Freie Universität Berlin
Institut für Mathematik
Arnimallee 6
14195 Berlin

Visibility-Driven Depth Determination of Surface Patches in Direct Volume Rendering

Master-Thesis

to reach the degree of Master of Science

Presented by: Sergej Stoppel

Matriculation number: 42 93 979

Born: 12.02.1987 in: Tomsk

Investigators:

Prof. Dr. Polthier
Freie Universität Berlin
Institut für Mathematik
Arnimallee 6
14195 Berlin

Prof. Dr. Wiebel
Grafische Datenverarbeitung
Fakultät Elektrotechnik
und Informatik
Hochschule Coburg
Friedrich-Streib-Str. 2
D-96450 Coburg

Berlin, October 1, 2013

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Berlin, October 1, 2013

This master thesis provides an algorithm *surfseek* for selecting surfaces on the most visible features in direct volume rendering (DVR). The algorithm is based on the algorithm "what you see is what you pick" (WYSIWYP), which was presented by Wiebel in [WVFH].
The algorithm *surfseek* projects a surface patch on the DVR image, consisting of multiple rays. For each ray the algorithm executes WYSIWYP or an adjusted picking criterion. In the next step the algorithm constructs an graph and computes a minimum cut on this graph. The minimum cut represents the minimum weighted surface. In the last step the selected surface is displayed.

Contents

1 Direct Volume Rendering and Visibility

1.1 Introduction

Direct volume rendering (DVR) is the state-of-the-art method for displaying volumetric data in medicine, engineering, physics and other natural sciences. As a flexible tool it provides a diversity of applications and solutions to numerous of problems concerning 3D scalar fields. For more details see the paper from P. Sabella [PS].
The goal of this thesis will be to provide a technique that allows the user to select the surface of some features in the DVR, that are the most visible for the user. Motivated by this idea, this thesis will be build up on previous works by Wiebel et al. [WVFH] and [WPFV]. These works present two intuitive algorithms for selecting volume features and drawing lines on the surface of a feature in the DVR, the WYSIWYP algorithm (What You See Is What You Pick) and VisiTrace respectively.

WYSIWYP computes a 3D point on a boundary of a feature in DVR from the mouse coordinates, after the user clicked on a certain position on the screen, where the user perceives the feature.
VisiTrace, which uses WYSIWYP, allows the user to draw strokes on the volume boundary.
Since the algorithm in this thesis will differ greatly from VisiTrace, we explain only WYSIWYP in a brief synopsis.

The approach in this thesis is a graph theoretical one. For the surface computation we will construct a non-trivial graph, where the wanted surface will be encoded as a minimal cut of the graph. The minimal cut will be computed using the Boykov-Kolmogorov algorithm, see [BK]. The needed mathematical terminology will be introduced in a later section.

In the following the reader will recognise that this work is divided in four main sections:
Part One: Introduction to DVR including the terminology, a discussion of what is visible and an introduction to WYSIWYP.
Part Two: Presentation of the algorithm *surfseek* for detection of surface patches in DVR.
Part Three: Analysis of the algorithm *surfseek*, introduction of noise and tests of *surfseek* on noisy data-sets.
Part Four: Discussion of limitations of the algorithm, a final conclusion and outlook.

Since some terms are not common in everyday use, or may differ from the intuitive meaning, the reader is invited to familiarize himself with the terminology in the appendix.

1.2 Direct Volume Rendering: an Overview

1.2.1 Direct Volume Rendering and other models

Direct Volume Rendering is a common technique to display three dimensional data structures. In most cases the data is given as a scalar value field, where each point in the data has a position and a scalar value (intensity), for example density. The data often comes from a CT-Scan or an MRT-Scan. Those scans measure in slices, which result in the 3D data once they are combined. In order to visualize the data one can move from slice to slice and reconstruct the volumetric image in the mind. However this is not a technique to perceive the volumetric image, as it does not represent the image as whole. One could try to show multiple slices but still this would never show all aspects of the data, as the reader can see in Figure 1 and 2. The reader can clearly see, that it is very difficult to understand the volume as whole by moving the slices.
Another technique to visualise a feature is to compute an isosurface, that is done by combining all points with the same threshold-value to a surface. The computational process is done using the marching cubes algorithm by Lorensen and Cline [LC] or some improved version like [N] by Nielson. The threshold-value must be de defined by the user beforehand. The hereby resulting images can give a good impression of the underlying feature, but the user is not able to see what is behind the isosurface. Furthermore it can be the case, that the interesting parts of the feature have different values than the chosen threshold value or the intensity of the desired feature can be unknown. As an illustration the reader can look at Figure 3 and 4. For these two images two different threshold-values were used, $Tv_1 = 312$ and $Tv_2 = 1200$. By just looking at the first picture the user can only guess how it looks inside the body.
In order to show the whole data, a transfer-function is applied on the data-set. The transfer-function is very crucial to DVR as it does influence the resulting image greatly. It maps the intensity to three color values (red,green and blue) and opacity alpha. Afterwards the most common and natural implementation is to cast rays along the viewing direction through the volume and accumulate the color information for the volumetric values along the ray. The density of the rays and the samples along the rays can be chosen by the user to cover the data sufficiently. Furthermore the user can decide between parallel projection or perspective projection, see Figure 5 and Figure 6.
The accumulation of the color information is accomplished through an emission-absorption-model, this is formalized through the *volume rendering integral*

$$I(r_{max}) = I_0 e^{\int_{r_0}^{r_{max}} \tau(t)dt} + \int_{r_0}^{r_{max}} Q(s) e^{\int_{s}^{r_{max}} \tau(t)dt} ds. \quad (1)$$

See Wiebel et al. [WVFH] for more information. This equation describes the accumulated intensity I in one color along the ray on the interval $[r_0, r_{max}]$, with r_{max} being at the camera position and r_0 at the back end of the volume. I_0 is the light intensity at the back of the volume. Furthermore, s and t are parameters in this interval, and $\tau(t)$ the

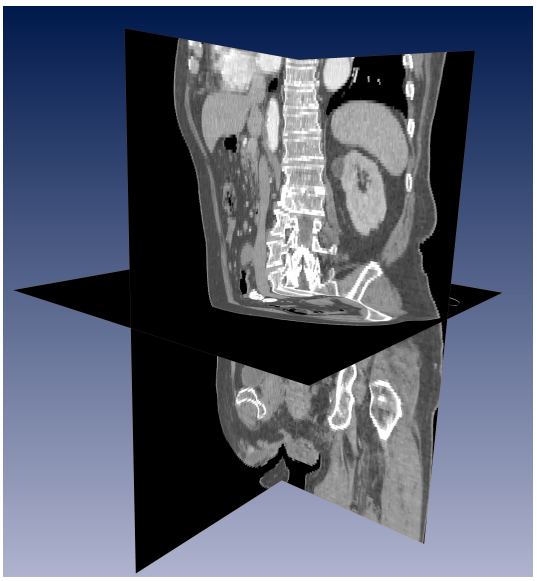

Figure 1: Multiple Slices of a Male Body CT-scan.

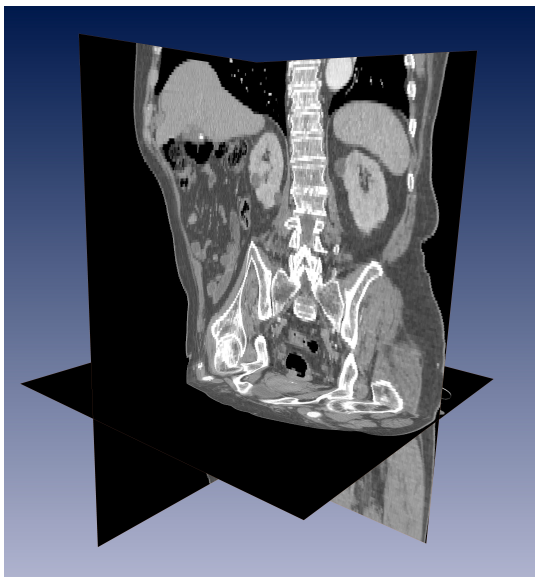

Figure 2: The Same CT-scan as in Figure 1, with different Slice Position. The insight of the body was not really changed.

opacity for a particular position on the ray described by t. Q is describes the light emission for a certain sample, described by s. Now the first part of the equation describes the remaining intensity from the background that reaches the viewer, the second part of the equation considers both the emission Q and as well the absorption along the ray. Adding those two up, we receive the overall intensity for one color $I(r_{max})$.
By looking at equation 1 we can see that the image information is derived of the values given by the transfer-function. Thus by slightly changing the transfer-function the user can receive totally different images with incredibly different insights. See Figure 7 and 8.

1.2.2 Discretized ray casting

So far for the analytical approach, in order to compute the integral for the samples, it has to be discretized. The iterative computation of the discretized version in a *front-to-back* manner can be formalized through:

$$c_{n+1}^{acc} = c_n^{acc} + (1 - \alpha_n^{acc})c_n^{tf} \tag{2}$$
$$\alpha_{n+1}^{acc} = \alpha_n^{acc} + (1 - \alpha_n^{acc})\alpha_n^{tf}. \tag{3}$$

Here acc denotes the accumulated values, tf denotes the values given by the transfer function, c is the color value, α is the opacity and n indicates the index of the sample on the ray. See [WVFH] for more details.
Note that the opacity naturally lies in $[0, 1]$, with 0 being totally transparent, and 1 being totally impenetrable. From this and equation 3 we can derive a few informations

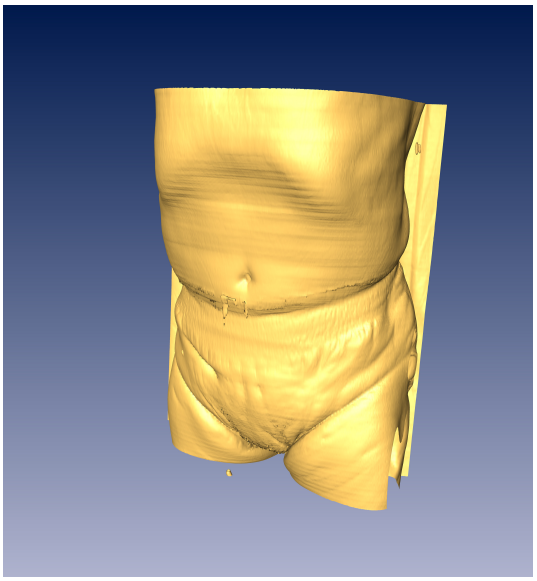

Figure 3: Isosurface with a threshold-value of 321.

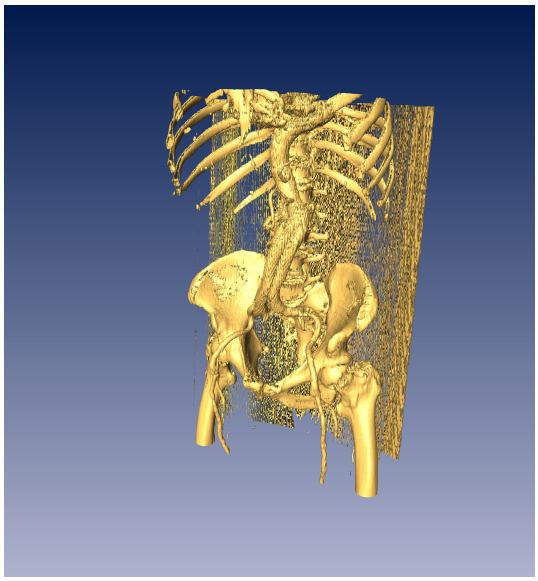

Figure 4: Isosurface with a threshold-value of 1200.

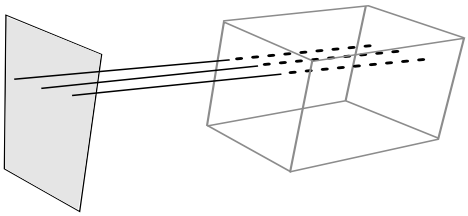

Figure 5: Parallel Ray Cating

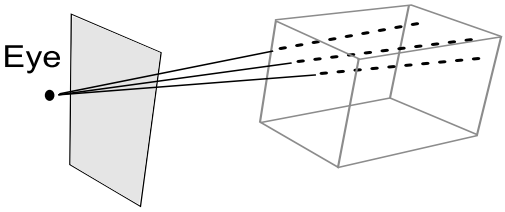

Figure 6: Perspective Ray Cating

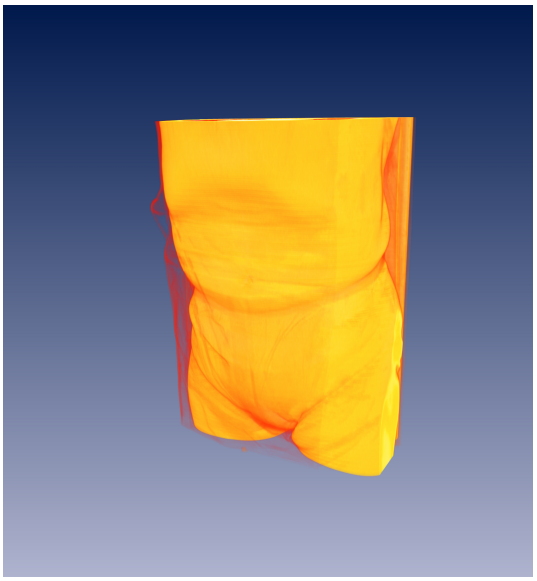

Figure 7: DVR Image with a transfer-function with support around low densities.

Figure 8: DVR Image with a transfer-function with support around high densities.

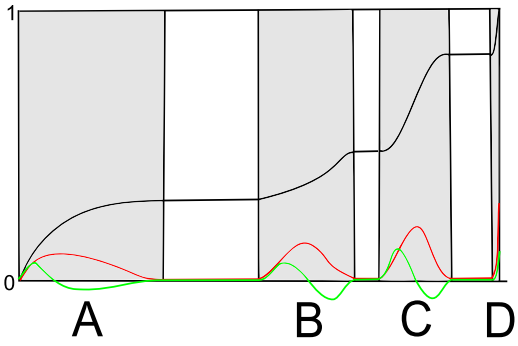

Figure 9: The black curve represents α^{acc}, the red curve β^{acc} and the green curve γ^{acc}. Here the interval C has the largest increase of opacity, where D contains the steepest increase. Note, that those curves are only illustrations and hence do not yield full correctness.

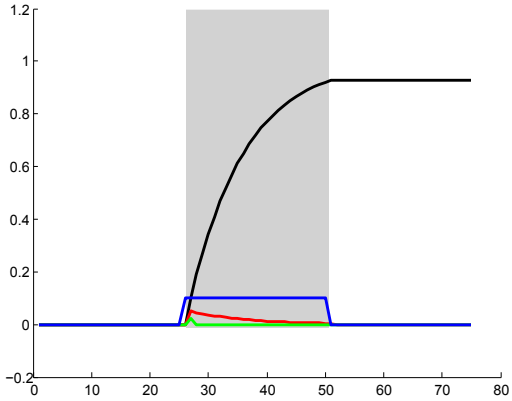

Figure 10: An example of calculated values for a feature with a constant opacity of 0.1, indicated by the blue curve. The black curve represents α^{acc}, the red curve β^{acc} and the green curve γ^{acc}. The gray area represents the registered feature.

about α^{acc}.
First: α_k^{acc} lies in $[0, 1]$ for all k; and second: α_k^{acc} is monotonically increasing.
Those two observations can be easily proven by induction.
It is worth mentioning, that the increase of the opacity is strongly dependent on the density of samples. This could lead to an opacity rescale if non equidistant samples are used. For simplicity we will restrict to equidistant cases only.

