Reconstructing bones from stereoscopic X-Ray images of fish

TDT4501
Datateknologi Fordypningsoppgave

Autumn 2014

Bart van Blokland

Supervisors:
Theoharis Theoharis
Christian Schellewald

NTNU – Trondheim
Norwegian University of
Science and Technology

Department of
Computer and Information Science

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The project documented in this report aims to filter bones out of X-Ray images of fish. A setup is used where a fish is transported on a conveyor belt through two X-Ray scanners that subsequently produce two stereoscopic X-Ray images. The objective is to first filter the bones out of these images, then use stereoscopic reconstruction to find the positions of the bones in 3D space. These positions can for example be used to deduce optimal cuts that yield a maximum amount of flesh and minimises waste.

A number of programs have been written as part of this project, which are mainly divisible into two parts: an X-Ray simulator and a bone filter. The simulator attempts to generate accurate X-Ray images from 3D models of fish filets, while the bone filter attempts to analyse X-Ray images of fish and extract the bones from them. The 3D reconstruction of the bones is not part of this project.

The simulator was created to allow experimentation with different setups and to analyse the effects of changes to them. A physical X-Ray scanner is difficult to modify, so a tool that approximates images which can be changed on demand thus has significant value in the context of developing the bone extraction tool.

The bone extraction tool aims to locate the bones inside a given X-Ray image of fish. Ideally, it results in a binary image containing only the pixels that were the result of bones inside the fish, and includes all bones which were present in the scanned fish filet.

The Simulator and bone filter will be discussed and documented in detail in separate sections.

Chapter 2

Background

2.1 Physical setup

Both the simulator and the bone filtering tool are based upon the same X-Ray scanning setup.

2.1.1 Description of scanning machine

Figure 2.1 shows this setup. A piece of raw fish (f) is placed upon a conveyor belt that moves at a constant speed in the negative x direction. Above this belt is located an X-Ray emitter (E) that acts as a point source emitting X-Rays. These rays are detected by a linear detector (d) located at the level of the belt starting at the origin and directed along the postive y-axis.

The detector is assumed to start at the origin and is directed in the positive y direction. The rays from the point source that intersect with the detector can be modelled by a triangular plane, as indicated. The detector takes images at a fixed rate.

By continuously adding these readings as columns into an image, a total scan of the fish can be produced. The produced image thus shows detector readings over time. It should be noted that this setup only produces images that correctly represent the object being scanned when the detector reading frequency matches the speed at which the conveyor belt is moving. When the reading becomes too slow or the fish moves too fast, the image does not accurately represent the real object. In the provess important information can be missed. In such a case the image might appear compressed or stretched horizontally.

The detector can be translated along the x-axis. This changes the emitter-detector plane as shown in figure 2.1. Specifically, by moving the detector along the x-axis, the angle the emitter-detector plane makes with the x-axis can be changed. By taking multiple images of the same fish where the emitter is located at different positions along the x-axis, the fish is in practice photographed from different points of view.

Two such images taken from different emitter positions represent a *stereoscopic* image

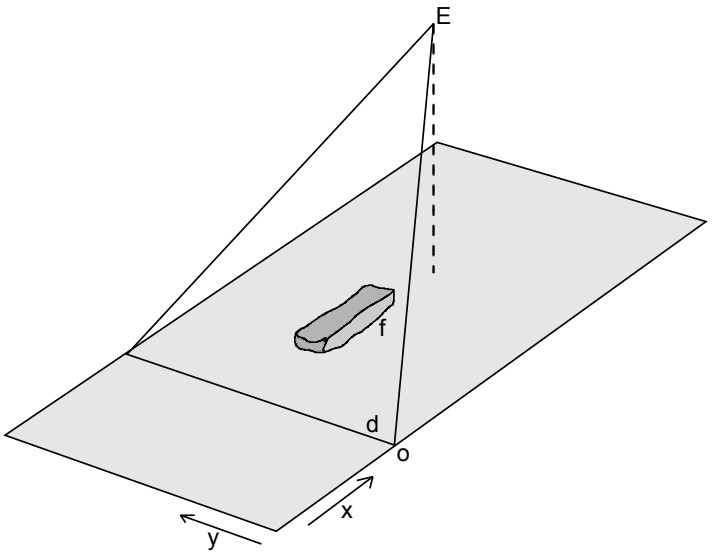


Figure 2.1: A diagram showing the most important parts of the experimental setup.

pair of the fish. If the bones are filtered out of the image pair, a reconstruction can be made about the position of the bones in 3D space.

Table 2.1 lists all variables that act as parameters to this setup. Other variables exist that influence the resulting image, but they all are calculated from the listing in the table.

The 3D positions of these bones have a very practical use. Figure 2.2 shows two pieces of raw fish that have been cut into smaller pieces. These pieces should preferably be bone free. This cutting is performed by an automated cutting machine shown in figure 2.3. If this machine knows where the bones are located inside the fish, it is able to deduce the optimal cut that yields the largest possible pieces and minimises the amount of flesh that contains bones. The deduction of such an optimal cut is not within the scope of this project.

Variable	Type	Description
D_1, D_2	Scalar	X coordinate of each of the detectors.
D_{min}, D_{max}	Scalar	Y coordinates of the start and end of the active detector area, respectively.
D	Point	A single point to denote the origin point of both detectors, used whenever both values apply.
P	Point	Location of some piece of bone relative to the origin of the bounding box of the fish. The program aims to reconstruct this point.
E	Point	Location of the emitter.
F	Point	Origin of the bounding box of the fish being scanned.
I	Point	Point that is only defined when P intersects an emitter-detector plane. Its Z coordinate is always 0 and its X coordinate is equal to F_x. Represents the location of the projected pixel on to the X-Ray image.
h	Scalar	Distance in the positive z direction that represents the distance between the detector and the bottom of the bounding box of the fish. This distance is mainly caused by the conveyor belt.
i	Point	Coordinate on X-Ray image. This point is deduced from the location of I.
R	Scalar	Units of distance represented by a single pixel. The X and Y axis are considered to have the same resolution.
v_x	Scalar	The speed of the fish in the negative x direction
f_d	Scalar	The frequency at which the detector is read off

Table 2.1: A listing of all relevant parameter variables defining the setup or defined by the setup. Their names will be used for the same concepts throughout the report.

Figure 2.2: [1] The machine that performs the cutting of the fish filets

Figure 2.3: [1] Some sample fish filets that have been cut in this setup

2.2 Previous work

Simulation of X-Ray images from 3D models has already been done before. For example, [2] discusses a method to render X-Ray images using voxels where rays are accurately simulated. For example, rays are governed by accurate physical calculations and individual voxels can be assigned individual materials. As the simulator for this project did not require such accuracy, this method was not used here. See 3.1 for a detailed explanation of which simplifications were chosen to be included into the project.

A setup identical to the one used in this project was described in [3] for use in X-Ray scanners on airports. However, this project mainly attempted to construct a setup that would allow the capturing of stereoscopic images. There was no attempt made to interpret the images automatically or to reconstruct 3D points from them.

An alternate approach is discussed in [4] where a number of linear X-Ray detectors are used instead of only two (as in this project). However, the project does not attempt to extract any 3D information, as mentioned in its conclusion. The focus of the authors appears to be to use stereoscopic X-Ray images to enhance the vision of customs officials at airports. For example, a subsequent paper [5] continues on reconstruction views from moving objects.

Chapter 3

X-Ray Simulator

This chapter documents all assumptions and methods that have been used in order to construct the X-Ray simulator. Additionally, a description of some relevant implementation details will be described. Finally, some of the limitations of the used model will be outlined.

3.1 Assumptions

A number of assumptions had to be made in order to simplify the implementation of the simulator. Each will be discussed and justified below.

The noise in the X-Ray images is Gaussian distributed

The images produced by the experimental setup had a significant amount of noise. The simulator should replicate this noise in its output. An analysis was performed on some output images to determine the nature of this noise. A distinction is made here between background noise and material noise. Background noise is noise in rays that did not intersect with anything between the emitter and detector. Material noise intersected with the fish meat, bones or both.

In figure 3.1 a histogram is shown that was generated from an image only containing background noise. If it is assumed that an image without noise has a constant colour, the histogram should show the nature of the noise distribution. The distribution is approximately shaped like a Gaussian distribution. A similar histogram has been constructed for an image that contained a part of the fish, and whose colour appeared to be approximately constant. The histogram of this image is shown in figure 3.2. This method was adapted from a method mentioned in [6].

This histogram appears to follow a Gaussian noise distribution, albeit less accurately than the background only histogram. It might be that the assumption that the chosen subimage had a constant colour does not accurately hold, causing a slight deviation. Still, both histograms suggest that assuming perfect Gaussian distributed noise for the simulated X-Ray images is justified.

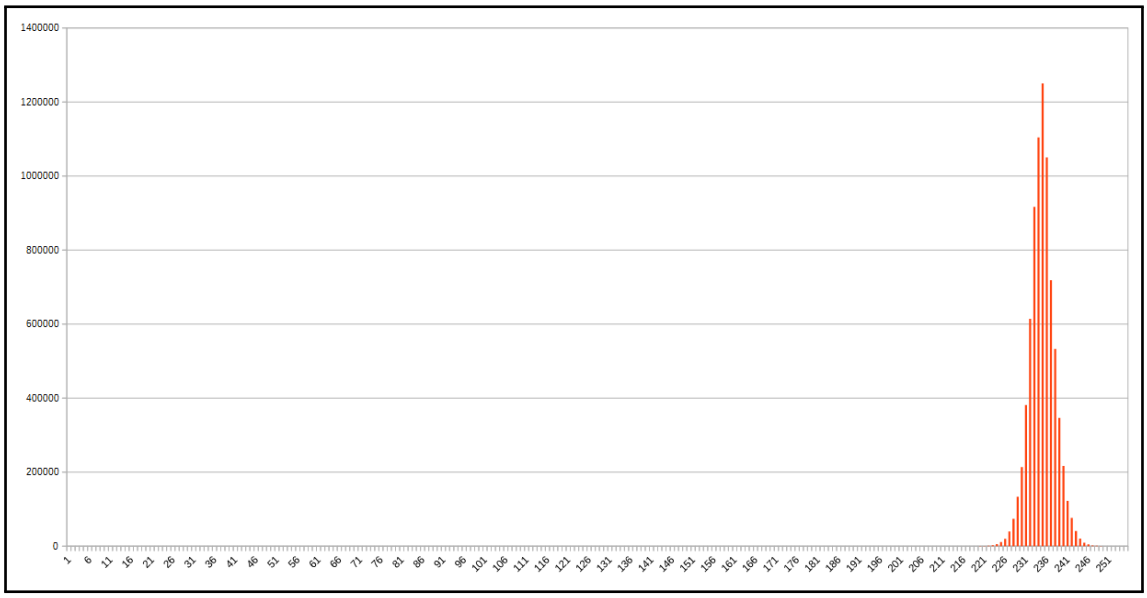


Figure 3.1: A histogram of a sample X-Ray image containing no fish flesh. The x-axis shows the 8-bit grayscale colour level.

