouse?

2. How could a multi-touch device be used to improve the interaction?

3. What are the advantages of using multi-touch instead of a mouse?

4. What are the disadvantages of using multi-touch instead of a mouse?

B.2 Tasks

Please perform the following tasks. The questions in the next sections are based on the experience you gained while solving the tasks. Before starting with the tasks, make sure your screen is calibrated properly. Start the application and open a patient dataset. Then solve the following tasks in order. (Please do not use presets.)

A From axial orientation, rotate the volume manually into coronal orientation, then into saggital orientation.

B Scale the volume so that the patient’s nose covers all of the available space and is completely visible.

C Create a new, narrow transfer function (ignore colors) that shows the patient’s skin.

D Adjust the transfer function from C so that you can only see the patient’s teeth.

E Adjust the transfer function from D so that it maps roughly 80% of the available densities to a non-zero alpha value.

F Adjust the transfer function from E so that you can see the patient’s skin, bones and teeth at the same time.
G Adjust the colors of different tissues to get a somewhat realistic image. Add new points if necessary.

H Remove a single point from the transfer function.

I Remove all but the first and the last point from the transfer function.

B.3 BoneTouch-related Questions

1. What factors made the use of the multi-touch user interface easy or difficult for you?

2. Considering the advantages you mentioned in B.1 3, have they been used to their full potential? Explain why you think so.

3. In what way have the disadvantages you mentioned in B.1 4 presented a practical problem?

4. Compare multi-touch and mouse input: What have you been able to do more/less efficiently and why?

5. Which gestures do/don’t you consider appropriate (i.e., easy to use & efficient) and why?

6. Has the lower resolution of the finger pointing mechanism been a problem? In which way?

7. Which gestures do you use the most often? Are there gestures you do not use at all?

8. How many hands did you use respectively for the tasks?

Additionally, please indicate whether the following tasks were a) easy to perform, b) easy to understand, c) both or d) neither easy to perform nor easy to understand.

1. Rotating the volume

2. Scaling the volume

3. Moving the Transfer Function

4. Scaling the Transfer Function

5. Adding new points

6. Setting the color of a new point

7. Moving existing points
8. Adjusting an existing point’s color

9. Removing a single point from the Transfer Function

10. Removing a set of points from the Transfer Function

B.4 Subjective Impression

Please elaborate whether you valued using the system.

1. Can you imagine using a multi-touch interface for your everyday work as a replacement for mouse input?

2. Does the system improve your spatial perception of the patient’s dataset?

3. Which input device do you think is more intuitive?

4. What other features can you envision to enhance the application?

5. Do you think you can fulfill your tasks more efficiently (with respect to quality of the result and time spent)?
References


[Lorensen87] Lorensen, William E. and Cline, Harvey E., Marching cubes: A high resolution 3D surface construction algorithm, SIGGRAPH Comput. Graph., 1987

[Shirley90] Shirley, Peter and Tuchman, Allan, A polygonal approximation to direct scalar volume rendering, SIGGRAPH Comput. Graph., 1990

[Han05] Han, Jefferson Y., Low-cost multi-touch sensing through frustrated total internal reflection, Proceedings of the 18th annual ACM symposium on User interface software and technology, 2005


[Hansen09] Hansen, Thomas and Hourcade, Juan Pablo and Virbel, Mathieu and Patali, Sharath and Serra, Tiago, PyMT: A Post-WIMP Multi-Touch User Interface Toolkit, Conference for Interactive Tabletops and Surfaces, 2009


[WB03] Wu, Mike and Balakrishnan, Ravin, *Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays*, In UIST ’03: Proceedings of the 16th annual ACM symposium on user interface software and technology, 2003