1.2.3 WYSIWYP

Now that we are familiar with Direct Volume Rendering, it is time to introduce the picking algorithm WYSIWYP. The first thing to say about this picking algorithm is, that it is a mainly visibility oriented picking algorithm. The procedure in the picking is as easy as it can be, the user clicks on a position on the screen to pick a feature, most visible to him or her. The algorithm transforms the screen coordinates to world coordinates and casts a ray through the data scene. Along this ray the the information is gathered according to the formulas 2 and 3. Finally a certain criterion is used to determine the position that is to pick in world coordinates.
Obviously it is the last step, that is crucial, and therefore is the heart of WYSIWYG.
In order to understand the picking criterion we need to establish a few notions. We denote the first derivative of α^{acc} with β^{acc} and the second derivative of α^{acc} with γ^{acc}.
We will now focus on the structure of α^{acc}, β^{acc} and γ^{acc}, and we will extend the observations to a hypothesis of what is perceived as most visible by the user.
The opacity is, to some extend, a natural indicator of what is perceived as visible. We

will not perceive something if it has no opacity at all, and we will not see any structures behind an object with a full opacity of 1. The amount of opacity contribution of a spatial feature determines the influence on the final color of the pixel and thus indicates which feature is perceived. In fact the user perceives those features with the highest accumulation of opacity as the most visible. This observation has been confirmed in experiments in the mentioned work by Wiebel at al. [WVFH].
Furthermore the experiments indicate that the perception does not depend on the steepness of the opacity increase, or to say on the magnitude of the β^{acc} values, but on how much the opacity increases over the consecutive samples in the feature, i.e. on the magnitude of the contribution to the final pixel.
But when does a feature begin and when does it end? The feature stars when the opacity starts to accumulate and it ends when the accumulation stops. This is illustrated in Figure 9 and 10. In principle the interval boundaries are defined through positions where γ^{acc} crosses the zero values. A crossing from below indicates the startposition of the interval, a crossing from above indicates the end of the interval. However this is only true if the opacity α^{acc} is strictly increasing. As illustrated in Figure 9 α^{acc} can have plateaus with zero increase. Thus the criterion for the boundaries adjusts to: an interval starts if γ^{acc} becomes positive, after being negative or zero, and an interval ends when γ^{acc} attains positive or zero values after being negative. We will call this boundary criterion the γ-criterion from now on.
After detecting all boundaries, the algorithm finds the feature with the greatest opacity increase and sets the picked position on its boundary.

1.2.4 WYSIWYP Limitations

WYSIWYP Relaxation
As Philipp Johannes Preis already mentioned in his diploma thesis [PJP], foggy datasets can lead to a problem regarding the definition of when an interval ends when using WYSIWYP. If the data is foggy, then every sample on a ray has a contribution to the resulting pixel, and hence the contribution to the pixel is decreasing with greater distance, γ^{acc} can not reach 0. In this case the algorithm will not detect the end of the feature (i.e. $\gamma^{acc} \geq 0$) and will set the end of the dataset as the backside of the feature.

Preis suggested, a solution to this problem is to relax the criterion for the end-boundary. This means instead of defining the boundary at $\gamma^{acc} \geq 0$ we define the boundary at $\gamma^{acc} \geq -r$ for some $r > 0$ with $r{:=}j \cdot c$. Philipp Johannes Preis suggested to define j and c as: $j{:=}\max_s\left(\beta^{acc}(i_s)\right)$ and $c{:=}1$.
For further information I refer to Preis's diplomathesis [PJP].

Different feature detection criterion
Although the idea to select the most visible feature is very good, the γ-criterion for detection of the feature boundaries is too strong. While the algorithm works perfectly

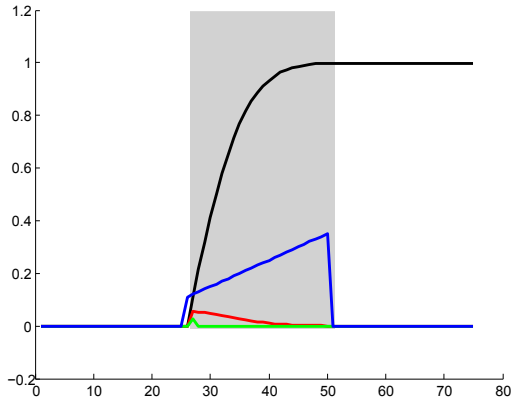

Figure 11: An example of calculated values for a feature with a increasing opacity of 0.1, indicated by the blue curve. The black curve represents α^{acc}, the red curve β^{acc} and the green curve γ^{acc}. The gray area represents the registered feature.

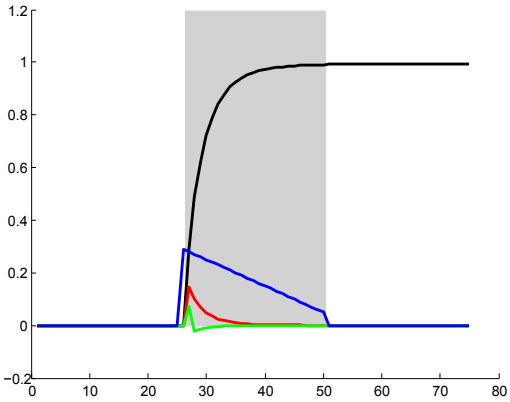

Figure 12: An example of calculated values for a feature with a decreasing opacity of 0.1, indicated by the blue curve. If the opacity has a constant gradient, then WYSIWYP detects the feature boarders correctly.

for features with constant or linear changing accumulation, see Figures 11 and 12, it often breaks up the features in parts when dealing with experimental data.
If the Accumulated opacity does not increase uniformly, but underlies some fluctuation, the algorithm will detect multiple features inside one single feature. A good example of this behaviour is ray cast through a CT-scan of a human body, Figure 13 and 14. When we look at the accumulated opacity values, it is very easy to determine the boundaries of the features the ray passes through, Figure 13. But if we use the WYSIWYP criterion for feature detection, the algorithm finds several features where there is one, see Figure 14. This is due to the small fluctuation of the accumulated opacity.
The same behaviour can be observed when dealing with synthetic data-sets. For instance, when we use WYSIWYP on a sphere with density values given by: $d = (1 - R) \cdot 100$, the algorithm detects multiple features again. See Figure 17 and 18. This unexpected behaviour is caused through the discretization of the transfer function over the dataset, which causes a small change of the incline, as one can see in Figure 18. In this case the values of the second derivative β are in the range of $[0, 2 \cdot 10^{-3}]$. Considering that β lives in a such small range, it is not that surprising any more, that even small discretezation errors can cause dramatic miscalculations.
But there is a solution to this problem that also takes up the idea behind WYSIWYP. Instead of looking at the γ-function we will concentrate on the β-function instead. The general idea is very simple. We say that a feature begins when the accumulated opacity starts to grow, and a feature ends when the accumulation growths stops. Since we consider the β-function, we will call this picking criterion the β-criterion. Considering the

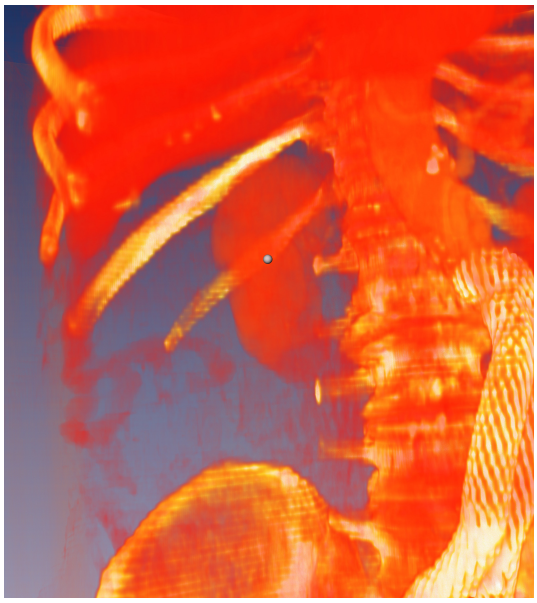

Figure 13: The ray is casted through the upper abdominal area, through a kidney and an underlying rib.

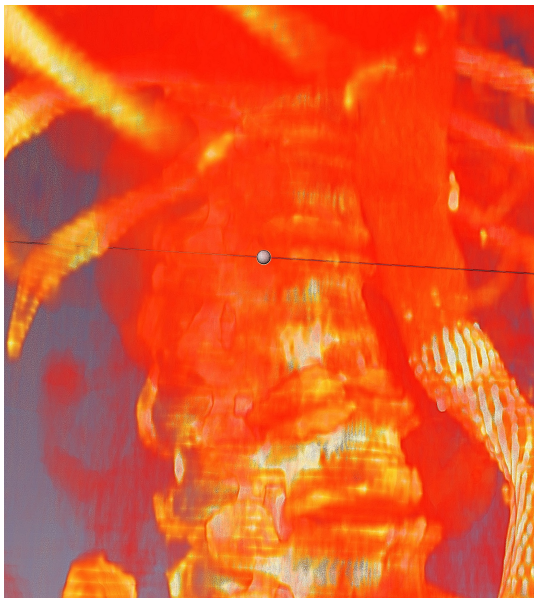

Figure 14: The same ray cast as in Figure 13 from the side view.

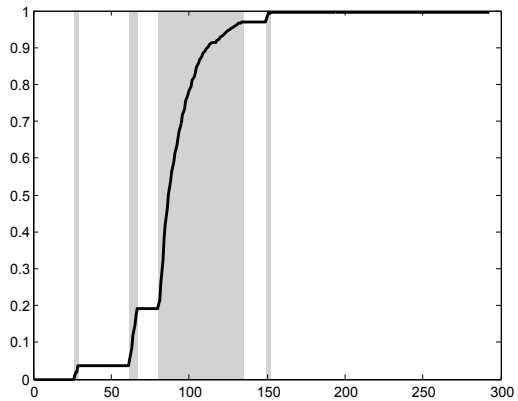

Figure 15: It is easy to find the boarders of the features while looking at the accumulated opacity function. The intuitive perceived features are highlighted in grey.

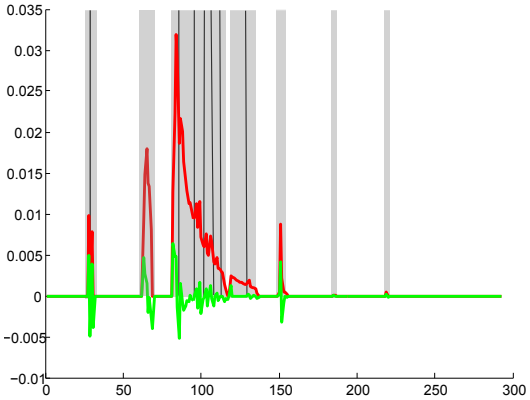

Figure 16: The first and second derivative of the accumulated opacity shown in Figure 13. Using the criterion for feature detection as suggested in WYSIWYP yields in divided features, grey areas separated by the black lines.

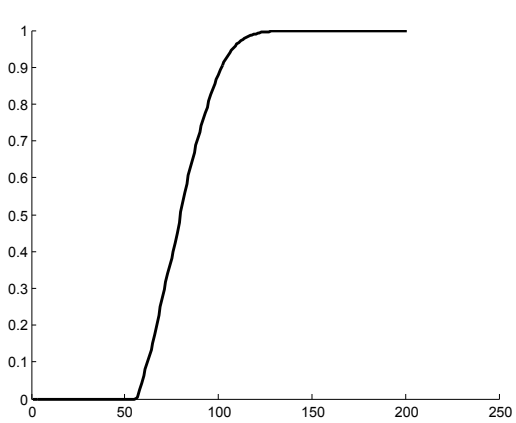

Figure 17: Accumulated opacity from a sphere, intuitively one assumes one feature.

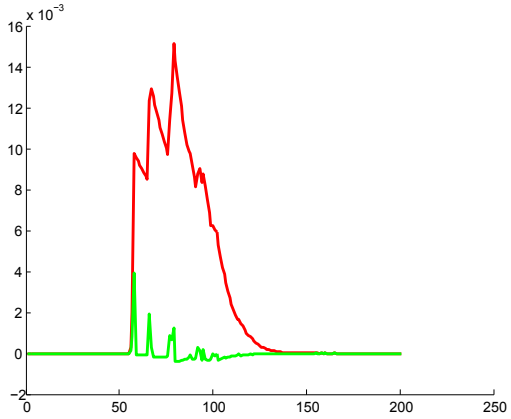

Figure 18: When using the γ-criterion from WYSIWYP on the accumulated opacity in Figure 17 the algorithm detects several features. This leads to a selection of a surface inside the volume.

discretized accumulation function it holds for the k-th step:

$$\text{sample } k \text{ is inside a feature if:} \qquad \beta_k > 0 \tag{4}$$

$$\text{sample } k \text{ is outside a feature if:} \qquad \beta_k = 0. \tag{5}$$

This criterion for a feature detection is not good enough when we deal with noisy or foggy datasets. When a feature is detected as soon as $\beta_k > 0$, then in case of foggy datasets the algorithm will detect a feature everywhere. Thus we need a lower limit ϵ such that we say:

$$\text{sample } k \text{ is inside a feature if:} \qquad \beta_k > \epsilon \tag{6}$$

$$\text{sample } k \text{ is outside a feature if:} \qquad \beta_k \leq \epsilon. \tag{7}$$

How do we choose such a limit? A constant limit ϵ is not a good choice, because the magnitude of the derivative of the α-function can vary greatly when dealing with different data-sets. For example, the range of β in Figure 14 is $[0, 0.033]$ in Figure 18 we see a range of $[0, 0.016]$, which is only half as large. But we could still detect all features correctly, when using $\epsilon = 0.005$. But such a fixed limit could cause unwanted results, if a feature has a high β-function but is wrapped in a fog with β-values greater than 0.005. Or we could deal with a feature that has low β-values, but is the only feature that is in the dataset. Such a feature would not be detected at all.
This calls for a dynamic limit. With x and y we will denote the positions of the rays and with j we denote the index of the j-th sample on each ray. During the work I found, that a limit defined as:

$$\epsilon = \max_{x,y,j} \beta_{x,y,j} \cdot 0.15, \tag{8}$$

Figure 19: A projection of a ribbon on a kidney using β limit criterion as feature detection and *surfseek* algorithm to find the minimum weighted surface.

Figure 20: The same surface as in 19 but with a different transfer-function for better insight on the shape of the kidney. Notice how the ribbon conforms the surface.

works good for the most data-sets. See Figure 19 and 20. We will see later that in some cases this limit is too low, and thus is not final.
That the choice of the last factor in equation 8 can be very crucial can be observed in Figure 21 and in Figure 22. In the both Figures the surface was computed using the β-criterion, in Figure 21 we used an ϵ factor of 0.2. In this case the more visible rear feature is selected. In Figure 22 the ϵ factor was chosen as 0.05, thus the algorithm does not detect a boundary between the two features and selects the less visible feature in front.
It is also possible to choose an $\epsilon_{x,y}$ separately for each single ray. This is more a question of taste, than correctness. Suppose we want to select some bone tissue which is embedded in some tissue material with a some minor β-values. Choosing a global ϵ would cause, that the minor tissue is ignored if it is smaller than $\epsilon = max_{x,y,k}\{\beta_{x,y,k}\} \cdot 0.15 = \max_{x,y,k}\{\beta_{x,y,k}|$ sample $j_{x,y,k}$ is inside bone$\}$. This can easily be the case when the bone tissue has a fast opacity accumulation.
Using different ϵ for each ray would cause that the rays that don't run through the bone tissue would recognize the tissue with minor β-values as a feature. This could lead to jumps when the feature with high β-values ends. This behaviour is illustrated in Figures 23 and 24. In my opinion it is more natural to select a single feature and neglect the surrounding properties, than ending up with a jaggy surface. Of course this means we accept some information loss in order to get smoother surfaces. Keeping this limitation in mind we will use a global ϵ from now on. At the end of this work we will discuss cases where the choice of a global ϵ leads to unwanted surfaces, but in most cases a global ϵ

Figure 21: A surface computed with *surf-seek* using the β-criterion with an ϵ factor of 0.2.

Figure 22: A surface computed with *surf-seek* using the β-criterion with an ϵ factor of 0.05.

works just fine.