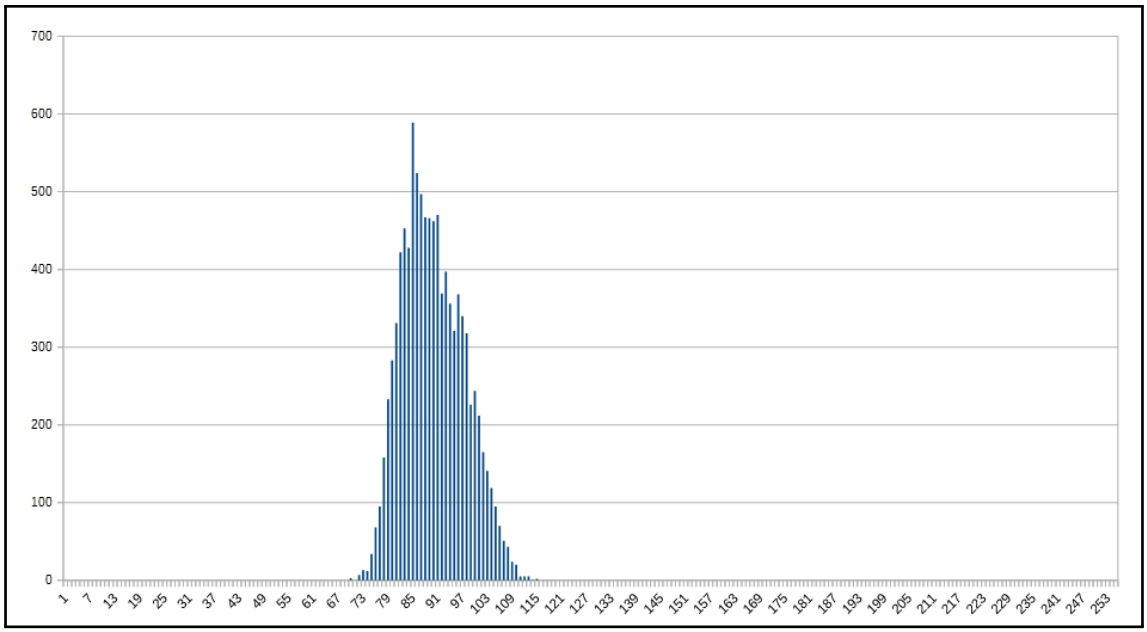


Figure 3.2: A histogram of a small piece of fish from a sample X-Ray image. The x-axis shows the 8-bit grayscale colour level.

The noise mean and standard deviation are proportional

Building on the assumption that the image noise is Gaussian distributed, it can be observed in figure 3.1 and 3.2 that the distribution appears to have varying properties depending on the density of the material it passes through. Specifically, the standard deviation of the background noise appears to be significantly lower in the background noise histogram than the fish histogram.

As the simulator should be able to apply a correct level of noise on any given depth, a method should be assumed for which the standard deviation of the noise can be calculated given any depth. The method that leads to the simplest implementation is to measure the standard deviation of the noise at two different depths, then assume that the noise standard deviation and the depth of the pixel are proportional. If the resulting images appear similar to physical scans, this method is justified. A different interpolation strategy might be needed otherwise.

These assumptions showed to give decent noise distributions in practice.

The density of the fish is distributed equally

Fish meat tends to mostly consist of fat, muscles and protein formations. This may cause deviations in the measured X-Ray intensity by the detector for a given part of the fish, even when the fish has a constant thickness. This in turn requires a more complex model for simulating the X-Rays. For example, input geometry would also need to give a specification of the consistency of the flesh at any point inside the fish filet. Instead, a simpler model is assumed in which the measured intensity of the incident X-Ray on the detector is assumed proportional to the distance the ray has travelled through the material.

The measured intensity of an X-Ray is the sum of all materials it has travelled through

There are complexities in accurately calculating the distance an X-Ray has travelled through a given material. See section 3.3 for a detailed discussion on this issue.

3.2 Description of simulator implementation

This section will outline the implementation of the X-Ray simulator.

3.2.1 Deduction of noise standard deviation function

It has been assumed that the noise mean and standard deviation are proportional. A linear function has been deduced to map the intensity of the simulated pixel (in a range of 0 to 255) to the standard deviation of the noise distribution at that pixel. The method employed for this was to calculate the mean and standard deviation in two images that were considered to have a constant colour. The histograms of the used images are shown in figure 3.1 and 3.2.

The data from the histograms was normalised, then fitted to a Gaussian curve to deduce the parameters of the distribution. The following table shows the measured values:

Image	Curve fitted mean	Curve fitted standard deviation
Background	234.8	2.5
Fish	86.7	7.2

Using these two points to deduce a linear equation yields the following function:

$$stdv(mean) = 9.95145 - 0.03174 \cdot mean$$

3.2.2 Depth determination

The simulator attempts to create an image in a similar way to the test setup of the experiment; the 3D mesh is moved along the x-axis in the negative x direction, and for every unit it has moved the detector is read. Every point on the detector then becomes a single pixel in the final image. Each mesh is first rendered separately. After that the final pixel colours are calculated by taking a weighted sum of the individual images produced by each mesh. The weights represent the relative densities of each of the meshes.

To render a single mesh, the simulator loops through all triangles in that mesh, and renders each to a depthbuffer. This depthbuffer differs from regular Z-buffers by remembering all depths for each pixel. It will subsequently be referred to as a "multilayer depth buffer".

The process of rendering a single triangle consists of determining what area of the final image the triangle will appear, then rendering that triangle in the calculated area using ray tracing. The ray tracing involves casting a ray between the emitter and a pixel on the detector and calculating the intersection (if any) with the triangle. In the simulator implementation, a ray-triangle intersection calculation algorithm described in [7] was used.

The final depth of a pixel of the final image is the sum of each pair of depths encountered in the list of intersections. It can occur that due to rounding errors an intersection calculation might have missed a triangle. In that case, when the total number of intersections along a ray is odd, the colour of the pixel is set to zero (no intersection). This was not found to be an issue in practice, but has been added as a precaution nonetheless.

3.2.3 Determination of screen space

In an early implementation of the rendering alhorithm, the whole model was tested against the emitter-detector ray every single pixel in the image. This method proved to be extremely slow even for small images.

In order to reduce the running time of the algorithm, a calculation was added prior to rendering the triangle to determine the screen space the triangle occupied in the final

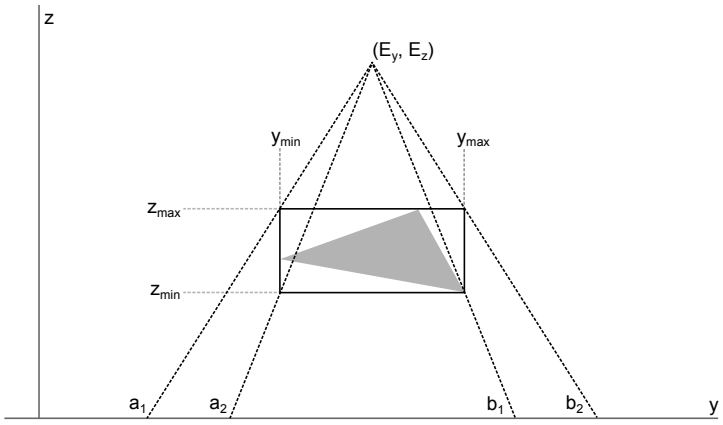


Figure 3.3: A diagram showing the determination of the screen space occupied by a single triangle on the final image on the y-axis.

image. Next, the basic algorithm was used only on the space on screen on which the triangle would appear.

As the number of pixels occupied by a single triangle was found to often be low, this caused a very significant decrease in the execution time of the rendering algorithm. The deduction of the performed calculations in this process will be discussed.

The space occupied by a considered triangle on screen is a rectangular area on the X and Y plane. As the methods for determining the used space on each of these axis differ, each of these axis will be described separately.

Y-axis screen space range

At the center of figure 3.3 a triangle is shown along with its bounding box. Only the bounding box is considered when calculating the used screen space. The detector is assumed to always be at z = 0 and always running along the y-axis.

The emitter is a point source located at (E_y, E_z). From the emitter, four rays are cast that intersect with each of the corners of the bounding box. These rays are intersected with the detector, resulting in two pairs of two y-coordinates (marked by a and b in the diagram).

Finally, the range on the y-axis in which the triangle will appear is determined by taking the minimum value of a_1 and a_2, and the maximum value of b_1 and b_2.

The calculation of the intersection of the rays from the emitter that pass through the corners of the triangle's bounding box is shown in figure 3.4. In the diagram, point c represents any corner of the bounding box of the triangle. The intersection point of a ray cast from E through c with the detector is (i_y, 0).