1.2.5 Beta-criterion vs gamma-criterion

Sadly the β-criterion is not the perfect solution for everything that can go wrong. While it is not vulnerable against noisy data, it is simultaneously too rigid in cases when the features overlap. When two features overlap, the β-criterion has two possible outcomes:
1 : The two features will be detected as one.
2 : The feature with the smaller β-will not be detected.
It is important to mention, that we assume the β-values not to fluctuate greatly, that is not to jump around ϵ. If those jumps occur, then there are of course more than those two outcomes possible.
That means, using the β-criterion leads to a lost of information as soon as two features overlap. Using the γ-criterion those features will be detected separately.
We will now study the two detection criteria on a synthetic example. Let us consider a dataset where we have a ball with relatively high opacity values inside a bigger ball with lower opacity values, such an example is illustrated in Figure 25. The measured opacity values are shown in Figure 26.
In this example the algorithm should detect the surface of the inner ball and select the inner ball as it contributes most to the opacity. The magnitude of the contribution can be observed in Figure 27, the reader should clearly see the boundaries of the inner ball at samples 75 and 175.
When the detection is made with the β-criterion the only chance of selecting the inner

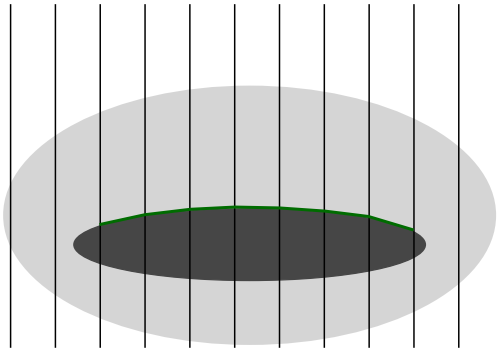

Figure 23: Using a global ϵ would cause to neglect the foggy or less visible area, thus the algorithm will only select the dark feature.

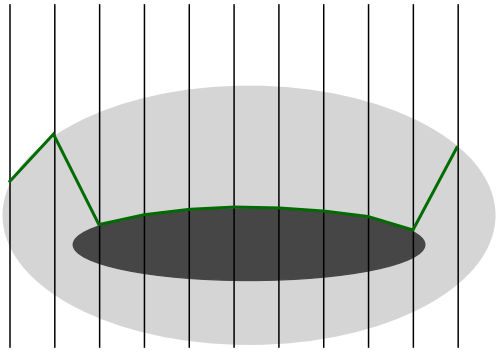

Figure 24: Using a local ϵ will lead to a new feature detection criterion in every ray. This results in a jaggy surface, which combines the dark feature and the outer regions of the less visible feature.

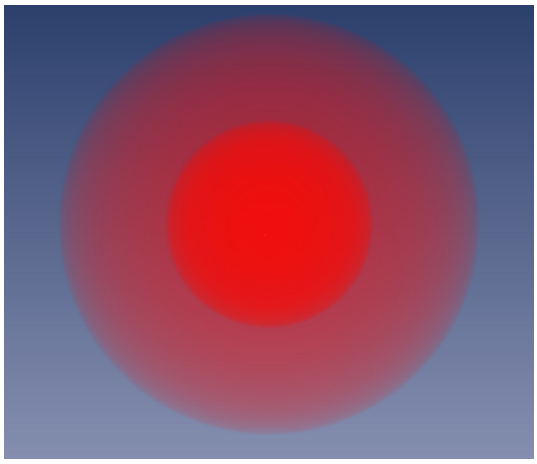

Figure 25: An Image from a ball with relatively high opacity in a bigger ball with lower opacity.

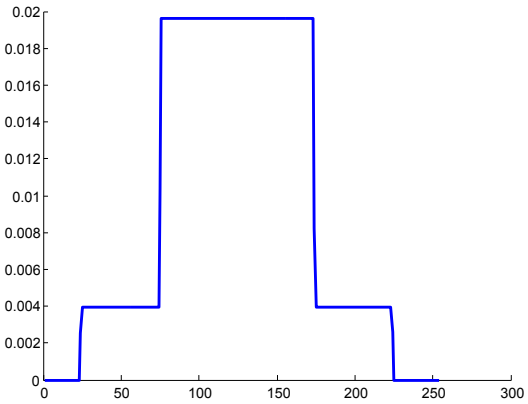

Figure 26: The opacity values along a ray that runs directly through the middle of the two balls in Figure 25.

ball is to choose $\epsilon > \max_{x,y,j} \beta_{x,y,j} \cdot 0.25$, but in this case the outer ball will be completely ignored. Which is a lost of information as mentioned before.
Using the γ-criterion, the boundaries of both balls can be detected, since the γ-function performs the characteristic zero crossing, see the second derivative in Figure 28. After evaluating the opacity contributions the inner ball would be selected.
Let us consider the same dataset with a small perturbation on the balls. The perturbation is so small, that the resulting DVR image seems to be the same as in Figure 25. Looking at Figure 30 the reader can see that the perturbation is in fact very small, the perturbation lies within a range of $[0, 0.0001]$.
As before the β-criterion can detect the inner ball if we choose ϵ big enough, for example $\epsilon = \max_{x,y,j} \beta_{x,y,j} \cdot 0.3$. But then we ignore the outer ball completely again.

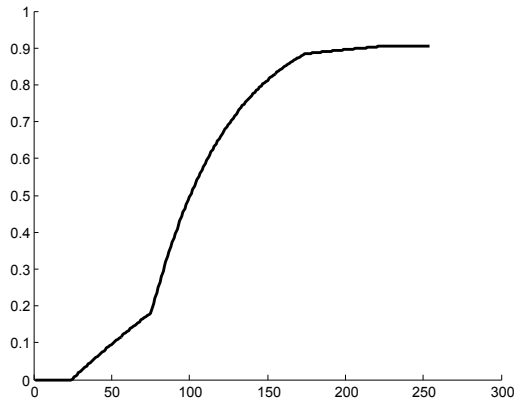

Figure 27: The accumulated opacity corresponding to the ray-cast in Figure 26.

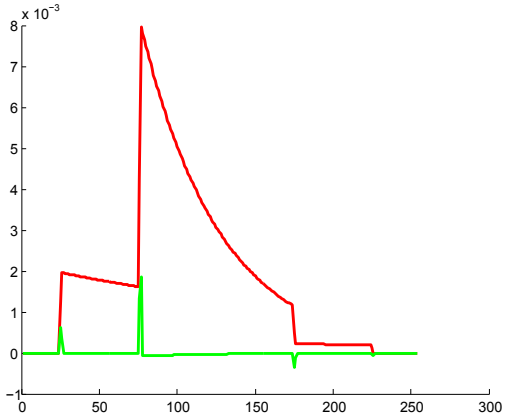

Figure 28: The first (red) and second (green) derivative of the accumulated opacity shown in Figure 27.

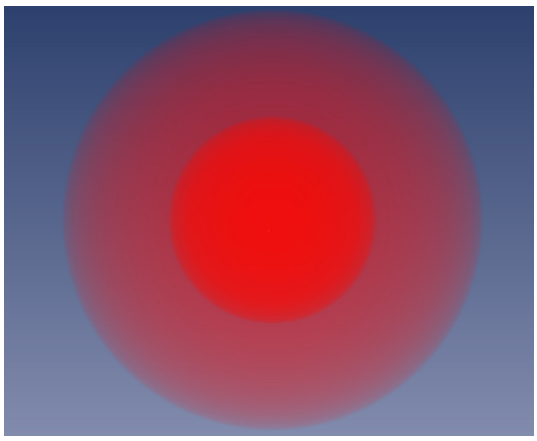

Figure 29: The same dataset as in Figure 25 with small random values added to the data.

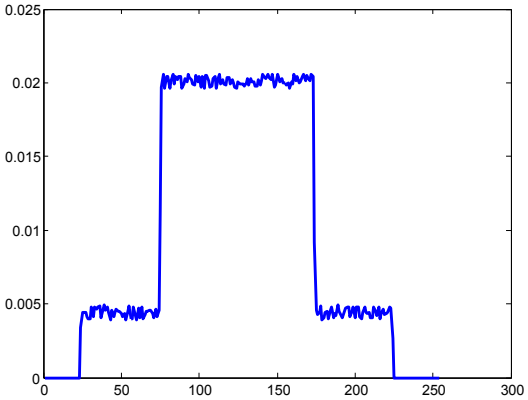

Figure 30: The opacity values along a ray that runs directly through the middle of the two balls in Figure 29.

What happens when we use the γ-criterion? When the boundaries are detected with the γ-criterion, then the algorithm will detect boundaries in a not predictable manner. See the sign change of the γ-function in Figure 32. Both of the balls are separated in several features when we choose the γ-criterion. This is qualitatively equivalent to a total neglect of the outer ball, since the each contribution of the divided features is very small. Furthermore the inner ball is divided up too, in extreme cases this could lead to selections inside the feature.

In the two previous examples a feature with low opacity was in front of a feature with high values. What happens when we have a feature with relatively high opacity in front of a feature with a low opacity?

In this case the γ-criterion can not detect the border, because the first derivative of the accumulated opacity will still be decreasing, thus the γ-function stay negative at the border. See Figure 34.

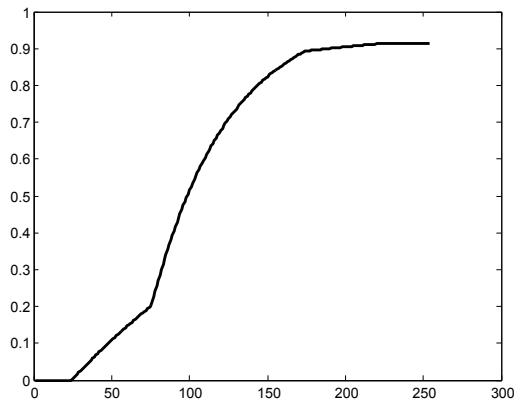

Figure 31: The accumulated opacity corresponding to the ray-cast in Figure 26.

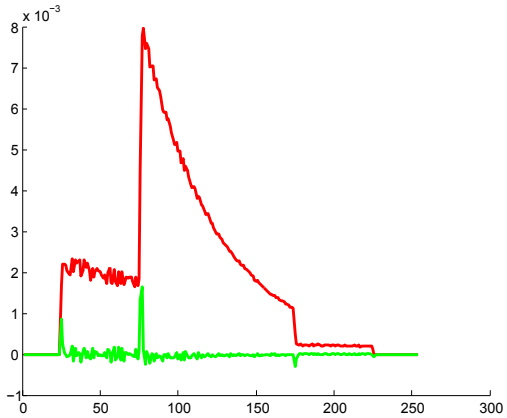

Figure 32: The first (red) and second (green) derivative of the accumulated opacity shown in Figure 27. Notice that the γ-function crosses zero randomly.

However the β-criterion has no chance to detect the boundary either. Since the β-function decreases very fast at the end of the feature.
In summary we see that the β-criterion is robust considering perturbations in the data, but it has a weakness when features overlap or when their boundaries meet. The γ-criterion can detect the boundaries of overlapping features correctly if the opacity is smooth enough. On the other hand it is vulnerable to noise, as it will split up the features.
Both of the detection criteria, can not detect a boundary behind a feature with high opacity.
Unfortunately even though we can adjust ϵ in a way that the algorithm only neglects features if the accumulated opacity is already saturated. It can still lead to unwanted pickings. This can be observed in Figures 35 and 36.
In Figure 36 an $\epsilon = \max_{x,y,j} \beta_{x,y,j} \cdot 0.3$ would detect both features and the algorithm would adjust the front feature a significantly bigger weight, as it is wanted.
In the Figure 35 the feature with the bigger opacity has a total opacity contribution of 0.5124 the feature with the lower opacity has a total opacity contribution of 0.469. The two features can be detected separately with an $\epsilon = \max_{x,y,j} \beta_{x,y,j} \cdot 0.6$, but because the β-function drops rapidly but stays positive for a long time the opacity contribution is detected wrong. The algorithm computes a total opacity contribution of 0.368 for the first feature and a total opacity contribution of 0.207 for the second feature, counting from the left. This form of distortion always occurs when we use the β-criterion for surface detection, since it detects the start of a feature after the opacity accumulation begun and it detects the end of a feature before the opacity accumulation stopped. The distortion of the start position is usually very small. However the detection of the end-position of a feature can be very displaced, because the β-function gets smaller but also flatter the longer the feature is. Thus we will often end up with a too early detected

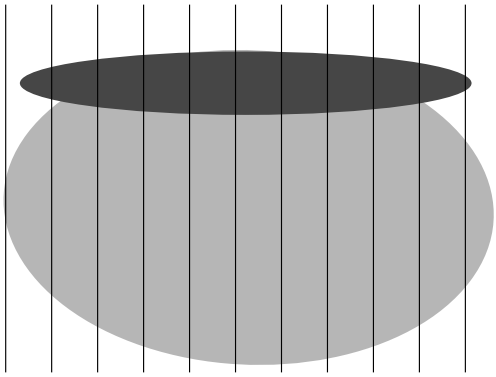

Figure 33: An illustration of a feature with relatively high opacity in front of a feature with lover opacity.

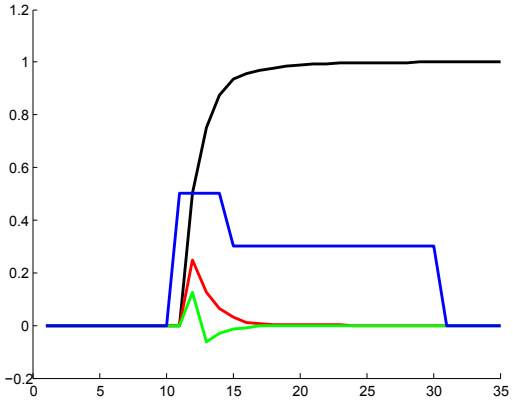

Figure 34: The opacity values (blue), accumulated opacity (black), β-function (red) and γ-function (green) of a example with a feature with relatively high opacity in front of a feature of low opacity.

end-position of a feature. Now since the opacity gain is smaller at the end of a feature, it holds that in most cases, such a false detection of the end of the feature does not influence the computed opacity contribution greatly. Never the less we deal with an information loss again.
When choosing a detection criterion, we have to consider that experimental data always underlies a certain noise from the measurement, see the the torso example Figure 16. In my opinion the neglect of features with low opacity gain and thus probably with small opacity contributions is cheaper than splitting those features in parts, because each detected boundary will be a node in a graph during the algorithm later on. But the weakness of the β-criterion with overlapping features and the distortion of the detected feature boundaries call for the γ-criterion when we can assure that there is no noise in the data or when it is clear that features overlap in the dataset.
Because of that it is for the best, when the user can decide which criterion he chooses. Since we experiment with medical and thus noisy dataset, we will mostly use the β-criterion during this work.

1.2.6 Other Picking Algorithms

In the following we will introduce and shortly discuss a few other existing picking algorithms and a two new algorithms that were tested for this thesis.

First Hit
This is the easiest and most naive method. The algorithms stops the first time it hits something opaque and returns this position as the picked one.
Obviously this method is untrustworthy, for foggy data-sets will lead to an almost immediate return of the picking position and thus cause the user to pick directly on the

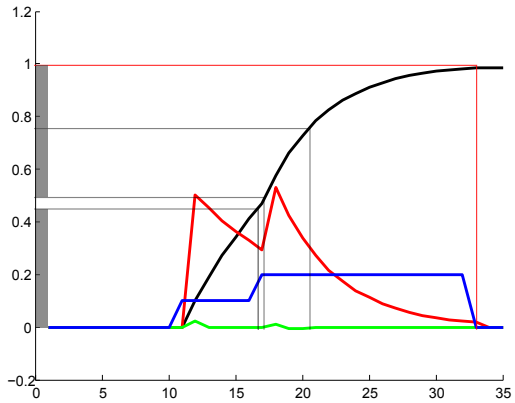

Figure 35: Using an ϵ big enough both features can be detected separately but the opacity contribution is computed wrong. The grey lines indicate the computed feature boundaries, the red line indicates the real backside boundary of the last feature.

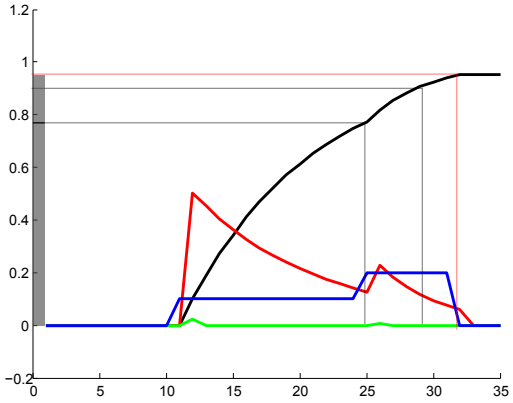

Figure 36: Using a suitable ϵ both of the features can be detected separately, the opacity contribution is computed wrong but the weight ration of the features is not reversed.

boundary of the data-set. On the other hand, even clear data-sets can contain some slightly visible artefacts or some misfits of the data in front of the feature, those would always lead to false picking. Therefore this method is not a real algorithm for feature detection. See [WM] for more information.

Threshold

In this method the opacity is most commonly accumulated according to formula 3, the algorithm returns the first position on the ray that exceeds some limit value, usually 0.9. This algorithm is not intuitive and in order to find the perceived feature boundary one has to adjust the limit value, as the picked position can lie inside the feature. Furthermore the algorithm will return no results when dealing with almost transparent features. See the work of Gobbetti et al. [GPZT] for more information.

Maximal Value Picking

Along each ray the sample with the maximal intensity is selected. With this picking criterion it can easily happen, that the algorithm selects a position that appears transparent due to the choice of the transfer-function. I refer to Bruckner et al. [BSGHBYD] for more information.

Maximal Opacity Picking

Similar to maximal value picking the algorithm selects the sample that contributes the most to the final pixel. But the sample with the highest contribution does not necessarily belong to the feature with the highest opacity contribution. I derived the idea for this picking criterion from the Maximal Value Picking criterion.

Gradient Picking
Here the picked position i is defined by $\max_k (\beta^{acc}(i_k))$.
In this algorithm it could happen, that the picked position lies behind a feature, that fully covers the picked position but that has a smaller gradient. See [BNG] for more information.

Second Derivative Picking
Similar to the gradient picking, in this algorithm the picked position i is defined by $\max_k (\gamma^{acc}(i_k))$.
When the data-set is free from noise this picking criterion can lead to satisfactory results but similar to gradient picking it can easily happen that the picked position lies behind a feature with bigger opacity.
I derived this picking criterion from the Gradient Picking criterion, while analysing the behaviours of the α^{acc}-, β^{acc}- and γ^{acc}-functions. In almost all cases the detected positions of this criterion corresponded to the positions computed with WYSIWYP using the γ-criterion. Which is natural since there is almost always only one local maximum of the γ^{acc}-function in one interval, detected with the γ-criterion.

2 Surface Patches Determination

In this chapter we will discuss the algorithm *surfseek*, that does not cast one ray and thus calculates one pick position, but that casts multiple rays and then calculates a surface patch of a feature computing a min-cut on a constructed directed graph. This will lead to new specific problems, which will be examined in the further course of this work. But first of all we will discuss what sets *surfseek* apart from other algorithms.

2.1 Why a new algorithm

When we try to construct a new algorithm, we have to answer one question first: Why? Why do we do all the work? If there is already a suitable algorithm, then we could use it and save us the time. So what makes *surfseek* different from other feature detection algorithms?
There are many papers on surface segmentation for 3D scalar-fields. Our approach is new since it deals with DVR-images and is the first approach that is oriented on the visibility of a feature. Furthermore as far as I know, all of the segmentation algorithms use regular samples distribution. With regular we mean, that the samples (or nodes) or nodes are arranged in a regular equidistant grid. For example one can look at the work of Kang Li et al. [LWCS]. In this paper a non trivial graph is constructed to compute a min-cut on it, but the skeleton of the graph is a highly regular graph that connects samples $s_{x,y,j}$ on one ray to the last sample with a higher index in the range of $[j, j+\Delta x]$, where Δx is a smoothness constrain chosen by he user. See Figure 37 for better insight. The advantage of such a graph is, that one can easily control the smoothness of the computed surface. In fact in the paper "Optimal Surface Segmentation in Volumetric Images - A Graph-Theoretic Approach" the computed surface must fulfil the smoothness constrain. But this is also the big drawback of this approach. The user has to judge which Δx he has to choose, for the wanted surface. It could happen that the desired surface does not fulfil the smoothness constrain and thus can not be selected at all. This means that the algorithm requires an estimate about the surface beforehand.
In the paper "Computing Exact Discrete Minimal Surfaces: Extending and Solving the Shortest Path Problem in 3D with Application to Segmentation" [LG] Leo Gardy takes a different approach.
The approach in this paper is to divide the 3D data-set in 2D slices and compute the a shortest path in each slice, where each pixel of the slice represent one node in a graph. Similar to the work of Preis [PJP] the shortest paths represent the boundaries of the features. After all slices have been treated the shortest paths are assembled to a minimum weighted surface.
The weakness of this approach is the isolation of the data in slices. With this approach there is no chance to consider the 3D neighbourhood of the data-points.
With *surfseek* we take a different approach, instead of constructing a graph from all of the sample points we only use the detected feature boundary points as the nodes in

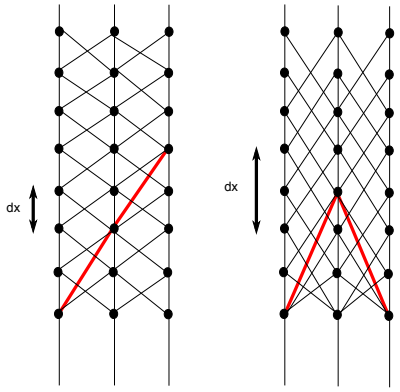

Figure 37: The red lines illustrate surfaces that can not be computed with the algorithm in [LWCS], because they violate the smoothness constrain dx.

our graph. This gives us the advantage, that the constructed graph will be significantly smaller, but we can not use any of the approaches on the regular graphs, since we can not guarantee such a structure.

2.2 Direct Patch Reconstruction

The easiest and most intuitive way to compute surface patches would be to perform the mentioned picking algorithms for each single ray, and then compute a surface from the picked positions. In fact this naive method could lead to satisfactory results, however there are a few problems that will cause jaggy surfaces or even tremendously wrong surfaces. Later on we will compare the surfaces from a direct pick with those from *surfseek*. Here are some situations that cause problems:

- **Misfits in the data.**
  It can happen that the data has some misfits, say some particle with high opacity in front of the feature. This would cause that the direct picking will recognize the misfit as a boundary of the behind lying feature, and thus cause a jagged peak in the surface.

- **Overlapping features.**
  Similar to the previous problem of some misfits in the data, one could have some small structures overlay the perceived feature. One could think of a blood vessel in front of an organ. Again direct picking would lead to composition of two surfaces and thus result in a false surface.

- **Foggy features**
  Foggy features or foggy datasets in general cause the most problems for the surface detection. First, the algorithm can find several boundaries within one foggy feature and it is not predictable which boundary will be used for the picked position. Second, it is very hard for the user to identify feature boundaries for himself, this makes it very hard to say what will be perceived as a surface if a surface is perceived at all.

2.3 Surfseek

This sections contains the basic *surfseek* algorithm and an introduction to the basic mathematical foundation that is needed to understand *surfseek*.
As already mentioned *surfseek* casts multiple rays that cover an area selected by the user. This is both an advantage and disadvantage of the algorithm. The advantage is that it is easier to find and sort out misfits in the data. The disadvantage is the following, when using WYSIWYP in some cases the algorithm picked a position that was not intended by the user, in this case the user assumed that "something gone wrong" and re-picked the position. Using multiple rays we do not have this option any more. The user could cast all rays again but if one position is wrong it could mess up the whole surface.
The idea of the algorithm is to find the surface by searching for greatest influences to the resulting pixel in the opacity values of the samples on the rays. Here is a step by step description of the algorithm:

1 The user selects a surface on the screen. The selected pixels are transformed into world coordinates and a parallel ray casting is applied for each of the stored coordinates.

2 Along each ray the the algorithm detects feature boundaries according to the γ- or the β-criterion. The positions and opacity values are stored for all detected feature boundaries.

3 Using the stored values, the algorithm constructs a network-graph with the samples positions being the nodes and construct the edges according to some criteria. All edges receive certain weights that are dependent on the stored opacity and position values.

4 The Boykov-Kolmogorv-Max-Flow algorithm is applied to the graph to compute the maximal flow. Using the Max-Flow-Min-Cut theorem the minimal cut is computed afterwards.

5 A surface is reconstructed using the minimal cut.

6 Optional: The computed surface can be filtered and recomputed.

7 The computed surface is displayed.

Before going into detail how the graph is constructed, we will discuss how the Boykov-Kolmogorv-Max-Flow algorithm works, since it will help to understand what necessary conditions the graph must satisfy. Furthermore we will give an overview of all terminology that is needed to understand the algorithm.

2.3.1 Max-Flow-Min-Cut

The Max-Flow-Min-Cut theorem is a theorem from the graph-theory, it states that the maximum amount of flow passing through a network from the *source* to the *sink* is equal to the capacity of a minimal edge cut. Before we go into further detail we have to give proper definitions of the used terms. The following definitions provide a compact overview, for detailed information see [MG] and [DW].

Definition 2.1
Let $D = (V, E)$ be a directed graph. We call $N = (D, s, t, c)$ a **network**, with the *source* $s \in V(D)$, the *sink* $t \in V(D)$ and the *capacity* $c : E(D) \to \mathbb{R}^+ \cup \{0\}$.

Definition 2.2
A **flow** f is a function, $f : E(D) \to \mathbb{R}$.
We define out-coming flow f^+ and in-coming flow f^- for a vertex as follows:

$$f^+(v) := \sum_{v \to u} f(vu) \tag{9}$$

$$f^-(v) := \sum_{u \to v} f(uv). \tag{10}$$

A **flow** f is **feasible** if,

$$\text{(i) } f^+(v) = f^-(v) \text{ for all } v \neq s, t \qquad \text{(conservation constraints)} \tag{11}$$

$$\text{(ii) } 0 \leq f(e) \leq c(e) \text{ for all } e \in E(D). \qquad \text{(capacity constraints)} \tag{12}$$

The **value** of a flow is defined as $val(f) := f^-(t) - f^+(t)$.
The **maximum flow**, is a feasible flow with the maximum value.

Definition 2.3
A **cut** C=(S,T) is a partition of V of graph $G = (V, E)$.
An **s-t cut** $C = (S, T)$ of a network $N = (D, s, t, c)$ is a cut of D such that $s \in S$ and $t \in T$.
The **cut-set** of a cut $C = (S, T)$ is the set $(u, v) \in E; u \in S, v \in T$.
A *s-t cut* C is called **minimum** if

$$c(S, T) = \sum_{e \in C} c(e) \leq \sum_{e \in C'} c(e) = c(S', T') \tag{13}$$

for all *s-t cuts* C'.

Definition 2.4
An s,t-path P in a graph G is an **f-augmenting path**, if:
$s = v_0, e_1, v_1, e_2, ..., v_{k-1}, e_k, v_k = t$ and for every e_i holds

1. $f(e_i) < c(e_i)$ provided e_i is a forward edge
2. $f(e_i) > 0$ provided e_i is a backwards edge.

Definition 2.5
Let P be an *f-augmenting path.* The **tolerance** of P is $\min\{\epsilon(e) : e \in E(P)\}$, where
$\epsilon(e) = c(e) - f(e)$ if e is forward, and
$\epsilon(e) = f(e)$ if e is backward.

Lemma 2.1
Let f be a feasible flow and P an f-augmenting path with tolerance ϵ. Define
$f'(e) := f(e) + \epsilon$ *if e is forward,*
$f'(e) := f(e) - \epsilon$ *if e is backward.* $f'(e) := f(e)$ *if* $e \notin E(P)$.
Then f' is feasible with $val(f') = val(f) + \epsilon$.

Lemma 2.2 (Characterization Lemma)
A feasible flow f is of maximum value if there is NO f-augmenting path.

In Figure 38 the reader can examine a simple example of a network with a feasible flow. The black values represent the capacity of an edge, the green values indicate the flow value, s and t indicate the source and the sink vertex respectively. The arrows show the direction of the edges, the different colours of the vertices indicate whether the vertex belongs to S (filled with white) or to T (filled with black) and the two red edges show a possible minimum cut.
It is worth noticing, that the minimum cut is not unique. This is due to the fact, that we used edges with the same capacities. For *surfseek* algorithm we will assume, that the edges will have different capacities, as it will be the natural expectation in the most cases. To be on the save side, we choose the nodes that are nearer to the viewer if the min-cut is not unique.

2.3.2 Boykov-Kolmogorov algorithm

The *Boykov-Kolmogorov-max-flow* algorithm is an implemented algorithm in the boost library in C++, it calculates the maximum flow of a network, see [BK] for more information on the implementation. For more details on the mathematical background of the algorithm I refer to the very detailed paper of Boykov and Kolmogorov [BK04].
The algorithm is a variety of the augmenting-path algorithm. First an augmenting path algorithm finds the shortest paths from the source to the sink vertex, and increases them by subtracting the bottleneck capacity (minimum capacity) on that path from the residual capacities of each edge and adding it to the total flow. Furthermore the minimum capacity is added to the residual capacity of the reverse edges. The algorithm

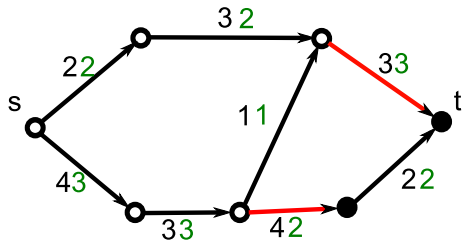

Figure 38: An example of a network. With capacity values (black), flow values (green), and a possible s-t cut (S white vertex, T black vertex).

terminates if no more paths can be found in the residual-edge tree. Instead of finding a new shortest path form source to sink in each iteration, as it is done in most algorithms that compute the maximum flow, the Boykov-Kolmogorov algorithm keeps the found paths as follows:
The algorithm builds a source- and a sink-tree where each vertex has a color that determines if the vertex is part of a tree and if so in which (black=source-tree S,white=sink-tree T, grey=not in a tree), furthermore each vertex receives a status-flag if it is active or passive. In the first step of the algorithm the source is coloured black and the sink white and both have active status. All of the remaining vertices are coloured grey and receive the passive status. Each iteration of the algorithm consists of three phases:

grow-phase: In this phase the active vertices acquire neighbour vertices that are connected through an edge that has a capacity greater than zero. Acquire means that those vertices become active and belong now to the search tree of the current active vertex. We call an acquired vertex a child of the active vertex, which we call a parent of the child vertex. If there are no more valid connections to neighbour vertices, the current vertex becomes passive and the grow phase continues with the next active vertex. When there are no more active vertices left or a vertex discovers a vertex from the other search tree the grow phase terminates. If a vertex from the other tree was found then the algorithm also found a path from source vertex to the sink vertex.

augment-phase: In this phase the path that was found in the grow phase is augmented. First the bottleneck capacity of the found path is computed, then the residual-capacity of the edges in this path is updated by subtracting the bottleneck capacity from the residual capacity. The residual capacity of the backwards edges is updated by adding the bottleneck capacity. Note, that this phase can destroy the build up search trees,

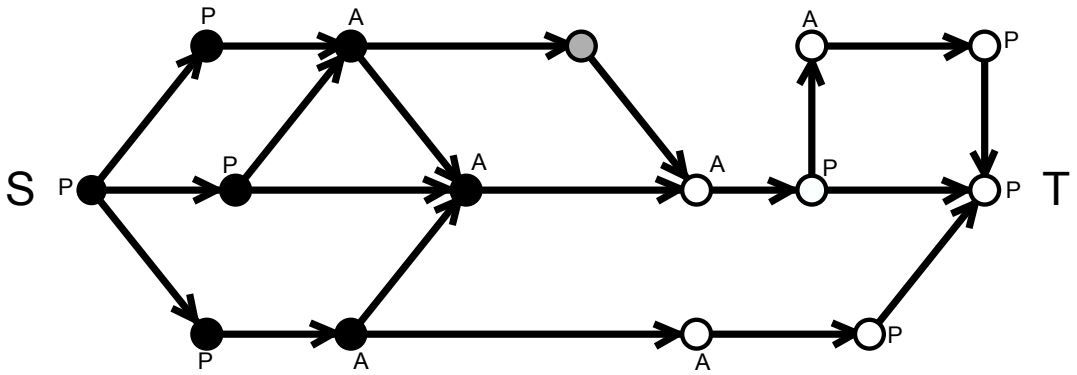

Figure 39: This Graph illustrates the end of the first grow-phase. The phase terminates after three iterations because a white neighbour was found by the black vertex from the Source-tree S in the middle path. A and P denote the active and passive nodes.

as it creates at least one saturated edge. Thus, some of the nodes in the trees S and T may become *orphans*, that is, the edges linking them to their parents are no longer valid (they are saturated). In fact, the augmentation phase may split the search trees S and T into forests. The source s and the sink t are still roots of two of the trees while orphans form roots of all other trees

adoption-phase: In this phase the search trees are reconstructed again by trying to find a new parent for each orphan. A new parent should belong to the same set, S or T, as the orphan. The connection from the child to the parent should occur through a non-saturated edge. If there is no qualified parent to be found, the orphan vertex is removed from the according search tree and is coloured grey as a free node. Furthermore all children of the grey coloured orphan are declared orphans as well. This phase terminates when there are no more orphans left, and thus the search trees S and T are restored.