First, two deltas are calculated between E and c; one for each axis. Next, the delta in the y direction is scaled by a factor of $\frac{E_z}{dz}$. This gives the absolute distance between E_y

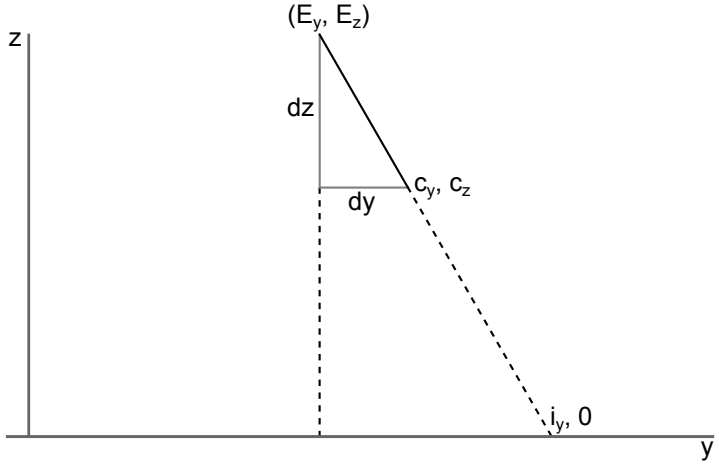


Figure 3.4: A diagram showing the projection of a ray from the emitter to a corner of the triangle's bounding box on the detector.

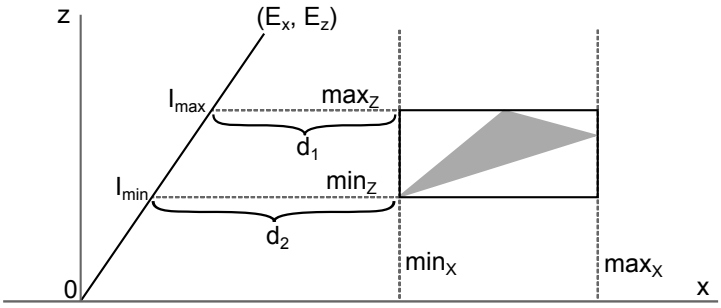


Figure 3.5: A diagram showing the determination of the screen space occupied by a single triangle on the final image on the x-axis.

and i_y. To find i_y, E_y has to be added to the final result.

As a single equation:

$$i_y = E_y + \frac{E_z \times dy}{dz} \tag{3.1}$$

This equation can be used to calculate the values of a_1, a_2, b_1 and b_2. The range along the y-axis along which the triangle can appear is determined by the values of a and b that are furthest apart. Using the calculated values of the a and b pairs gives the range:

$$[min(a_1, a_2), max(b_1, b_2))] \tag{3.2}$$

X-axis screen space range

The x-coordinate of a pixel on the rendered image represents the distance the fish has travelled through the detector. The calculation for the range of x coordinates on which the considered triangle will appear thus involves predicting at which x coordinates the

triangle will intersect with the emitter - detector plane. Note that the detector is always located at the origin.

Like the calculation of the y-axis range, a bounding box is calculated from the considered triangle. The occupied space in the image is then deduced from this bounding box.

In figure 3.5 two distances have been marked that represent the distance from the left hand corners of the bounding box to their respective intersection point with the emitter - detector plane. These distances have been marked with d_1 and d_2. The right hand side corners are calculated in a similar way.

Consider any bounding box corner c on the x-z plane (c_x, c_z). This point has to travel a distance d in the negative x direction before it intersects with the X-Ray plane. The intersection point I can be calculated by considering the X-Ray plane a linear function of the z coordinate.

This linear function is:

$$I = \frac{E_x}{E_z} \times c_z \tag{3.3}$$

The slope is defined only by location of the emitter, as the detector is located at the origin. The value of d can be calculated by subtracting I from c_x:

$$d = c_x - \frac{E_x}{E_z} \times c_z \tag{3.4}$$

The range of x coordinates in which the bounding box of the considered triangle will intersect the X-Ray plane lies between the first and last x-coordinate that intersects with the plane.

Considering d as a function taking a point on the x-z plane, the range is thus:

$$[min(d(min_x, min_z), d(min_x, max_z)), max(d(max_x, min_z), d(max_x, max_z))] \tag{3.5}$$

Where the parameters of d refer to values shown in figure 3.5.

Calculation of area on image

To construct the screen space area that has to be rendered, the calculated x and y coordinate ranges have to be rounded to integer values. This has to be done because the rendered image uses integer coordinates for all pixels. For both ranges, the start coordinates are rounded down and the end coordinates are rounded up.

3.2.4 Generation of image

To render an X-Ray image, the simulator first translates and scales all meshes such that they fit into the image and the total bounding box is aligned with the image's origin. This removes the requirement that models are of a certain scale. However, meshes are required to be in the same relative scale. They also need to have the same rotation.

Next, an image is generated for each model containing the read X-Ray intensity at every coordinate that is included in the final image. These images are added together as a weighted sum. Finally, the resulting image is normalised and converted into a grayscale image.

Normalising the image ensures that the image is always taken in an optimal way, without being over or underexposed. This is not entirely an accurate reproduction of an actual X-Ray image, which may show such deficiencies.

3.3 Limitations

There are some known limitations to the current setup:

Noise only measured from single image

For estimating the noise present in the X-Ray images, only a single sample image was used. This could result in a possible statistical bias in the obtained values for the noise standard deviation at different means. However, because the amount of pixels (samples) was very large, and the noise of the setup should approximately be the same for every image, measuring the noise from only a single image seems justified.

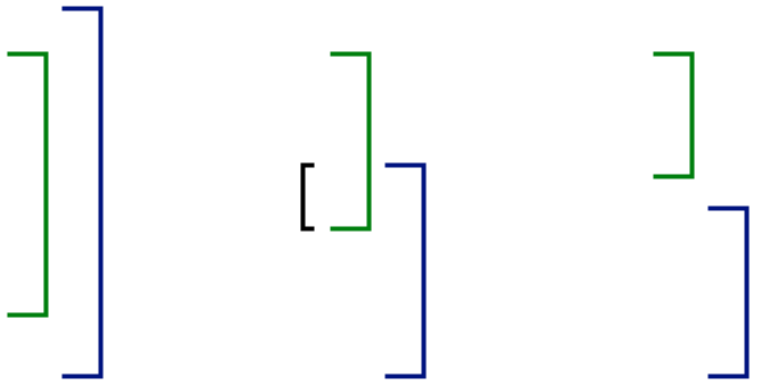
Oversimplified X-Ray intensity calculation

The simulator assumes that the intensity is the weighted sum of individual intensities. This is not an accurate representation. Figure 3.6 shows three combinations of the interaction between two objects, each having two intersections with the X-Ray. The most common situation in the case of a fish is 3.6a, as the bones (green) will most likely be inside the flesh of the fish (blue). In the simulator, situation 3.6a and 3.6b are assumed to be the same as 3.6c.

In this case the correct measured intensity of the X-Ray is: $I = \alpha\Delta Green + \beta(\Delta Blue - \Delta Green)$, where α is the density of Green and β is the density of Blue. It should also be noted that in this case there is a question of *precedence*: is one object inside another, or the other way round? Just the measured ray entry and exit depths by themselves do not reveal this information. In this example, the Green object has taken precedence.

In 3.6b the objects partially overlap. The intensity of the X-Ray in this case is $I = \alpha Green - \beta(Blue - Black)$, where Black represents the distance that the two objects overlap. In this case there is also a question of precedence. Here Blue has taken precedence.

When more than two objects are considered, the intensity calculations become very complex. A simple weighted sum (thus assuming exclusion) is therefore a good tradeoff that prefers speed over accuracy. However, future work could include more accurate intensity calculations.



(a) Inclusion (b) Overlap (c) Exclusion

Figure 3.6: Different possibilities for the intersection of two different objects

Large memory usage

The multilayer depth buffer allocated by the simulator works very fast, but also requires a very large amount of memory. The amount of memory needed for these depth buffers is: image width × image height × depth × (bytes per depth value) × (mesh count). With large images, the required amounts quickly become very large.

Chapter 4

Bone filtering

4.1 Process overview

The process of identifying bones in the X-Ray images consists of a series of filters. The aim of these filters is to extract a binary image solely consisting of pixels belonging to bones. Next, two images that are made by different detectors must be compared to create pairs of points. Each pair represents the location of where a single piece of bone was projected on to the detector. Finally, using the point pairs as well as known information about the setup, a 3D point cloud can be calculated that represents the location of the piece of bone in 3D space.

4.2 Stereoscopic reconstruction

Image rectification attempts to greatly simplify the process of unprojecting two or more stereo images by transforming the images such that all point pairs will have the same Y coordinate. This makes the process of searching for matching pairs of points much easier due to the greatly reduced search space. One of the assumptions used in the unprojection calculations is that the point pairs have the same Y coordinate horizontally. In this sense the X-Ray produced by the described setup can already considered to be rectified.

If in practice the detectors are not aligned this way, a rectification method will have to be utilised. There are a number of rectification methods that are commonly used to this end.

Planar
: Planar rectification attempts to rectify two stereo images by drawing them in 3D space through some transformation [8]. The planar name thus comes from transforming the image as a whole rather than transforming individual pixels. The method assumes that the two cameras the stereo pair is captured with are calibrated. E.g. the location and orientation of both cameras are known. This can be assumed to be the case for the used X-Ray setup.

The major downside of this method is that it does not work well on cameras that are far apart [9]. In such cases it may be much more difficult to obtain accurate coordinate pairs from the rectified images. For example, in some setups the resulting rectified images may be severely compressed. This means that in the rectification process a lot of useful information is lost that can not be represented in the unprojection.

Cylindrical

Unlike planar rectification, cylindrical rectification does not assume that the cameras are calibrated [10]. Note that the method can be slightly simplified if the cameras are calibrated. The method aims to rectify images by mapping the images produced by the two cameras on to cylinders that are directed along a common central axis.