After the adoption-phase is completed the algorithm returns to the growth stage. The algorithm terminates if the search trees S and T can not longer grow (no active vertices), and the trees are separated by saturated edges. This implies that the maximum flow is achieved. The corresponding minimum cut can be determined by S and T, as it cuts exactly the edges connecting S and T. An example of the algorithm is illustrated in Figures 39,40 and 41.

2.4 Graph Construction

In this section we will discuss, what conditions must be fulfilled by the graph in order that the algorithm returns the desired surface. There are three conditions that must be

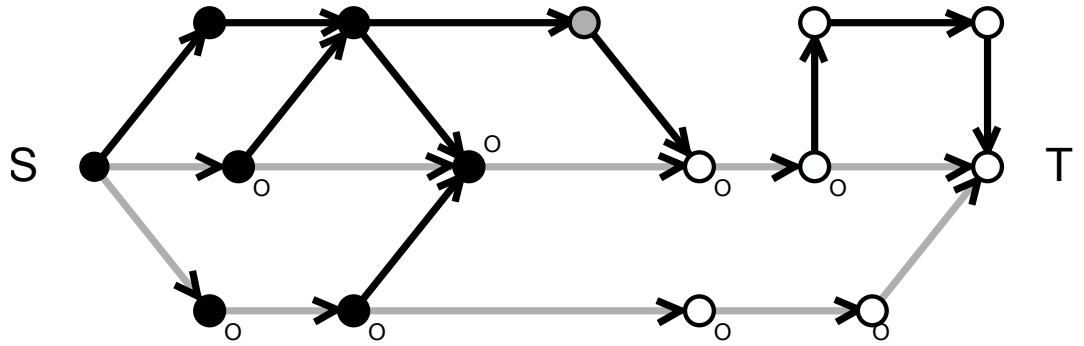

Figure 40: This Graph illustrates the end of the first augment-phase. The grey edges are saturated thus all nodes on the middle and the lower path become orphans (O).

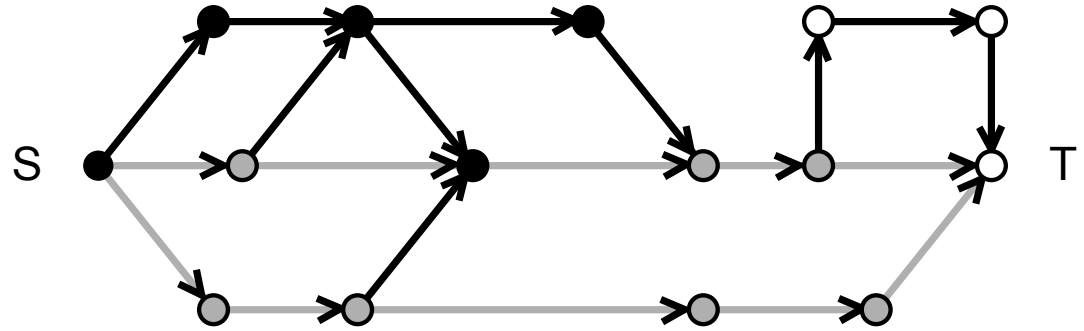

Figure 41: This Graph illustrates the end of the first adoption-phase. After the grey node of the augment-phase was declared an orphan as well, a new S- and T-trees are constructed. In the following Augment phase the algorithm will terminate, because all grey nodes will be discovered without finding a node from the opposite tree.

true for every ray:

1: On each ray the detected cut edge must be unique.

2: If a ray has at least one detected feature, then the algorithm must detect a cut-edge on the ray.

3: The algorithm must not neglect any detected positions.

In the following will step by step build up a graph that satisfies the requirements. In each step the graph will become more complex, and we will discuss possibilities that would cause false detection of a surface. From here on we will call the detected feature surface positions, computed by WYSIWYP, nodes. We assume the viewer being above the graph, thus we will say a node lies behind an other if it lies under the node on in the graph. Furthermore all graphs will be presented as planar graphs for better illustration purposes. Since a graph has no natural dimension he lives in, the limitation on 2D graphs yields no false results. For the Max-Flow computation we create two virtual

nodes source and sink. Each ray will be connected to the source and sink respectively by exactly one edge, that will receive maximum possible weight to make sure that it is not cut.

2.4.1 Intra-ray-edges

With intra-ray-edges we denote edges that connect the nodes on the same ray. We will call those edges intra-edges for short. The first observation that we make is, that we can not simply connect the nodes on the ray with the following one, and the first node on each ray with the source and the last node with the sink. This would lead to two problems:
First: if only one node is detected, the algorithm must cut an edge that is connected to the source or the sink node. This would lead to uncertainty which edge is in the cut and condition 1 will be disregarded. Furthermore since the edges that are connect to the source or the sink receive maximal possible weight, a cut of those could cause errors in further flow computation.
Second: If there are two or more nodes on the ray, then connecting the first node to the source and the last one to the sink, would not cause a cut of one of the unwanted source or sink edges. But the user will have to select if the front or the back nodes should represent the surface, and we want the algorithm to be explicit. Furthermore always taking the front or the back node could lead to jaggy surfaces as one can see in Figure 42.
This Problem can be solved by adding virtual nodes at the end of each ray, see Figure 43. Adding a virtual node allows to select single nodes on a ray, furthermore it gives us an extra edge, this way we can project the weights of each node on the following edge.
With the virtual nodes we construct the Graph as follows:
Each node get an index x, y, z, where x and y determine the ray on which the node lies, and n or z determines the position of the node along the ray ending in the virtual node.
A weight is assigned to each node v, that is

$$w(v_{x,y,n}) = 1 - (\alpha^{acc}(i_{n_0}) - \alpha^{acc}(i_{n_{max}}))\,, \tag{14}$$

where i_{n_0} is the sample position on the ray where the n-th feature starts and $i_{n_{max}}$ the sample position where the n-th feature ends.
We denote the edges by $e_{x,y,z} = (v_{x,y,z}, v_{x,y,z+1})$ or e_z if the ray position is irrelevant. An edge $e_{z,z+1}$ obtains the weight $w(e_z) = w(v_z)$, that is we project the weight of each node, except the virtual ones, on the following edge.
Clearly a min-cut on such a graph corresponds exactly to a direct pick with the WYSIWYP algorithm.
By now we have the graph $G = (V, E)$ with:

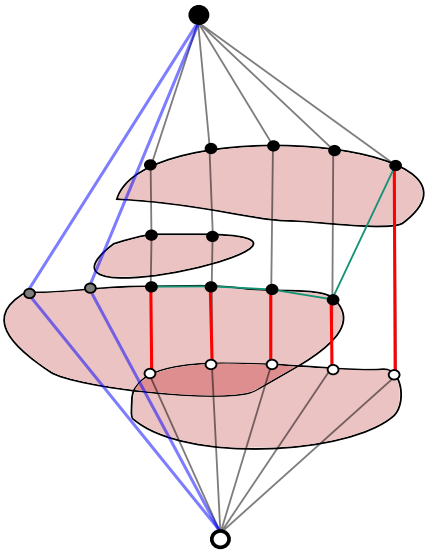

Figure 42: The blue edges indicate that the algorithm can cut both, the source and the sink edge. The red edges indicate the cut edges, the surface can be reconstructed from the upper or the lower nodes.

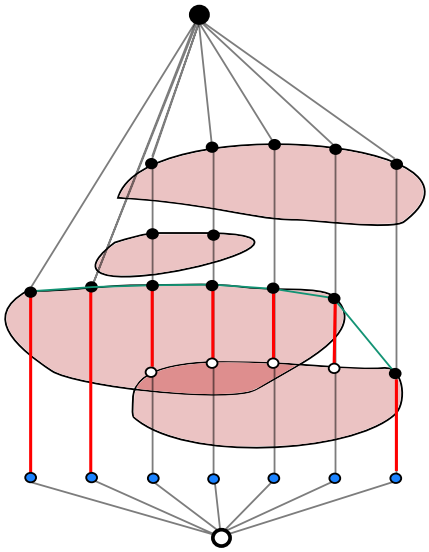

Figure 43: The blue nodes represent the virtual nodes, naturally those are coloured white, as they belong to the sink-tree T. Using the virtual nodes at the end of the ray, lead to a natural selection of the front nodes as the surface representative.

$$V = \{v_{x,y,z} = i_{x,y,n_0}\} \tag{15}$$
$$E = \{e_{x,y,z} = (v_{x,y,z}, v_{x,y,z+1})\}, \tag{16}$$

with the assigned weight $w(e_z) = w(v_z)$ and the natural direction from $v_{x,y,z}$ to $v_{x,y,z+1}$. All backward edges get the zero capacity.

2.4.2 Inter-ray-edges

With inter-ray-edges we denote edges that connect rays with their corresponding neighbour rays. With neighbour rays we mean rays that differ in exactly one index by exactly one. That is ray $r_{x,y}$ has the neighbours $r_{x+1,y}$, $r_{x-1,y}$, $r_{x,y+1}$ ans $r_{x,y-1}$.
Adding inter-ray-edges allow us to take the nodes of the neighbour rays into account, but we have to be careful not to disregard the three conditions. The main difficulty with the construction of the inter-ray-edges is the fact, that the number of nodes can vary greatly for different rays. This irregularity could lead to unwanted or even ambiguous cuts, see Figure 44 and 45. These figures show extreme graph constructions that lead to unwanted cuts. For simplicity we assumed that all edges in this graph have the same capacity. Even though the graphs look extremely irregular constructed, one could ad

further edges with sufficiently big capacities without changing the cut.
In Figure 44 the extra edges coming from the middle ray force the algorithm to cut the first edge on the ray, since otherwise it would have to cut more then one edge.
In Figure 45 the multiple outgoing edges from the first white node on the middle ray cause to cut the foregoing edge. The edges that come from the neighbour rays to the third node on the middle ray causes the flow to continue on the middle ray, again the algorithm cuts the following edge on the middle ray since it is cheaper that cutting two edges. The cut on the outside rays can run through the second, third or the fourth edge.
In order to avoid the unwanted cuts, the graph is not allowed the have multiple inter-ray-edges on one node, that is every node has at most one edge connecting it with a neighbour ray, and the weight of edges must be adjusted according to the neighbourhood. In the following we will first discuss the graph construction, and we will explain the weight selection later on. With $w(e)$ we will denote the weight of an edge. The reader can imagine the distance between two nodes as the weight of the corresponding edge, as it fulfils the purposes of illustration. We will clarify the weight selection in a later section.
Since we can not use multiple inter-ray-edges, we have to preselect the most representative edges, a naive attempt is to select edges with the minimum weight. This would lead to a cut that is equivalent to the direct pick, see Figure 46. Here the inter-ray-edges are highlighted in green. It is not necessary to cut them in order to stop the flow.
A better way is to connect the edges with the next ongoing node. That is we construct edges

$$e_{z,z'+1} = (v_{x,y,z}, v_{x',y',z'+1}), \text{ where} \tag{17}$$

$$w(e_{z,z'}) = \min_k \{w(v_{x,y,z}, v_{x',y',k})\}. \tag{18}$$

With x' and y' we denote the ray indices of a neighbour ray of $r_{x,y}$.
But due to the irregularity in the number of nodes per ray, multiple edges in one node are very common with this edge construction, see Figure 47. To prevent multiple edges from one node to a neighbour ray, we simply keep the edge with the smallest edge weight, and delete all other edges ending in the specific node.

2.5 Weight selection

In the following chapters we will discuss how to assign weights to the edges, or what is the weight-function for the edges. First we settle the components that the weight should have, then we will deal with the weight adjustment of single edges.
It is obvious that the weight-function has a crucial impact on the later surface construction. Therefore we have to determine some conditions that the constructed surface has to fulfil.

Requirement 1: The algorithm should select the most visible surface, i.e. the one

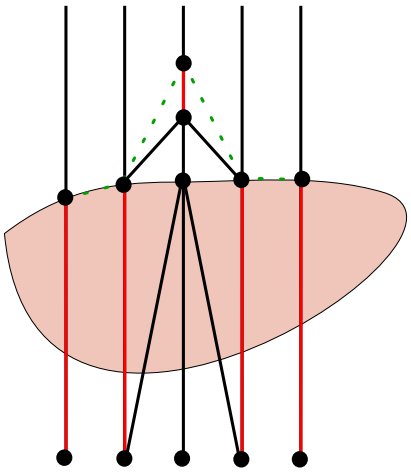

Figure 44: The red edges indicate the cut edges. Suppose that the ray in the middle hits some foggy area, and thus register multiple features. All the other rays register only one feature. Connecting the graph this way will lead to an unwanted jaggy surface (dotted green line).

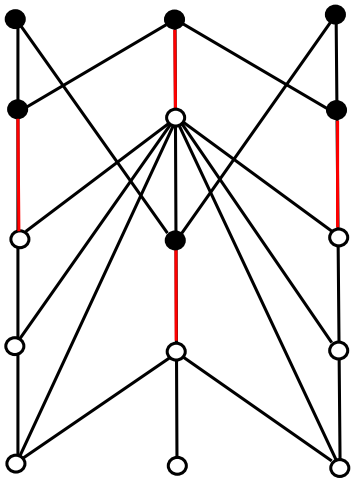

Figure 45: The red edges indicate the cut edges. Assuming that all edges have the same capacity, the min-cut contains two edges on the same ray.

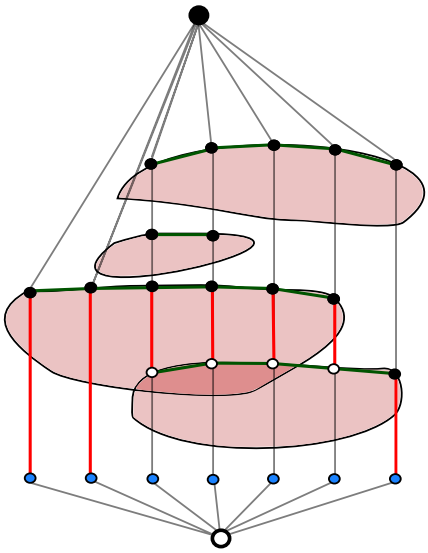

Figure 46: The selection of inter-ray-edges (green) with minimal weight leads to unnecessary edges, that don't influence the flow.

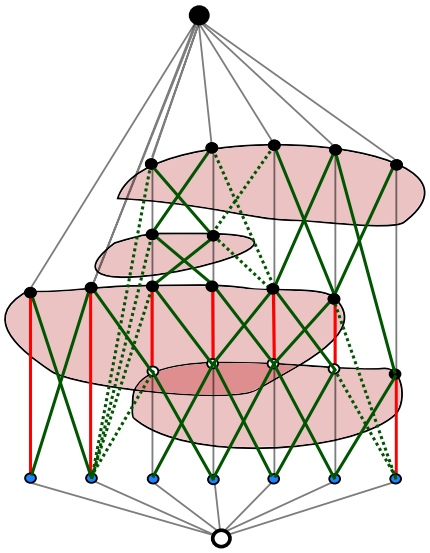

Figure 47: Connecting the nodes with the ongoing nodes on the neighbouring rays, that follow nodes which form the minimal weight, can lead to multiple edges in one node, dotted line.

with the most opacity influence. Because a feature with the most opacity influence is the most visible and thus is probably meant to be selected by the user.

Requirement 2: The computed surface should belong mostly to only one surface, that is to say the computed surface should be smooth. Dealing with foggy datasets or some small opaque features should not influence the surface of a big feature behind.

Using direct picking would cause highly jaggy surfaces when dealing with foggy datasets, because the foggy regions can be detected as features with high opacity influence. The algorithm should select nodes with lesser opacity influences if the resulting surface will be smoother, since it is unlikely for a surface to have single jumps. See Figure 48.

Requirement 3: If the opacity influences of two features are very similar the algorithm should compute one smooth surface and not jump between the two surfaces.

Requirement 4: If a surface ends and the user selects further area, the choice of the new surface should not depend on the distance to the previous surface but on the visibility of the new feature only.

The algorithm should not neglect a surface if it lies further away from the previous selection. The jump to a more opaque feature should be more probable, since this feature is more likely to be seen by the user. See Figure 49.

Requirement 5: A surface that is lightly inclined with respect to the viewer, should not get a significant bigger weight than a surface that is totally perpendicular to the view-direction.

It is natural not to expect totally planar surfaces, thus a weight of a surface that is bend should not be bigger than a totally planar surface.

The Requirements 2 and 3 can be fulfilled by using distance of the nodes to each other as one weight criterion. The Requirements 1, 4 and 5 demand that the opacity influence is a part of the weight.
It is obvious that the total weight of an edge will be a combination of the distance-weight and the opacity-weight. Since there are more then one possible candidates for the distance- and opacity-weights, we are going to discuss both of them in detail. Hereby we have to consider that the edges that should be contained in the minimum cut should receive smaller weight, than those outside the cut.

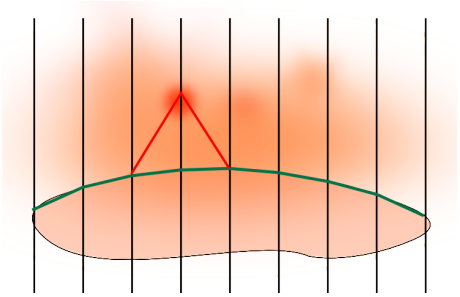

Figure 48: The computed surface should ignore the opaque area and compute a smooth surface, represented by the green line. Thus avoid the jaggy jump (red line).

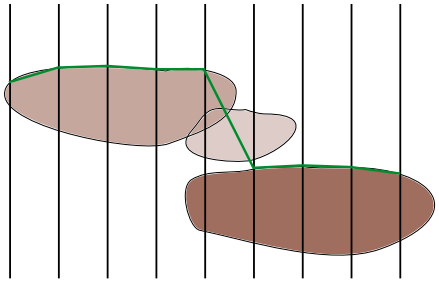

Figure 49: The feature on the left side ends. When further surface must be selected, then the surface should jump to the most opaque feature.

2.5.1 Distance as Weight

When using distance as a part of the weight function we have two clever options for the choice of the weight-function. The first choice is the euclidean distance the second choice are the index difference of the nodes.

Euclidean Distance

The euclidean distance between two nodes N and M is

$$d_{N,M} = \left|\overline{NM}\right| = \sqrt{(N_x - M_x)^2 + (N_y - M_y)^2 + (N_z - M_z)^2}. \tag{19}$$

It is clear that the edge with the minimal euclidean weight is perpendicular to the rays if all rays are parallel. This is not the case with perspective rays. When choosing euclidean distance as weight, one has to consider two behaviours of the weight-function.
When using parallel ray casting, the weight function is highly dependent on the ray density. This is illustrated in Figure 50, if the minimal distance $d_{A,B}$ between the ray is getting bigger, the difference between $d_{A,C}$ and $d_{A,D}$ vanishes.

$$\frac{d_{A,C}}{d_{A,D}} = \frac{\sqrt{(d_{A,B}^2 + d_{B,C}^2)}}{\sqrt{(d_{A,B}^2 + d_{B,D}^2)}} \tag{20}$$

$$\lim_{d_{A,B} \to \text{inf}} \frac{d_{A,C}}{d_{A,D}} = 1. \tag{21}$$

However this is not a real problem if one chooses a sufficient ray density.

When using perspective rays, there is another problem with the weight. Since the rays tend to move further away from each other, the distance between the nodes grows naturally with the distance to the camera position. This has two consequences:
First, as mentioned above, the weights of the further nodes become more similar.
Second, the nodes that are further away get bigger weights, then those in the front. See

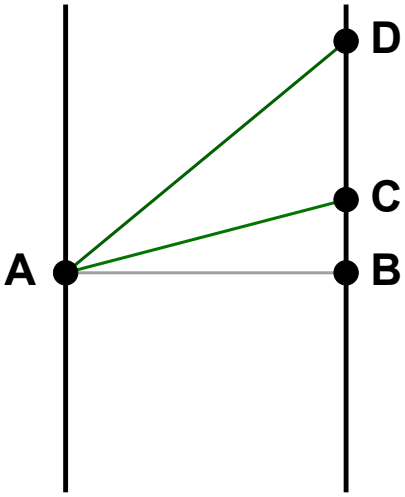

Figure 50: With growing distance between the rays, the distances $d_{A,C}$ and $d_{A,D}$ become more and more similar.

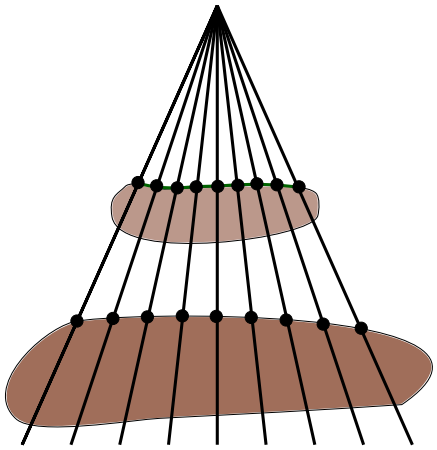

Figure 51: The feature on the left side ends. When further surface must be selected, then the surface should jump to the most opaque feature.

Figure 51 for illustration, here the front less opaque feature is selected because those nodes are closer.
By choosing high ray density, those consequences can be weaken, but they can not be removed this way. Thus perspective ray casting, should be avoided if using euclidean distance as a weight.

Index Difference
Using index difference as weight brings the independence of the ray density but it also has two disadvantages. Casting rays on a dataset that is inclined causes errors, when the rays start taking samples in the dataset only. When using the index difference as weight one should start to count the samples from a mutual plateau. If the the samples are counted as soon as a ray enters the data-set, it can cause a distortion of the weights. Because samples with same indices can lie wide apart if the dataset borders are not perpendicular to the viewing direction, see Figure 52 and 53.
The other disadvantage is by far more significant. When two detected nodes have the same index, the the corresponding edge immediately gets the distance weight zero. This makes it much harder to include the opacity weight in the total-weight, since the natural way is to multiply the two weight components and not to add them up. Furthermore this is a direct violation of requirement 5. One could try to fix this problem by always adding one to the index difference, but this again would distort the total weight.

Conclusion
When the nodes lie wide apart on the rays, both weight functions behave similar. Re-

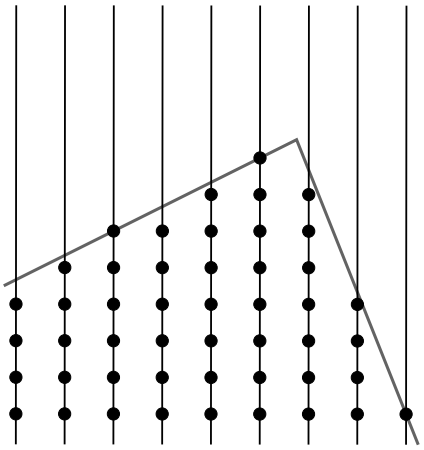

Figure 52: When the sample counting starts as soon as the rays enter the dataset, samples with the same index can lay wide apart.

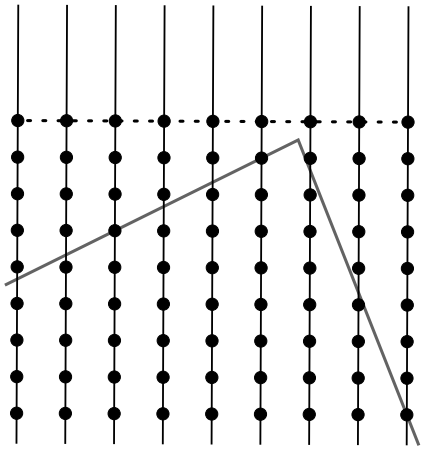

Figure 53: Starting from a mutual plateau (dotted line) causes the samples with same index be nearer than samples with different indices. But the algorithm has to store more information.

quirement 2, 3 and 5 are all fulfilled with the euclidean distance, while the first requirement is not violated. using index-difference as weight could violate requirement 1, 4 and 5. Considering this the choice of the euclidean distance as weight appears to be more convenient, therefore we will use the euclidean distance on parallel rays as a distance weight function $w^{dist}(e)$. For computational purposes all distance weights will be normalized.
The fourth requirement can not be fulfilled with both of the introduced weight functions. But the natural properties of the minimum cut will fulfil the fourth requirement if the new feature is big enough. That is if the inevitable jump lead to a feature with a sufficient big surface, then a bigger jump would be cheaper that cutting all of those edges with higher weight of the bigger surface.

2.5.2 Opacity as Weight

Requirement 1 and 4 call for the opacity to have an influence on the weight function. We will discuss two possible opacity weight functions in this section.

Opacity contribution
The gain of opacity for a certain feature as computed in WYSIWYP is a natural candidate for the opacity-weight function. Considering requirements 1 and 4, the opacity gain of a feature should make up, at least partly, the total weight of an edge. This is an natural conclusion since the features with the most opacity gain are the most visible ones. Thus it is highly probable that the user wants to select exactly those features.

The gain of the opacity lies in the interval $[0, 1]$. Since we want it to be more probable to cut edges that connect highly visible features, we have to reverse the weights:

$$w^{op}(v_{x,y,z}) = 1 - (\text{Opacity gain of the z-th detected feature } v \text{ on ray } r_{x,y}). \tag{22}$$

To project the weights from the vertices $v_{x,y,z}$ and $v_{x',y',z'}$ on the edges we simply compute the mean weight of the vertices:

$$w^{op}(e) = \frac{w^{op}(v_{x,y,z}) + w^{op}(v_{x',y',z'})}{2}. \tag{23}$$

We will call the opacity weight of the n-th feature on a specific ray w_n^{op} from now on and we will add the indices x and y if the ray position needs to be clarified.

Opacity contribution difference
Similar to the index difference, one could also argue for the difference of the opacity contribution as weight. But as with the index difference the difference of the opacity contribution as weight can lead to an unreasonably preference of features with small but very similar opacity contributions. In practice this weight selection led to jumps between very small features that lied far apart, but that had very similar opacity contributions. Thus we will not use the difference of the opacity contribution as weight.

2.6 Combining the weights

The method we use to combine the weights has en essential influence on the properties of the minimum cut and thus on the surface detection. While the distance weight function satisfies requirements 2, 3 and 5, the opacity weight function satisfies the requirements 1, 4 and 5. When combining the weights it is not manageable to fully satisfy all of the requirements, since they may conflict with each other, instead we have to find a compromise that suits our needs.
Philip Preis already discussed different weight combinations in his diploma thesis [PJP]. Preis used the weights to find a shortest path in a graph, since this approach is different from ours, we can not automatically adopt his results.
In the following we will denote the total weight of an edge as

$$w(e) = w\left(w^{op}(e), w^{dist}(e)\right). \tag{24}$$

We will discuss different possibilities for the weight function in the following.
A naive and not a good choice for the weight combination is the function

$$w(e) = w^{dist}(e) + w^{op}(e). \tag{25}$$

Choosing the sum as the combination of the weight leads to unwanted jaggy surfaces. The bad surface is constructed because the weight don't matter if they are small. Even

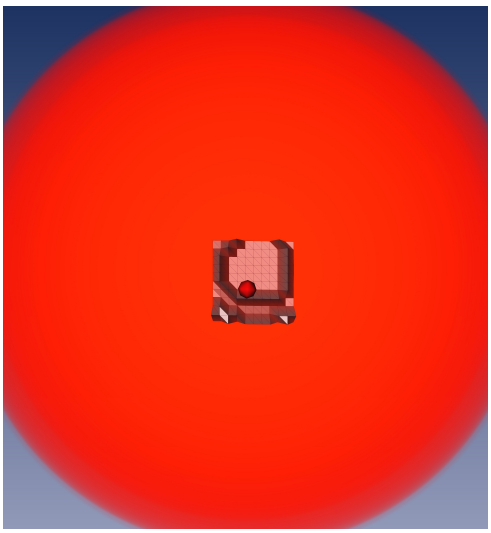

Figure 54: A simple multiplication of the weight components as combinations leads to highly jaggy surfaces. 11x11 rays were projected on a sphere to compute this surface with the γ-criterion.

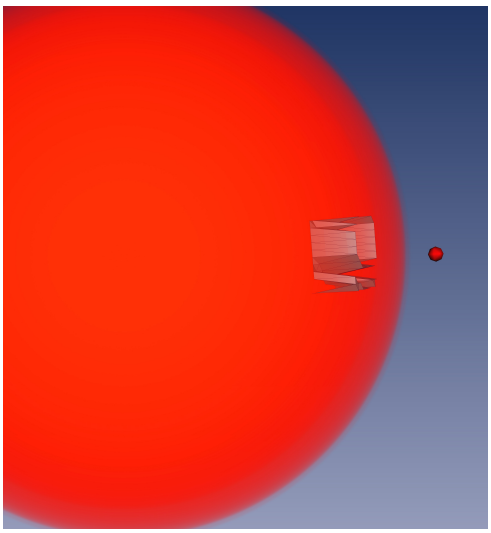

Figure 55: The same surface as in Figure 54 from the side view.

when normalizing the weights, one can have unwanted results if one of the weights is very small. For instance if it is smaller than some $0 < \epsilon < \max\left(w^{op}(e), w^{dist}(e)\right)/10$:

$$w(e) \approx w^{op}(e) \text{ if } w^{dist}(e) < \epsilon \tag{26}$$

$$w(e) \approx w^{dist}(e) \text{ if } w^{op}(e) < \epsilon. \tag{27}$$

A better way is to combine the weights is a simple multiplication:

$$w(e) = w^{dist}(e) \cdot w^{op}(e). \tag{28}$$

Choosing a simple multiplication as the combination of the weight functions can lead to jaggy surfaces too. This is due to the fact, that the distance weight is to weak and thus the smoothness is not provided. This is especially true when we use the γ-criterion, because the γ-criterion can split up the features, and thus naturally tends to jaggy surfaces. See Figure 54 and 55. Using the β-criterion the ball shown inf Figures 54 and 55 would be detected as one feature, thus we can not demonstrate the same behaviour on this dataset. But the same construction of false surfaces can be observed with experimental data-sets, see Figures 56 and 57. Note that the unwanted detected points on the upper right side of the surface patch are inevitable, because the bone tissue is uncommonly thin at this position and thus it forms a hole, where the rays have no chance to detect a feature boundary. This can be observed in Figure 58 when using a different transfer-function.
In order to get a smoother surface one has to increase the leverage of the distance weight.

Figure 56: A simple multiplication of the weight components as combinations leads to unwanted surfaces. For the computation of this surface the β-criterion was used. Note that the unwanted detected points on the upper right side are inevitably.

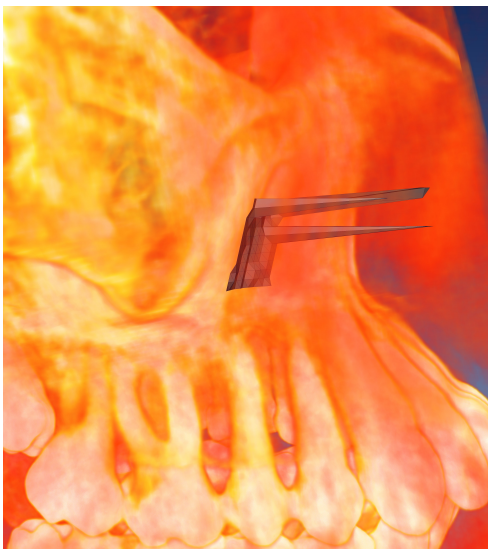

Figure 57: The same surface as in Figure 56 from the side view.