One of the main advantages of cylindrical rectification over planar images is it attempts to minimise the information loss per pixel during the rectification process. Moreover, the rectified images are of constant size. Depending on the placement of the cameras, images rectified using the planar method can grow arbitrarily large.

The main downside of this method, as [11] claims, is that the operations that have to be performed on the images are relatively complicated. Additionally, the method assumes that both cameras are viewing the scene from approximately the same angle or side.

Polar

The polar line rectification is the only method discussed here that garuantees that no information is lost in the rectification process [11]. Moreover, the size of the rectified image is bounded by $(\sqrt{W^2 + H^2}, 2 \times (W + H))$, where W and H are the width and height of the original image, respectively. It thus garuantees that images can not become infinitely large, as can be the case with the planar method.

It functions by transforming the image pixels from X-Y coordinate space to a polar coordinate space (hence the name) using a series of linear transformations. Spaces are filled up by interpolating pixels around it. The paper presenting the method claims that the implementation is much simpler than the Cylindrical method.

The applicability of these methods on the X-Ray scanner is unknown. The main reason for this is that all described methods all assume cameras with perspective projections. The X-Ray scanning setup uses a combination of these. The x-axis is an orthogonal projection while the y-axis is a perspective projection. The view frustrum of this camera is shown in figure 4.1. Due to this different view frustrum, a custom method might need to be developed to rectify the stereo images properly.

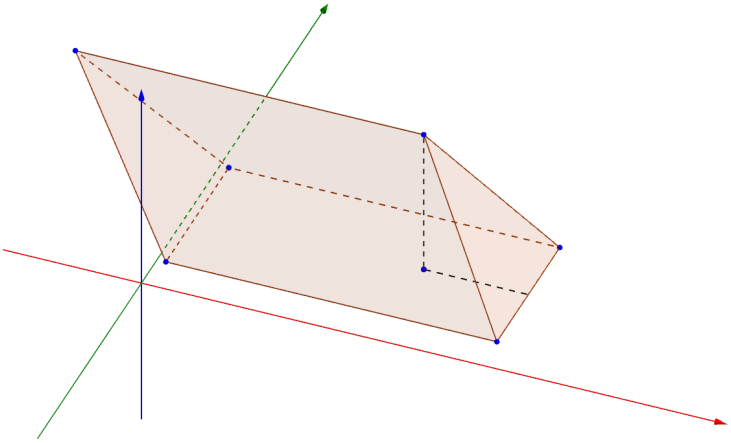


Figure 4.1: A representation of the view frustrum of the X-Ray scanner.

4.3 Filtering of bones

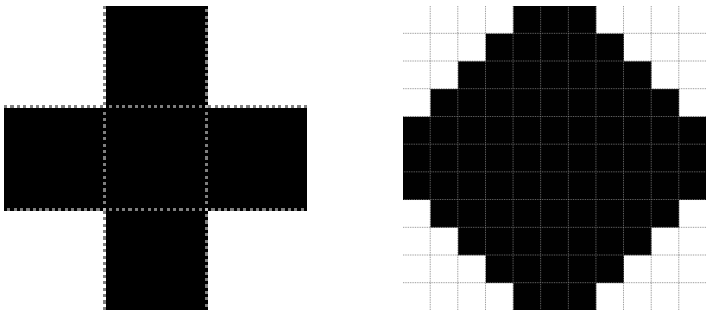
The process of filtering fish bones out of the X-Ray images consists of the applying a number of basic filters. The filters, their order and parameters are outlined below. Note that each filter is applied on the output of the previous filter unless stated otherwise. They will be discussed and justified in detail afterwards.

4.3.1 Filtering process overview

The kernels referred to by the process are shown below. A black pixel means the pixel is "active", while a white one means the pixel is ignored.

1. **Gaussian blur**
   5x5 kernel

2. **Dilation**
   Using a large kernel



(a) The small kernel (b) The large kernel

Figure 4.2: The two kernels used in the filtering process

3. **Erosion**
   Using a large kernel

4. **Absolute Difference**
   Between the original image and the result of the previous erosion step

5. **Sharpen**
   Unsharp mask using a 3x3 Gaussian blur. The input image to this step is weighted 1.5, and the blurred image 0.5.

6. **Erosion**
   Erosion using a small kernel.

7. **Gaussian blur**
   5x5 kernel

8. **Threshold**
   Using Otsu's method [12] to find a suitable threshold

9. **Skeletonisation**
   Using a custom algorithm.

10. **Line recognition**
    Reading out line segments from the skeletonised image.

11. **Bone reconstruction**
    Merging line segments together and filtering out noise

4.3.2 Bit depth

In order to maintain accuracy, the implementation of the filtering process has been done with a 16-bit depth. Many of the X-Ray scans made with a physical setup had 16-bit accuracy, so the decision was made to attempt to preserve this accuracy throughout the filtering process. Step 1 to 7 all support 16-bit depth in the implementation. Step 8 does not. The bit depth is not relevant for later steps.

4.3.3 Detailed filtering process

The image filtering process will now be discussed in detail and rationalised. Any step numbers referenced in this description refer to the numbers listed in section 4.3.1.

The X-Ray image shown in figure 4.3 will be used as an example to show the results of the various processing steps.

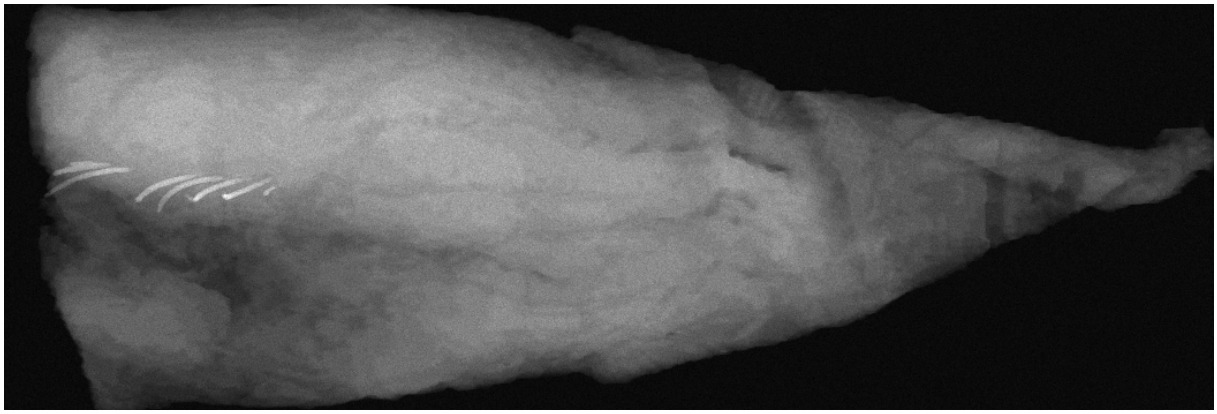


Figure 4.3: An X-Ray image generated by the simulator.

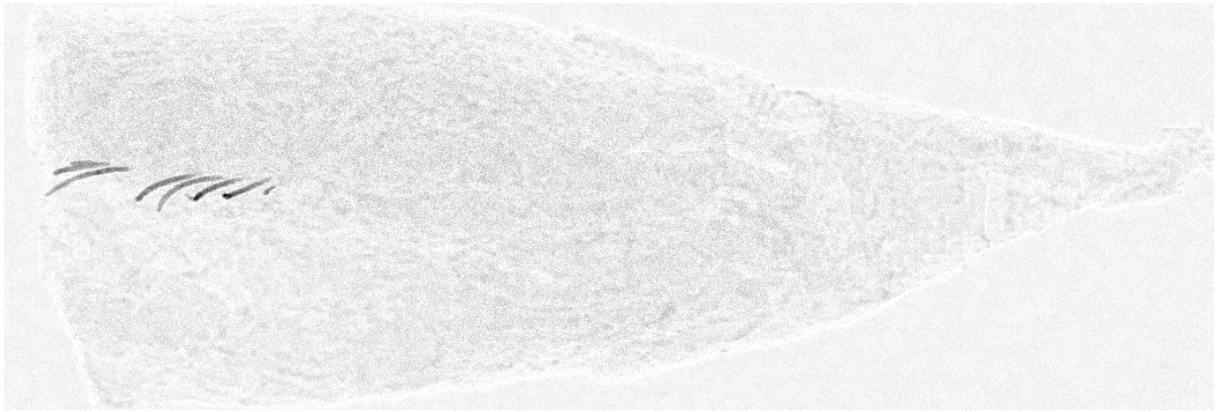


Figure 4.4: An X-Ray image after the difference step that was not blurred.

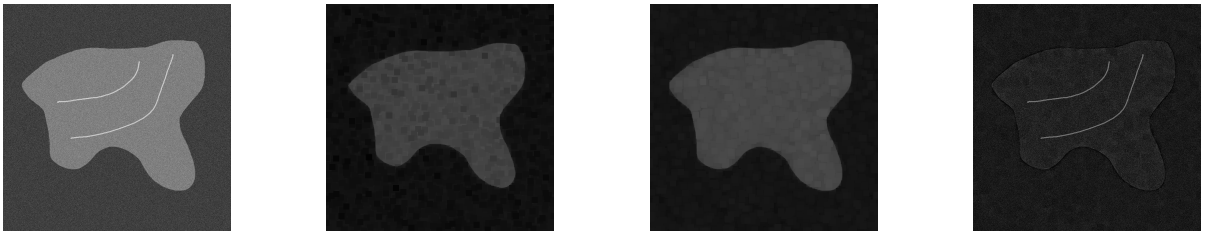
Blur (step 1)

The first step in the filtering process is the application of a blur filter. As the input X-Ray images are generally very noisy, this step will take out the strongest outlying pixels. The utility of this filter step is shown in figure 4.4. This image was generated by starting the image filter process at step 2, thus skipping the blur. Comparing this image to the one shown in figure 4.6, a significant difference in the noise level can be observed.

Some of the test images used in the project reacted heavily to this noise in the thresholding step. The noise thus became visible on images that should only contain pixels belonging to bones. The addition of the blur filter step before taking the absolute difference between images was considered a good solution to reducing the noise levels.

Removal of unrelated areas (step 2, 3 and 4)

This step attempts to filter away all parts of the image that do not belong to a bone. For a more comprehensive overview, the effect of the filters applied during this step are shown in their respective order in figure 4.5.



(a) Original (b) Dilated (c) Eroded (d) Difference

Figure 4.5: The filters applied during step 2, 3 and 4 on a sample image containing bones.

Figure 4.6: The X-Ray image after step 4 has been applied.

On X-Ray images, bones generally have a higher intensity than their immediate surroundings. Additionally, bones are considered thin line segments. If a morphological dilation is applied on an image, the relatively thin bones can be reliably removed, given that the dilation kernel is large. Because the dilation filter prefers smaller intensities, it tends to choose pixels that lie around the bones rather than the pixels that belong to the bones themselves.

Next, an erosion filter is applied on the dilated image to revert the effects of the dilation. The result is a mask that represents all background artefacts. Thus, taking the difference with the original significantly reduces all non-bone elements present on the image, while closely maintaining the shape of the bones themselves. Additionally, because the bones have a relatively high intensity compared to the background mask, their relative intensity remains high when calculating the absolute difference.

This effect of this step is that the intensity of the bones is increased relative to the background, making it significantly easier to perform a threshold operation later. Section 4.3.4 presents an alternate method for performing this step.

Removal of intense noise spots (step 5 and 6)

After step 4 there is often still a significant level of noise present in the image. This noise can be seen on figure 4.6. As the thresholding step will show, it is essential that the noise is reduced as much as possible to ensure the thresholding yields good results. The noise

Figure 4.7: The X-Ray image after step 6 has been applied.

reduction done in this step is done using two filters.

First, a sharpen filter is applied. This filter creates a larger separation between the intense and dark pixels on the image. The kernels used for this as well as the erosion filter that follows it are only 3x3 pixels large, so that only the immediate neighbours are relevant to the result. Larger kernels could eliminate the noise almost completely, but lost a lot of detail on the bones.

The erosion step that follows benefits from the sharpening, because the high intensity pixels are now "correcting" the noisy low intensity ones when they are picked up by the max function. This significantly reduces the noise level of the image. It also affects the bones, albeit not significantly.

Thresholding (step 7 and 8)

The next step attempts to convert the filtered X-Ray image into a binary image. The filtering process uses a global threshold that is determined by Otsu's method [12]. An adaptive thresholding method was considered, but it proved not to be suitable to this particular case because it reacted too heavily to the noise that remained on the image. This is demonstrated in figure 4.8.

The issue with using a global threshold is that as much of the bones should appear on the resulting image as possible, while keeping out as much of the noise as possible. Otsu's method allows the determination of an appropriate threshold level using the image's histogram. The algorithm was found to produce decent threshold levels in practice.

The choice for a global thresholding method also requires that the bones are distinguishing themselves clearly from any noise. Thus a noise reduction step (step 5 and 6) was needed. From experimentation it was found that using a blur filter before performing the threshold resulted in less noise spots being left on the binary image while ensuring that some of the pixels eroded away in the noise reduction step that belonged to bone were filled up, thus being included in the thresholded image.

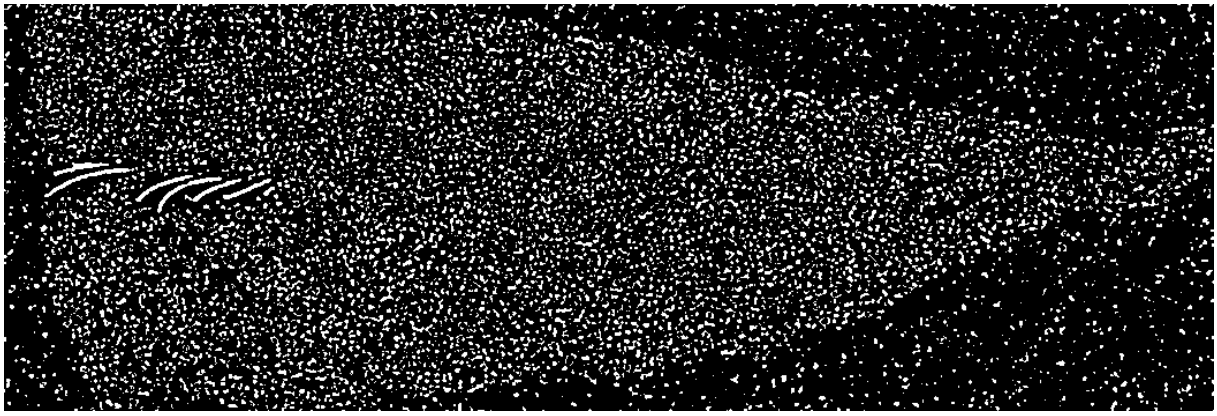


Figure 4.8: An X-Ray image that has been thresholded using the adaptive gaussian method.

Figure 4.9: The X-Ray image after the image has been thresholded.

Figure 4.10: The X-Ray image after the image has been skeletonised.

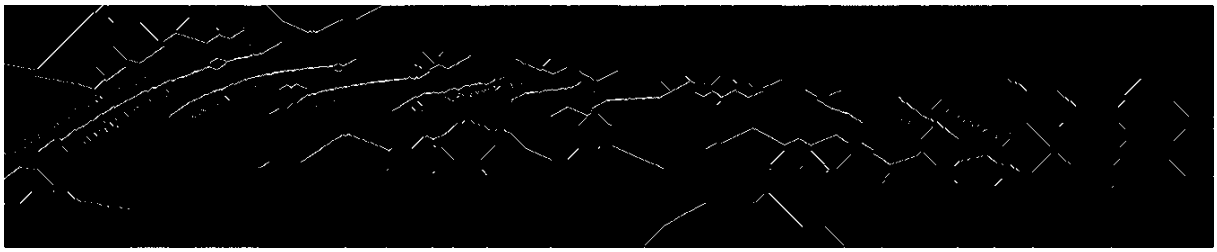


Figure 4.11: An X-Ray image that has been skeletonised using morphological filters.

Skeletonisation (step 9)

As can be seen in figure 4.9, the image coming in to this step should mostly contain bones relative to a minimal amount of noise. The next stage in processing the image is to assume that all low intensity pixels remaining in the image belong to bones. These pixels must now be converted into line segments that represent a projection of the center of each bone. This simplifies the process of generating a 3D point cloud from the stereoscopic images as there is a smaller number of points that has to be matched. The method commonly used for this is a skeletonisation filter.

There are a number of algorithms for skeletonising an image. The main problem with many of these is that they attempt to create a single connected web of connections. This means that often bones will be connected together due to their often close placement in the image. Moreover, the approach attempted to include noise patches or create new line segments entirely. An example is shown in figure 4.11. While the image should only have contained a small number of bones, a large amount of new line segments have been added, making the result completely unusable.

A simplified approach was adopted instead. For every vertical sequence of pixels, the average height is calculated. This sequence is then substituted by a single pixel. All averages are rounded down. Figure 4.12 shows the result of this method. A piece of bone, represented by black pixels is replaced by its skeleton representation marked by red pixels. The figure also shows that this method gives suboptimal results when presented with a bone that is partially directed vertically.

As such bones were not present in the test set of X-Ray images, this was not considered to be an issue. Additionally, the subsequent bone reconstruction processing stage resolves the discontinuity that can occur in that case. The algorithm proved to provide a decent skeletonisation at a relatively small performance cost.

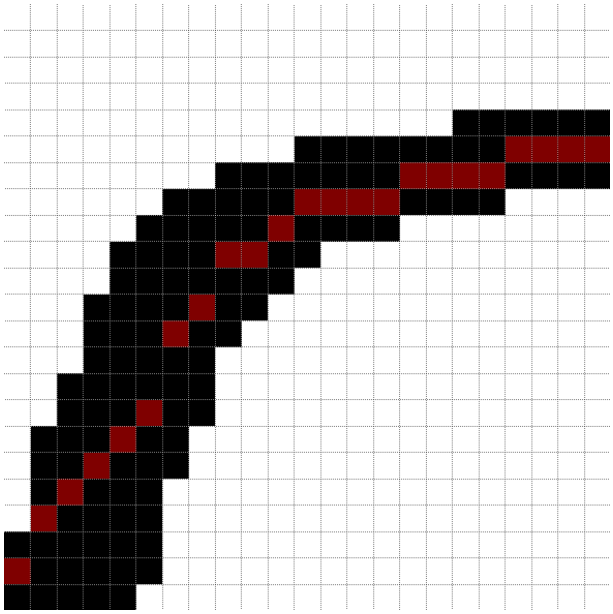


Figure 4.12: A section of a bone that has been skeletonised using the vertical average skeletonisation algorithm.

Line recognition and reconstruction (step 10 and 11)

The skeletonisation algorithm has as an additional benefit that it always produces 8-connected line segments given that the bones are mostly horizontally oriented. This eases the process of converting them into abstract line segments as all line segments can be extracted from the image without issues.

The line segments present on this image, however, do not represent the complete bones that were present on the input image. Figure 4.12 shows gaps between some bones that appear to belong together. These represent details that were lost in the filtering process. One method that was considered for closing these gaps was a morphological close operation between step 8 and 9 (before the skeletonisation). However, this operation could also connect different bones that were close together or even overlapping together, discarding useful information.

Instead, a bone reconstruction step was introduced after converting the skeletonised image into line segments. An example of such line segments is shown in figure 4.13. Note that there is a gap present in the center of the image, and that the line segments shown appear to belong to the same bone. The bone reconstruction algorithm attempts to reconnect these disconnected parts.