A simple way to adjust the leverage of the weights is to raise the power of those:

$$w(e) = \left(w^{dist}(e)\right)^k \cdot \left(w^{op}(e)\right)^j . \tag{29}$$

It turns out that, the choosing $k = 2$ and $j = 1$ is a good choice, when the surface is relatively perpendicular to the viewing direction. When dealing with surfaces that are relatively planar to the viewing direction this choice of $k = 2$ and $j = 1$ can lead to jumps from one surface to another surface, this behaviour is good illustrated when using the γ-criterion on the ball seen in Figure 54. See Figure 59 for illustration. The jumps occur between the features the ball was divided up in. The previous weight choice of $k = 1$ and $j = 1$ would lead to the a surface as in Figure 61 and 60, this is a by far smoother surface, but it is still not an adequate one.
When the wanted surface is relatively perpendicular the computed surface described leverage of $k = 2$ and $j = 1$ works very good in most cases, see Figures 62 and 63 for the ball dataset with the γ-criterion and Figures 64 and 65 for the skull dataset with the β-criterion.
However the choice of $k = 2$ and $j = 1$ may be not enough when we use *surfseek* on a data-set that is strongly contaminated with noise. We can see the leverage factor k as a degree for the smoothness of the surface. The higher we choose k the more the algorithm tends to smoother surfaces. In the most cases the choice of a higher k, say $k = 3$, lead to exactly the same results as with $k = 2$, but when the noise is significantly high a choice of $k = 3$ yields better results. We will provide some examples of a different leverage

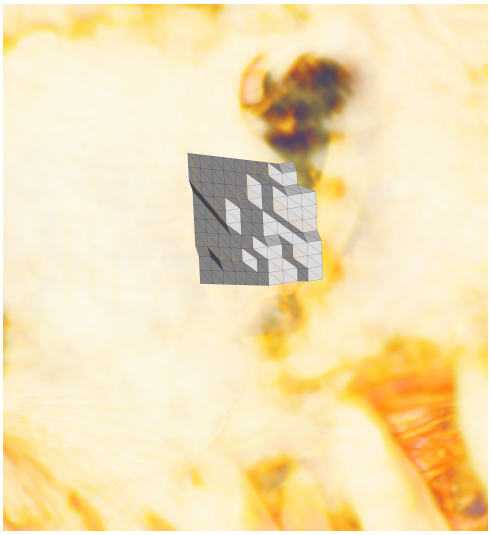

Figure 58: A different opacity function shows that the bone tissue has a "hole" region, and thus the algorithm has no chance to detect a surface there.

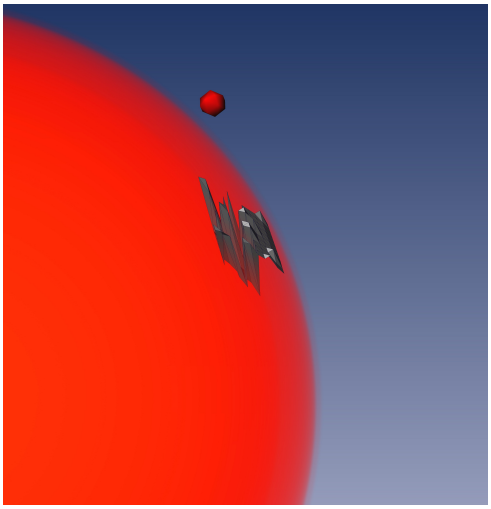

Figure 59: Using the γ-criterion the feature is divided in several features. In order to compute that minimum weighted surface the algorithm jumps from one surface to another, when we choose a big leverage on the distance weight.

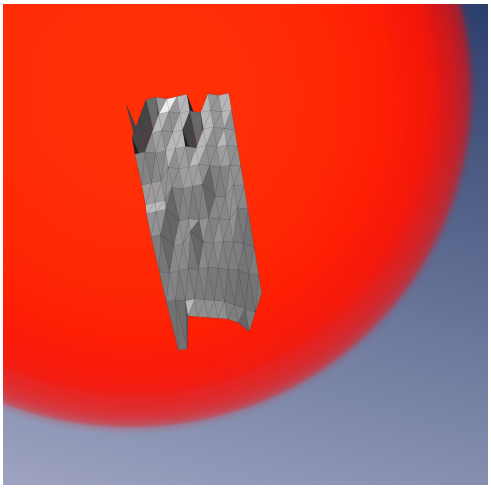

Figure 60: A simple multiplication of the weight can lead to better results, when the wanted surface is relatively planar to the viewing direction.

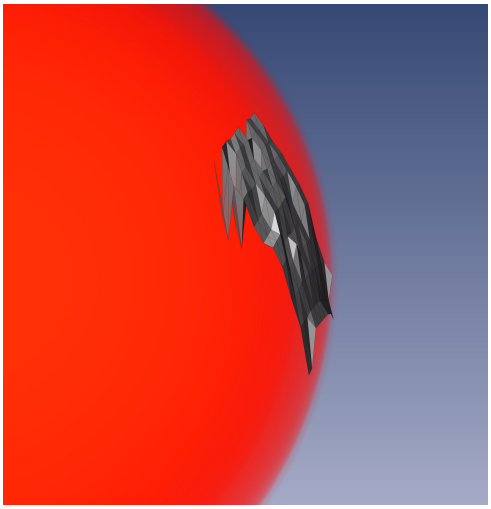

Figure 61: Same projection as in Figure 60 from the side view.

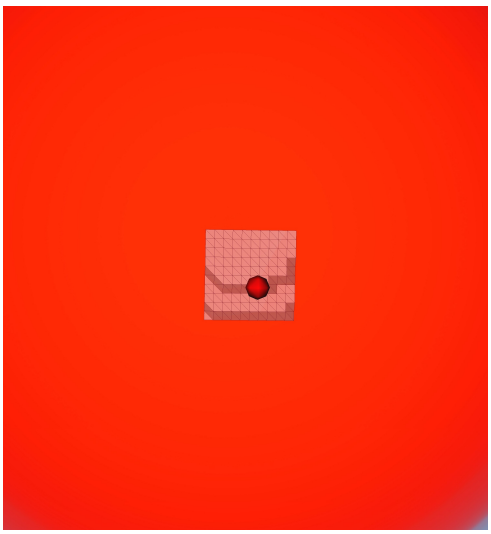

Figure 62: Using a leverage of $k = 2$ and $j = 1$ the surface computed with the γ-criterion is much smoother if the wanted surface is relatively perpendicular to the viewing direction.

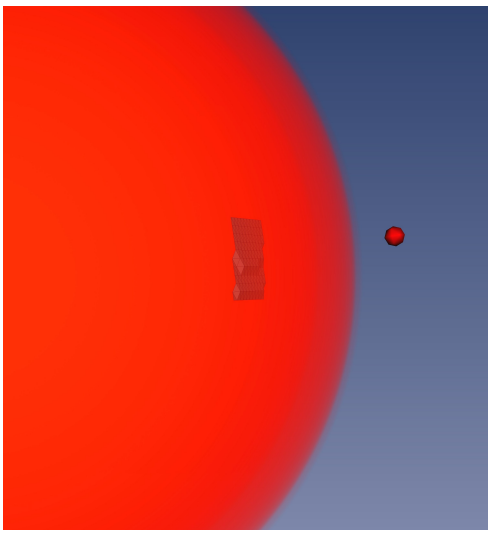

Figure 63: The same surface as in Figure 62.

choices in section 3. Since the choice $k = 3$ has no negative influence on the resulting surface, for the data-sets seen in Figures 64 and 65 it stayed the same, but leads to a smoother surface in highly noisy data-set, we will choose $k = 3$ and $j = 1$ from now on.

All of those thoughts would be irrelevant, when the direct-pick method would give us equivalently good or better results. To show this is not the case, the reader can observe surfaces computed with the direct-pick method for the γ-criterion on the ball data-set in Figures 66 and 67. In this example the computed surface is piecewise very smooth and it nestles itself on the boundary of the ball. This is the case because we have a synthetic example without any noise. But the reader can observe a change in the boundary detection in the middle of the computed surface. This is the point, where the opacity influence changes because the ball is divided up in parts.
As mentioned before the β-criterion would result in a single feature when applied on the ball data-set. When we use the direct pick with the β-criterion on the skull data-set we receive the results as in Figures 68 and 69. The reader can clearly observe that there are several "false" detected points when using the direct pick.
Let's summarize what we have so far. We discussed how we will build up the graph, meaning how the edges are constructed, we discussed different weights for the edges, but we did not explain how we will assign the weight to the edges. This is the content of the next section.

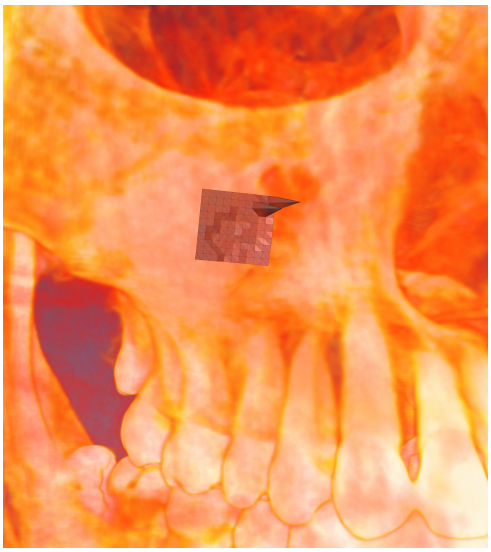

Figure 64: Using a leverage of $k = 2$ and $j = 1$ the surface computed with the β-criterion is much smoother if the wanted surface is relatively perpendicular to the viewing direction. Again the unwanted points on the upper right side are inevitably.

Figure 65: Same projection as in Figure 64 from the side view.

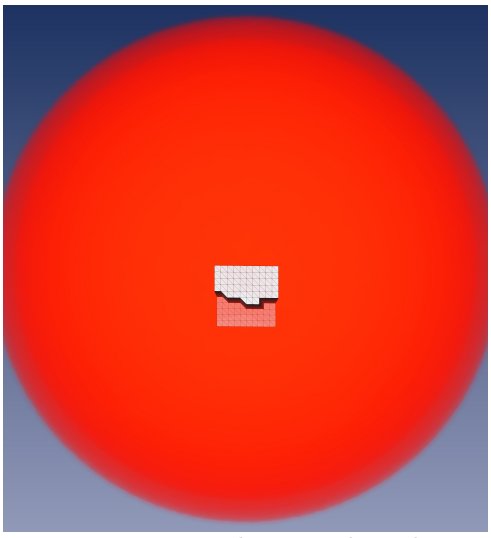

Figure 66: Using a direct pick with WYSIWYP with the γ-criterion the surface is piecewise smooth, but the algorithm yields a surface jump, when the opacity influence changes.

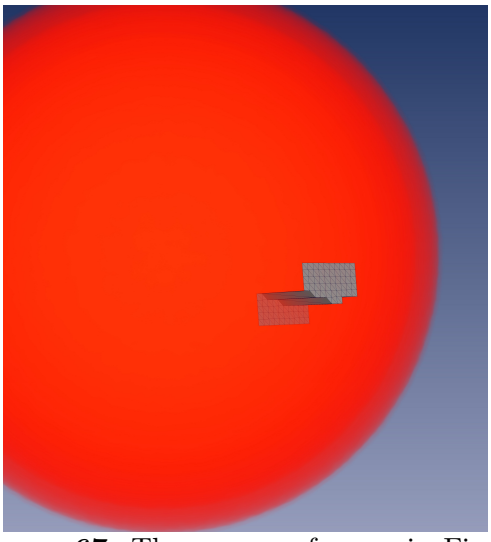

Figure 67: The same surface as in Figure 66 from the side view.

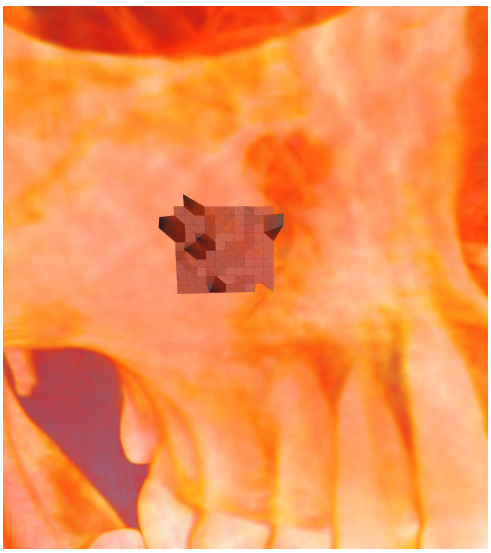

Figure 68: When using a direct pick with WYSIWYP with the β-criterion several points are detected false.

Figure 69: The same surface as in Figure 68 from the side view.

2.7 Assigning the weight to the edges

As discussed in section 2.4 our graph consists of inter-edges and intra-edges.
Intra-edges connect the nodes on one ray to the second "nearest" ongoing nodes on a neighbour ray. For this edges we can easily use the combined weight presented in section 2.5:

$$w(e) = \left(w^{dist}(e)\right)^k \cdot \left(w^{op}(e)\right)^j . \tag{30}$$

Inter-edges connect the nodes on the same ray. This makes the usage of the combined weight impossible, since we do not want that nodes on one ray influence one another. As already mentioned in section 2.4, we can assign the weight

$$w(e_n) = 1 - w_n^{op} \tag{31}$$

to the n-th edge e_n but this could lead to an unwanted cut. Imagine a ray $r_{x,y}$ with many detected features that is neighbouring to a ray $r_{x+1,y}$ with only a few detected features, inevitably there will be nodes on ray $r_{x,y}$ with only inter-edges going out from them, on other positions there will be nodes with the inter-edge and maybe several outgoing intra-edges. Clearly the single edges are preferred by the min cut algorithm if they are not uncommonly heavy, see Figure 70.
To counteract the preference we have to increase the weights of intra-ray edges when the corresponding node have none or a few neighbours. During the work I found out that, the simple weight adjustment

$$w(e_n) = (1 - w_n^{op}) \cdot \frac{N_{pos}}{\text{number of neighbours of n-th node} + 1} \tag{32}$$

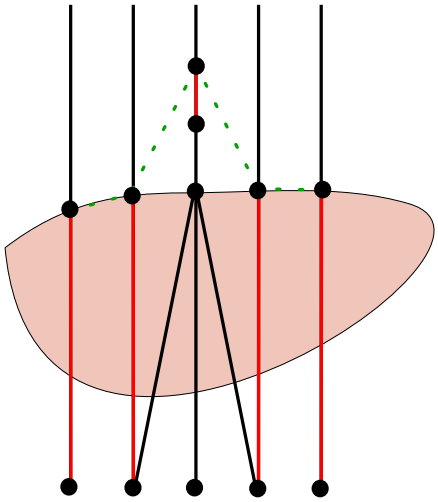

Figure 70: The min-cut would prefer to cut single edges instead of multiple edges, this could lead to unwanted cuts.