Figure 4.14 shows the reconnection process. First, a certain number of points at the end

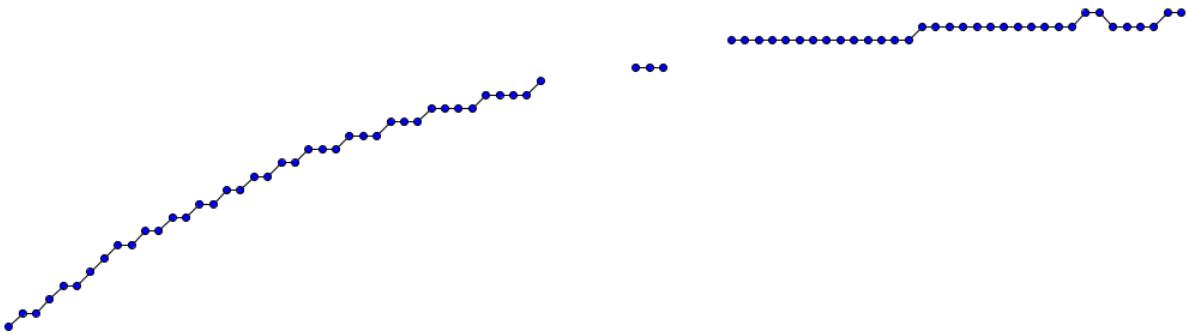


Figure 4.13: A single bone converted to line segments.

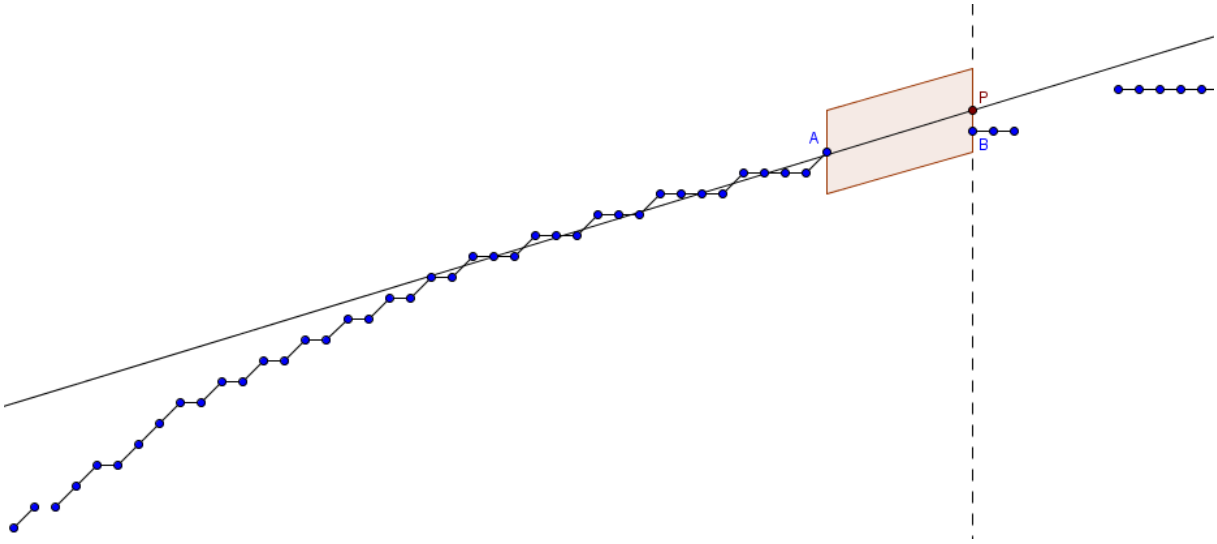


Figure 4.14: Two pieces of a bone being connected together.

of a line segment are used to calculate a best-fit line. In this case, the last 10 points of line segment A are used. This gives a trend line towards where possible other parts might be found. Next, the algorithm looks for another line segment that starts to the right of the current one, here: segment B. The starting point of that segment is projected on to the best-fit line (point P).

If the difference in X coordinates of the last point in A and first point in B is smaller than some threshold, and the difference in Y coordinates between B and P is smaller than some other threshold, B can be considered an extension of A. The shaded area in figure 4.14 shows where the starting points of other line segments would have to lie inside for them to be connected to A. After a new segment is connected, the best-fit line is recalculated to inlcude the added line segment.

This process is also done for the opposite direction. The two thresholds can be varied on demand. It should be noted that larger thresholds allow more room for deviations that may have been introduced in the filtering process, but may also result in line segments being connected together that belong to different bones.

A final step that is done in the implementation is to remove bones that are shorter than a

Figure 4.15: The X-Ray image after line recognition and bone reconstruction has been applied.

certain number of points. As some areas in the input images can cause these to propagate through the filtering process as bones, this acts as a final noise removal filter. Testing showed that such noise patches often resulted in bones that were very small. Thus a filtering step for very small bones is an effective way to remove them.

Process completion

After the bone reconstruction is complete, the result is a set of line segments that each represent a single bone. The result is shown in figure 4.15. Combined with another image to form a stereoscopic pair, this can be used to create a 3D point cloud reconstruction. However, combining the images into point pairs and performing the unprojection is beyond the scope of this project.

4.3.4 Alternate method for step 2, 3 and 4

An alternate method for removing the areas unrelated to bones was investigated in detail. It attempts to exploit two features of the image:

1. Bones have a much higher intensity that their surrounding pixels. The intensity around the bone drops quickly.
2. The intensity around the fish itself changes gradually.

Areas on the image that have a relatively high difference in intensities have higher variances. If variances are used as values of pixels, it might be possible to extract the bones from an image with a single operation. Moreover, this method should also run faster due to a smaller number of filters having to be applied. Unfortunately, applying a variance filter by itself proved not to give useful results as shown in figure 4.16. As can be seen on 4.16a, there is a very significant drop in intensity between the fish and its background. This drop is much greater than the area around the bones.

(a) Original (b) Variance using a 13x13 kernel.

Figure 4.16: A basic variance filter applied on an X-Ray image

(a) Original X-Ray scan (b) Normalised variance (c) Threshold step of bone extraction process.

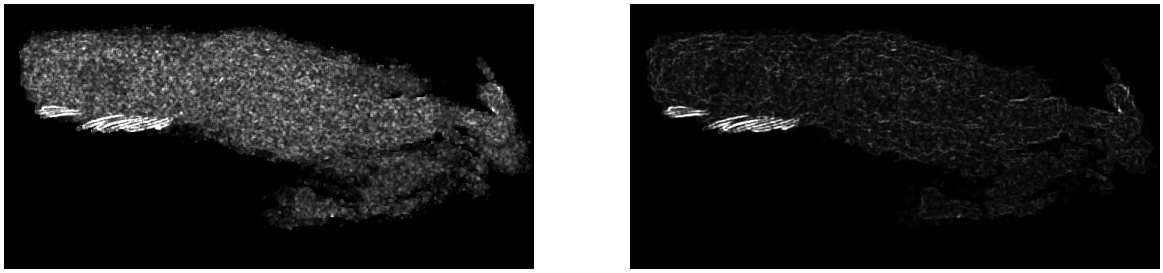
Figure 4.17: An X-Ray image of flower decoration foam processed with two bone extraction methods

The result of applying the basic variance filter is shown in figure 4.16b. The bones are no longer visible in the image. In order to make the bones clearly visible, the background pixels should not be included in the variance calculation. This approach proved much more successful. The algorithm calculates the variance only over pixels inside the kernel that have an intensity lower than some threshold. It assumes that there is a clear intensity gap between the fish and its background, so that this measure effectively separates the pixels belonging to the fish from its background.

A very good result obtained with this algorithm is shown in figure 4.17b. The X-Ray image was made with flower decoration foam in which had been inserted some dense objects to act as bones. The variance algorithm gives a near-perfect binarisation, while the process described earlier includes a very large amount of noise. It should be noted here that this test image does not contain many of the irregularities seen in images of actual fish, making it easier to get a high quality binarisation of the image. It should be noted that the image has relatively solid colours, so bone extraction is relatively simple compared to actual fish.

Even though a number of images gave very positive results, this method has a number of significant problems. These problems led to the method background filtering process using morphological filters described in 4.3.3 was chosen in favour of the variance method.

The first problem is selecting a good threshold. If the threshold is too low, too few pixels will be included in the variance calculation, thus likely leaving out parts of the bones.



(a) Not blurred (b) Blurred

Figure 4.18: Variance of an image that has been blurred versus one that was not blurred

At the same time, higher thresholds will include more of the fish flesh, which in itself is relatively noisy. Variance is susceptible to noise due to the nature of its derivation. It was found that different images have different ideal thresholds that balance the noise level and the visibility of the bones.

Figure 4.18 shows a solution that was also appliable on the bone filtering process: applying a blur filter prior to performing the bone extraction to reduce the noise level. This also showed to have positive results with the variance filter. In many cases the image could be thresholded directly with decent results without additional filtering, something that was required with the morphological approach.

Unfortunately, good results could only be obtained by manually picking the optimal threshold. No good method was found to do this automatically. In the worst case finding this threshold involved running the filter a number of times with different thresholds, and attempting to evaluate the result to find one that best suited the image. This process would likely be quite complex and might even lose the speed advantage of the variance method, the viability of the variance method was significantly reduced.

Disregarding the problem of selecting a good threshold and assuming this can be done with little difficulty, some problems still remain. Most notably, due to the nature of variance, this method does not detect the bones themselves but their edges. If bones are located very close to each other, these outlines could merge together, making it very difficult to extract the locations of the actual bones. An example of this can be seen in figure 4.18b. This effect is increased for larger kernels.

Second, on many images the ideal threshold still often lets through a number of areas that are not bones. These are propagated through the conversion to a binary image. Filtering out these areas is not a straightforward problem as experimentation showed that the areas are often shaped similar to actual bones. Exploiting properties of these areas to filter them out is thus a difficult problem.

Finally, because the background is left out of the variance calculations bones that are partially sticking out of the flesh are not caught by the variance filter, and appeared to have disappeared on the filtered images. Because of the many difficulties this method had compared to the method described in 4.3.3, the variance method was discarded.

Chapter 5

Conclusions

5.1 Performance

The performance of the bone filter implementation has been measured in different ways. Table 5.1 shows the average time a filter required to process an image of the listed resolution. Each image was processed 10000 times to ensure accurate measurements. The test was done on an Intel® Core 2 Quad™ Q8200 CPU. It should be noted that some parts of the implementation were not fully optimised, so an improved implementation might be able to get better results.

Moreover, for relatively small images such as the 1050x352 resolution, filtering two images sequentially would be expected to take approximately 0.154 seconds. If the unprojection algorithm takes a similar amount of time, the 3D point cloud of the bones can be extracted from the stereoscopic images within half a second.

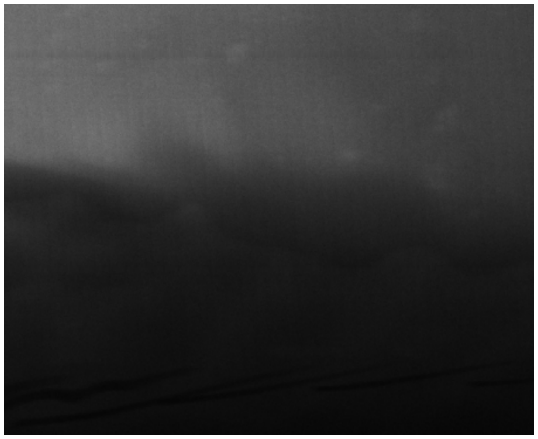
Image resolution	**Average processing time (s)**	**Average time per pixel (s)**
1050x352	0.077	2.083333×10^{-7}
1721x473	0.183	2.248066×10^{-7}
1608x1095	0.369	2.095686×10^{-7}

Table 5.1: Average run time of the bone filter implementation measured over 10000 runs.

5.2 Limitations of method

Experimentation with input images showed that the described bone extraction method requires the input image to have a number of features:

Bones must have a clear contrast with the background
The current algorithm can not effectively extract bones if the contrast between the bones and the backgound is not high enough. An example of images with high and



(a) Original, low contrast

(b) Difference, low contrast

(c) Original, high contrast

(d) Difference, high contrast

Figure 5.1: The original and difference step of the bone extraction process, for an image with low and high contrast between the bones and background

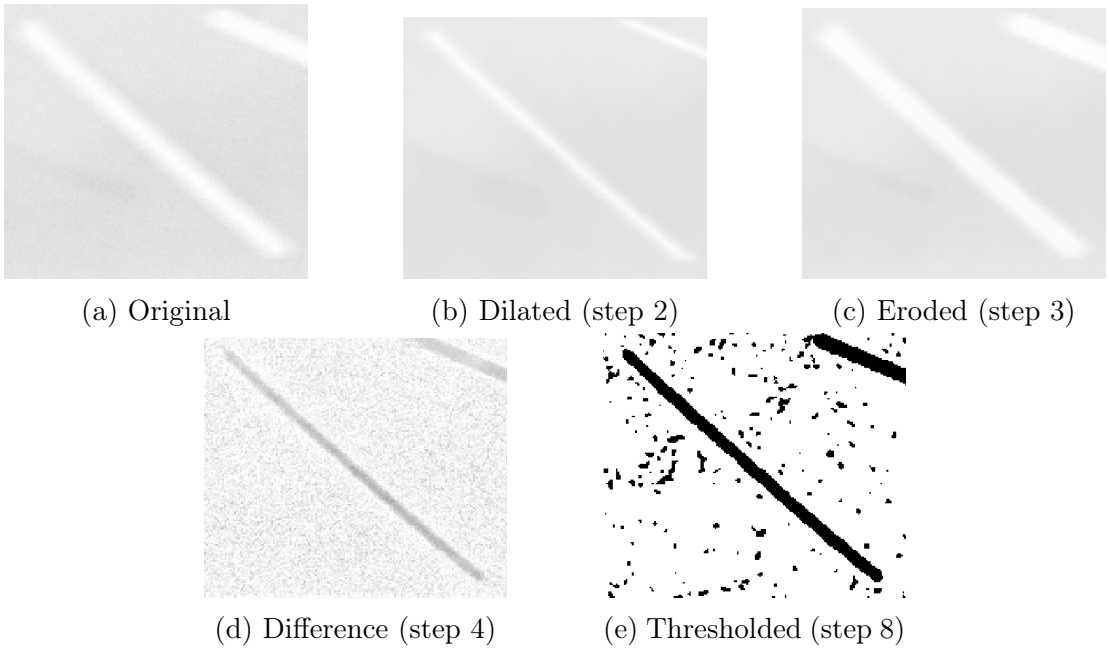
low contrast is shown in figure 5.1. When the difference step (step 4) is applied, in principle the difference is calculated between the bone and its background, as explained earlier. When this difference is not very large, the noise present in the image becomes more significant.

In figure 5.1b a significant amount of noise can be observed at the higher part of the image, that has approximately the same intensity as the bones near the bottom. Unfortunately, the measures that have been put in place to reduce the noise still present after the difference step are not effective enough to eliminate the noise without erasing the bones in the process.

Figure 5.1d shows a much lower noise level as a direct result of the bones having a much higher contrast to their background. Notice that both 5.1b and 5.1d have been normalised.

Bones must be thin

Bones must be thin relative to the kernel size used on the dilation and erosion step (step 2 and 3). Figure 5.2 shows various steps in the filtering process of an image whose bones are relatively thick compared to the filtering kernel. As can be seen in the figure, the bone is not dilated away in step 2. The erosion step then causes it to return to its original shape. The resulting difference (step 4) does not come from the contrast the bone makes with its background, but rather inconsistent intensities inside the bone itself.



(a) Original (b) Dilated (step 2) (c) Eroded (step 3)

(d) Difference (step 4) (e) Thresholded (step 8)

Figure 5.2: Various steps in the bone filtering process of an image where the bones were too thick.

Relative to the noise of the image this difference is not very significant, and the resulting partition still shows a significant amount of noise as a result. This effect is influenced by the kernel size, as well as the resolution of the image itself (a higher resolution means more pixels per bone).

Bones must be separated

When bones are partially overlapping, the skeletonisation algorithm used can not distinguish between different bones. In figure 5.3 can be seen that some overlapping bones are reduced to a single bone only. It should be noted here that the end points of the overlapping bones are fairly light gray, and might have been filtered away as noise before reaching the skeletonisation algorithm. The algorithm thus does not have all relevant information for producing an optimal skeleton.

No vertical bones

This was a limitation described in section 4.3.3, step 9. When the bones are oriented verically, the used skeletonisation algorithm produces non-ideal skeletons. However, if the vertical distance of the bone is short, it is likely that the bone reconstruction step will still be able to fill in the missing pixels.

(a) Original

(b) Extracted bone

Figure 5.3: An image with overlapping bones and the corresponding output of the bone extractor.

5.3 Conclusion

The project has delivered a method that successfully extracts bones from most images, given some conditions. It does this within a reasonable period of time. Additionally, an X-Ray simulator has been constructed successfully, which also achieves the goals that it set out to do.

5.4 Future work

A number of issues have been left to be expanded upon later.

Most notably, the final step in the X-Ray image analysis has not been included into the scope of this project. The images generated by the image filtering process need to be combined into pairs of points, that can be unprojected into 3D space using the equations described in this report. A method for performing this matching of point pairs has not yet been investigated in detail. Additionally, a possible method for rectifying images has not been investigated in detail either.

As discussed in section 5.2, the developed method is not guaranteed to give good results at all times. Future work could attempt to improve bone extraction when bones have a low contrast relative to their background. Additionally, the skeletonisation algorithm can be improved to be able to recognise overlapping bones.

Finally, a number of X-Ray images in the used test set contained fish pieces that were relatively small compared to the size of the image. Developing an algorithm that automatically crops the image to "relevant" areas may therefore prove a worthwhile optimisation.

5.5 Acknowledgements

The simulator implementation contains a triangle-line intersection algorithm that was taken from [7]. It also contains an event system that was initially written by the Author for the NTNU course "IT1901", although contains minor changes by Henning Wold and Håvard Eidheim. For an overview of these changes, see this webpage.

A number of X-Ray images presented in this report were originally made by Valka EHF. They have been used with permission.

The following libraries have been used in the different programs that were written as part of this project:

LWJGL
Light-Weight Java Game Library. Provides OpenGL bindings and input interfaces.
http://www.lwjgl.org/

OpenCV
Image processing library. Many of the implementations of filters used in the filtering process were provided by this library.
http://opencv.org/

Slick-Util
Part of the larger Slick2D game library. Used for showing text in OpenGL.
http://slick.ninjacave.com/slick-util/

Appendix A

User Guides

A.1 Running Java programs

Many of the tools that were made as part of this project were written in Java. All of them require an installation of java 7 or higher in order to run. The tools themselves are packaged inside "JAR" files.

On windows, java installers automatically associate themselves with the .jar filetype. So running should only involve opening the file by double-clicking on it. The same counts for Mac OSX.

If another program has associated itself with JAR files, or the program requires command line parameters (as stated in the guide), it might be easier to run the file from the command line. The command for this is:

```
java -jar [filename].jar ["parameter 1"] ["parameter 2"]
```

Executed from imside the directory of the JAR file. Note that it may be needed to add the Java Runtime Environment to the system PATH variable if the installation has not done so itself.

A.2 STL file viewer

The STL file viewer is a java application. See A.1 for instructions on how to run the program. It does take an *optional* parameter in the form of a file path that the viewer should open. This can either be a relative path (to the execution folder), or an absolute path. An example relative path would be:

```
java -jar STLViewer.jar "models/robot.stl"
```

And a (linux) absolute path:

Figure A.1: The controls for the STL viewer and Unprojection viewer.

```
java -jar STLViewer.jar "/usr/home/administrator/models/robot.stl"
```

The camera of the viewer can either be controlled through an XBOX® 360 controller or a keyboard. The button layout for the controller is shown in figure A.1. For the keyboard controls, see table A.1.

Key	**Action**
W	Move forward
A	Move left
S	Move backward
D	Move right
Q	Move down
E	Move up
Arrow keys	Rotate camera (look around)

Table A.1: Keyboard bindings for the STL viewer and unprojection viewer

A.3 Unprojection viewer

The unprojection viewer is a java program, so refer section A.1 on how to run the tool. Unlike the STL Viewer, the unprojection viewer has no (optional) command line parameters. However, the camera controls are exactly the same as the STL viewer.

A.4 X-Ray simulator

The X-Ray simulator is a java program. Section A.1 explains how to run the program. The main window of the X-Ray simulator is shown in figure A.2. This window is used to

set up the simulation.

The window is divided vertically into two sections. The top half is used to set up the individual geometry sources and the bottom half configures amongst others the setup of the imaging. The simulator does not allow simulation of an empty scene. The button that starts the simulation is therefore initially disabled when the program is started. Geometry sources can be added by clicking the "Add.." button. A file selection window will pop up. Only STL files are supported by the simulator.

Once an STL file has been selected, it will appear in the geometry sources list. To remove a file from this list, select it and click the "Remove selected" button. The simulator requires that the density of each model is known. This can be specified for a single geometry source by selecting it in the sources list, then specifying some density in the input box to its right. Note that these boxes specify the relative density. For example, two geometry sources with relative densities will imply that the second source is twice as dense as the first, respectively.

When the geometry sources have been added and configured, the X-Ray scanning setup has to be specified. On the lower half of the screen are input boxes for the 3D coordinate of the emitter, as well as the X positions of the two detectors.

Finally, the resolution of the image has to be specified. A recommended value is 1000 pixels. Note that larger resolutions will quickly lead to large memory usage. Also note that the image height is calculated automatically from the size of the geometry. The geometry is also scaled so that it always completely fills the image.

The fish rotation is optional, but can be used to rotate the fish around the Z-axis.

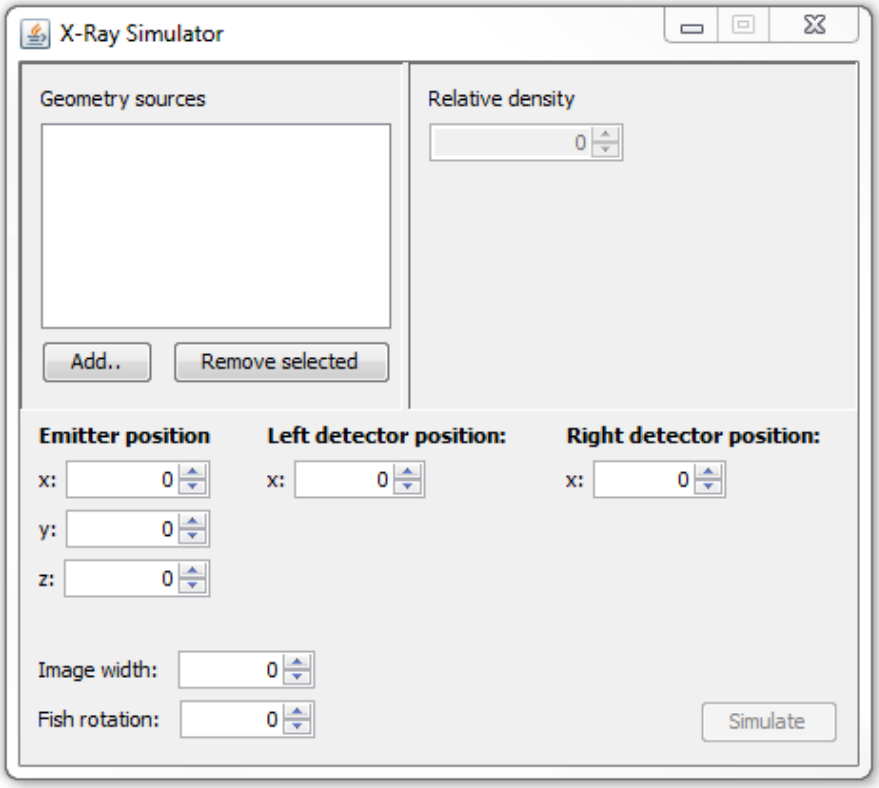


Figure A.2: The main window of the X-Ray simulator.

A.5 Point projection calculator

The point projection calculator is a java program. See section A.1 for running instructions. The calculator (shown in figure A.3) is a small tool to calculate where a coordinate inside a fish mesh will appear on a simulated X-Ray image, given a scene setup. After entering an emitter position and two detector positions, entering any coordinate and clicking the "Calculate" button, the projected coordinates will be shown on the right hand side of the window.

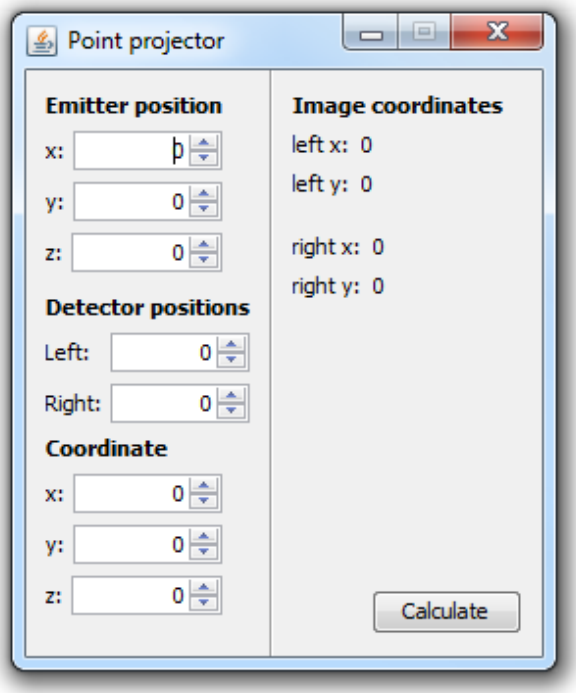


Figure A.3: The point projection calculator window.

A.6 Bone processor

The bone processor is the only program that is not a java program. Note that it has only been tested on, and compiled for windows.

Its build comes in two flavours: an image filter only (Bonefilter.exe), and an unprojector (Unprojector.exe). Bonefilter is used to process a single image, and Unprojector can be used to process, then unproject two stereoscopic images into a point cloud.

Both rely on a configuration file for a number of settings. This file is located in the same directory as the executables, called "config.cfg". The settings in this file are:

emitter.x, emitter.y, emitter.z Emitter position

fish-origin.x, fish-origin.y, fish-origin.z Origin of the fish relative to the origin of the scene

detector1.x X coordinate of detector 1

detector2.x X coordinate of detector 2

image.width, image.height The width and height of the input image(s) (used for generating the output images)

bone_reconstruction.slope_used_coordinate_count Number of points used for calculating the slope of a line segment during bone reconstruction

bone_reconstruction.max_travel_x, bone_reconstruction.max_travel_y Maximum number of pixels another line segment can be separated from a bone

bone_reconstruction.min_bone_size The minimum required number of pixels a bone must have. It is considered noise and removed if it does not meet this minumum.

Only the settings with prefix "bone_reconstruction" are required for running BoneFilter.exe. All other settings are needed when performing an unprojection.

When the desired settings have been configured, the two tools are started by the following commands:

```
Bonefilter.exe "path/to/image.png" "image_name" [-verbose]
```

The first parameter specifies what image should be processed. The second specifies the name of the image. This name is used in the file name of step images (images that are saved after every processing step). Finally, an optional parameter can be added to make the program write those step images, as well as showing more progress information in the console window. Note that this program has no output at all when verbose is turned off. It will, however, give an indication of the time used to process the image.

The unprojection tool is used as follows:

```
Unprojector.exe "path/to/left/image.png" "path/to/right/image.png" [-verbose]
```

As an unprojection requires a stereoscopic pair, this tool takes in two image files as parameters. The verbose parameter is again optional, and has a similar effect as the bone processing tool.

Bibliography

[1] "Valka X-Ray Guided Cutting Machine promotional materials, © Valka EHF," 2014. Images used with permission from Valka EHF.

[2] L. D. Harris, E. L. Ritman, and R. A. Robb, "Computer generated 3-d display of x-ray computed tomographic volume image data," in *1st European Conference on Cineradiography with Photons or Particles*, pp. 176–182, International Society for Optics and Photonics, 1983.

[3] J. Evans and M. Robinson, "Design of a stereoscopic x-ray imaging system using a single x-ray source," *NDT & E International*, vol. 33, no. 5, pp. 325–332, 2000.

[4] J. Evans and H. Hon, "Dynamic stereoscopic x-ray imaging," *NDT & E International*, vol. 35, no. 5, pp. 337–345, 2002.

[5] J. P. O. Evans, Y. Liu, J. Chan, and D. Downes, "View synthesis for depth from motion 3d x-ray imaging," *Pattern recognition letters*, vol. 27, no. 15, pp. 1863–1873, 2006.

[6] P. Gravel, G. Beaudoin, and J. A. De Guise, "A method for modeling noise in medical images," *Medical Imaging, IEEE Transactions on*, vol. 23, no. 10, pp. 1221–1232, 2004.

[7] C. Ericson, *Real-time collision detection*, vol. 14. Elsevier Amsterdam/Boston, 2005. Triangle-Line intersection algorithm from section 5.3.4 used in simulator implementation.

[8] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, no. 1, pp. 16–22, 2000.

[9] D. Oram, "Rectification for any epipolar geometry.," in *BMVC*, vol. 1, pp. 653–662, 2001.

[10] S. Roy, J. Meunier, and I. J. Cox, "Cylindrical rectification to minimize epipolar distortion," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 393–399, IEEE, 1997.

[11] M. Pollefeys, R. Koch, and L. Van Gool, "A simple and efficient rectification method for general motion," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, pp. 496–501, IEEE, 1999.

[12] "A threshold selection method from gray-level histograms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 9, pp. 62–66, Jan 1979.