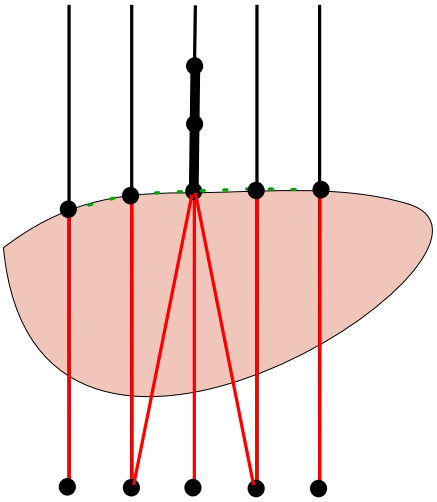

Figure 71: Adjusting the weights of the edges helps to remove the preference of single edges.

is a reliable choice for an intra-ray weight. Hereby N_{pos} denotes the number of beneighboured rays for the considered ray. Note that the formula must be further adjusted if the distance weight is not normalized beforehand.
Now we know how to construct the graph, we know how to compute the weights for the graph and how to assign weights to the edges of the graph. In the next chapter we will investigate how robust the algorithm *surfseek* is, when it is used on noisy data-sets. First we will clarify what noise is and what kind of noise is interesting to us.
Before we come to the noise we provide a few results with the γ- and β-criterion on the same dataset for demonstration purposes. We use the previous skull data-set with a different transfer function, the reader can see that the surface is not that clearly to determine, see Figure 72. We will focus on the jawbone in this case, note that the foggy area in front of the bone and the different bone tissue properties make it harder to determine the surface clearly. Especially the red area in the middle of the jawbone is not clearly to assign. We will use the γ- and β-criterion on this dataset and we will show the results of the direct pick and the results from *surfseek*. Using a direct picking on this dataset leads to terrible surfaces, because the foggy property of this data-set leads to multiple feature detection, where the opacity contribution of each single feature can not be predicted. Using the direct pick with the γ-criterion leads to surfaces as in Figures 73 and 74 using the β-criterion leads to no better surfaces, see Figures 75 and 76.
In this pictures the reader can also observe how transparent the bone tissue in the red zone is. This is especially good to see in Figure 73 and 75, because here the reader can clearly see the detected surface behind the bone tissue.
Now applying the *sufssek* algorithm on this data-set yields the following results, see

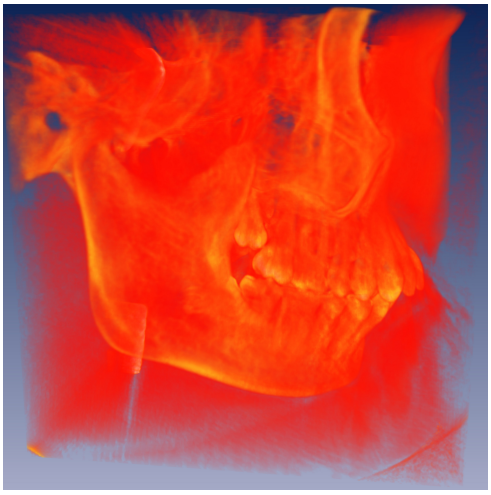

Figure 72: The skull data-set with a different transfer function. The jawbone is perceived as a whole feature, but the foggy area in front of it and the different bone tissue properties make it hard to register the surface.

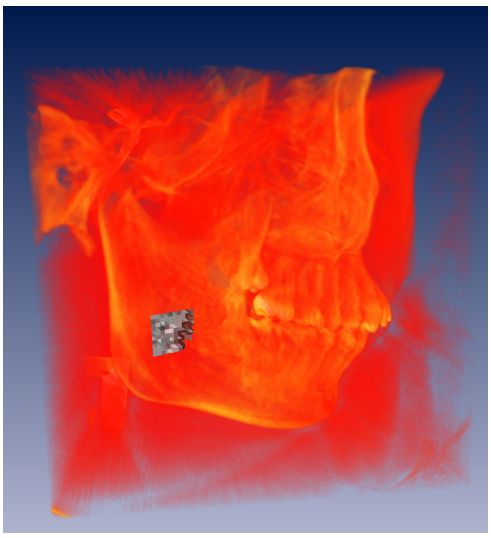

Figure 73: A direct pick with the γ-criterion on a foggy skull data-set.

Figures 77 and 78 for the γ-criterion and Figures 79, 80 and 81 for the β-criterion. In this example both of the criteria give good results, thus both criteria are justified. However especially with this experimental data-set I had the feeling that *surfseek* with the β-criterion yields a surface that is smoother and closer to the feature compared to the surface that is computed with the γ-criterion.

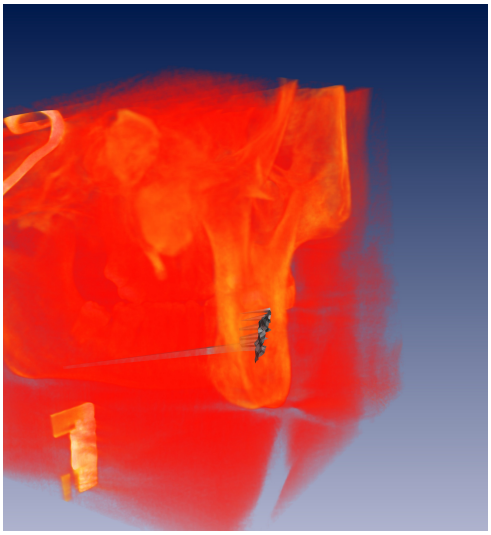

Figure 74: A surface computed with a direct pick with the γ-criterion on a foggy skull data-set shown from the side.

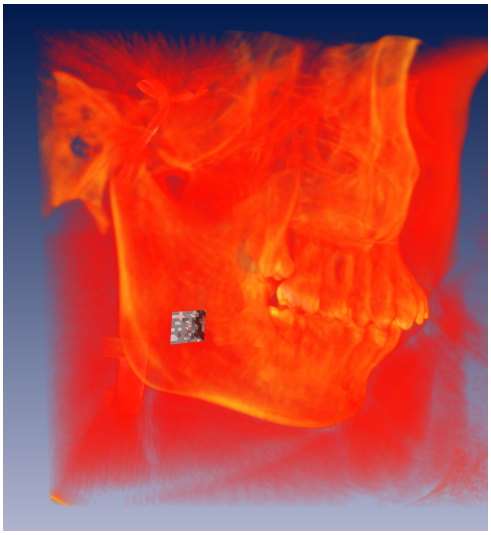

Figure 75: A direct pick with the β-criterion on a foggy skull data-set.

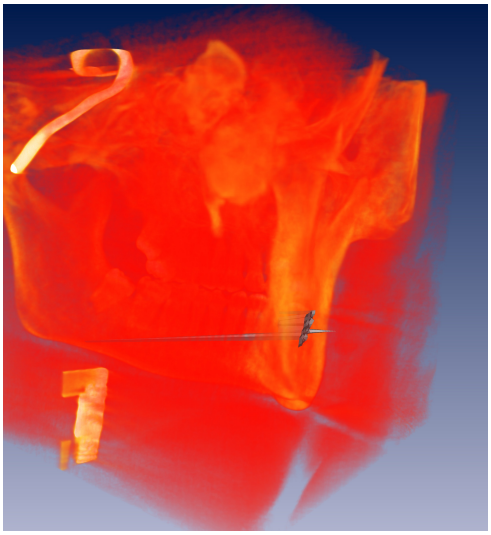

Figure 76: A direct pick with the β-criterion on a foggy skull data-set shown from the side.

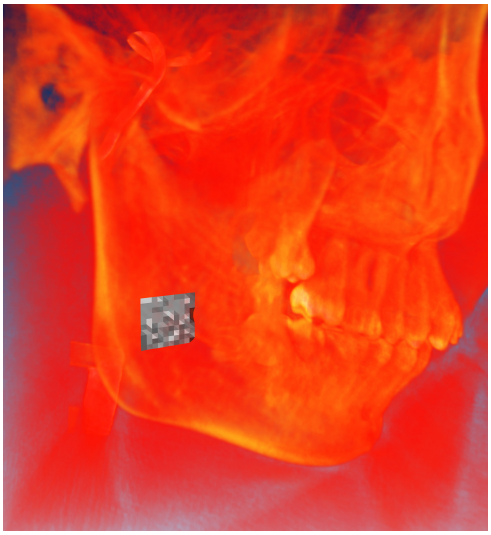

Figure 77: A surface computed using *surf-seek* with the γ-criterion.

Figure 78: A surface computed using *surfseek* with the γ-criterion shown from the side.

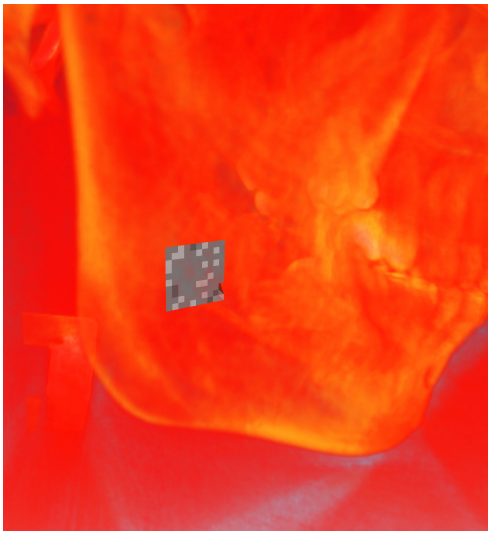

Figure 79: A surface computed using *surfseek* with the β-criterion.

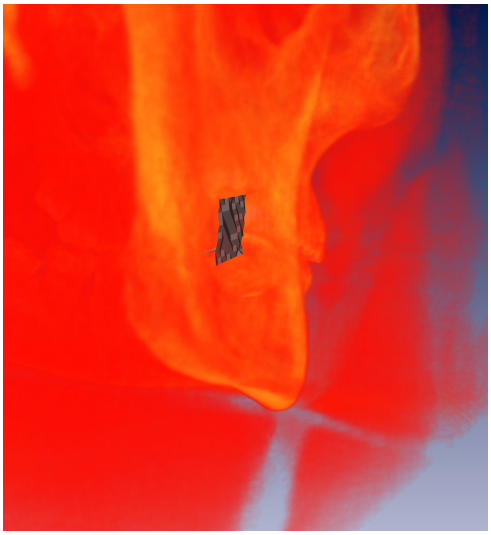

Figure 80: A surface computed using *surfseek* with the β-criterion shown from the side.

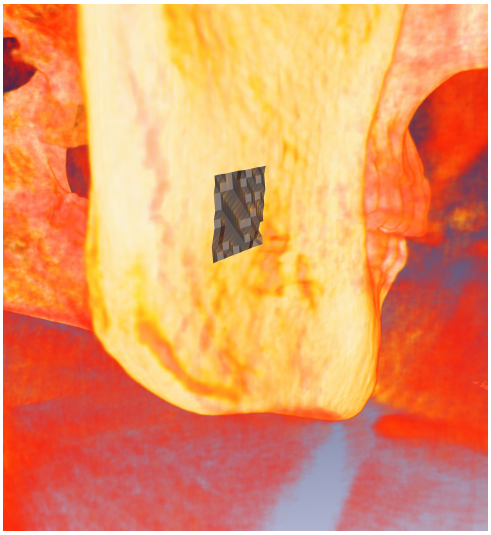

Figure 81: The same surface as in Figure 80 with a changed transfer function in order to show the surface and the jawbone clearer.

3 Noise in the data-set

We already mentioned noise before in this work and it is intuitively clear what is meant by noise, namely errors in the data-set. Since the algorithm *surfseek* is inspired but not limited on medical data, more specifically CT-scans, we will concentrate on possible errors in the data resulting from those scans.
When talking about errors in a CT-scan it makes sense to distinguish between noise and artefacts. With noise we mean errors during the measurement or during the later data processing. Artefacts are not real errors but unlucky or unwanted constellations during the measurement, such as movement of the patient, overlapping tissue or implants. We will discuss both artefacts and noise in detail. For more detailed information I refer to [L], [KS] or [SO] where the following information is derived from.

3.1 Artefacts in CT-scans

As already mentioned artefacts are unlucky circumstances during a CT-scan, the most common artefacts are:

1. Movement of the patient
2. Failure of the measuring electronics
3. Metal-implants
4. Measuring limits transgression
5. Partial-volume averaging
6. Artefacts through beam hardening
7. Artefacts through scattered beams

Many of those artefacts are intuitively understandable, however in the following we will discuss each of the artefacts in detail.

Movement of the patient
The movement of the patient is the most intuitively accessible artefact. Simplified a CT-scan is made in multiple slices that occur successively in time. If the patient moves or takes big breaths during the scan it distorts the data. In the most cases it remains in the distortion of the data. In some extreme cases, especially when the movement occurs during the measurement of one slice, those artefacts can lead to interference structures in the slices, see Figure 82.

Failure of the measuring electronics
If the measuring components fails it leads to missing or false measured data. See Figure

83. Such artefacts are quite common because they can be caused by small dust particles. Furthermore some computer components are very sensitive to temperature and humidity, which can cause further failure of the measuring equipment.

Metal-implants
Metal-implants lead to a glow in the data, because of the different tissue properties. Implants that have exceptionally high or low attenuation can create streaking artefacts by forcing the detectors to operate in a non-linear response region. Figure 84 demonstrates a star pattern caused by a high-density implant.

Measuring limits transgression
It can happen that the patient or the object, that is to be scanned, exceeds the measuring area. In this case the object is "cut off" where it leaves the measuring field. In Figure 85 the body is "cut off" on both sides.

Partial-volume averaging
When tissues of widely different absorption occupy the same voxel, a volume average is computed for such voxels, leading to the partial-volume error. Such artefacts are very common in scans of the liver and pancreas or the scan of the head, where the bone tissue changes the orientation from perpendicular to oblique.

Artefacts through beam hardening
Beam-hardening artefacts result from the preferential absorption of low energy photons from the beam. The effect is more pronounced in areas of large attenuation, such as bone tissue. Those artefact can bee seen as a shadow beneath ribs or increased shadows in the skull. Modern reconstruction algorithms can remove those artefacts. Figure 86 illustrates the *Hounsefield Beam*, which is a technical term for the beam hardening.

Artefacts through scattered beams
Every beam that is shot through an object is slightly scattered. Depending on the alignment of the features inside the object this can lead to miscalculations when the body is reconstructed. This is illustrated in Figure 87, in this figure the reader can observe the primal intensity of the beams shown nearer to the object, and the low scatter intensity further away and nearer to the zero axis.
There are a few other different artefacts that can occur during a CT-scan but those artefacts listed here, are the best known and the most common to occur during a scan.

3.2 Noise during a CT-scan

When we talk about noise in a CT-scan we can group the noise in three main families:

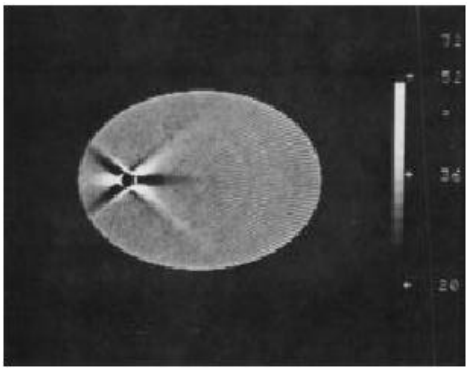

Figure 82: Movement during a CT-scan can lead to interference structures in the data. This figure is taken from [L].

Figure 83: If a measuring component fails it can lead to areas with missing data, in this case one beam on the right side could not be detected. This figure is taken from [L].

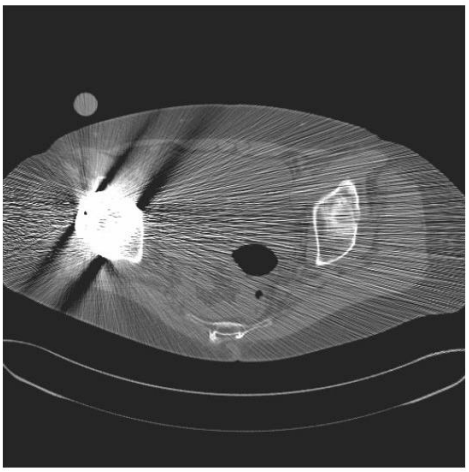

Figure 84: Implants that have exceptionally high or low attenuation can create streaking artefacts by forcing the detectors to operate in a non-linear response region. This figure is taken from [L].

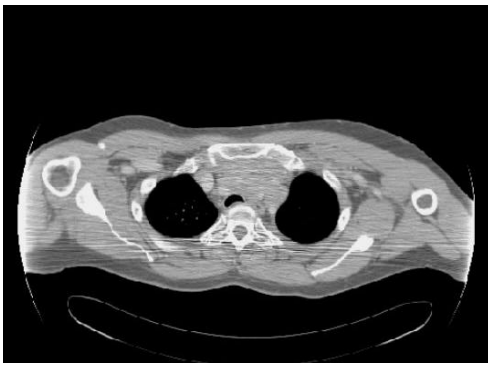

Figure 85: Sometimes the body or the measured object exceeds the measuring area at some point. Here the body is "cut off" on the right and left side. This figure is taken from [L].

Figure 86: Very strong absorbing tissue can lead to a *Hounsefield Beam*, the darker area shown by the arrows. This figure is taken from [L].

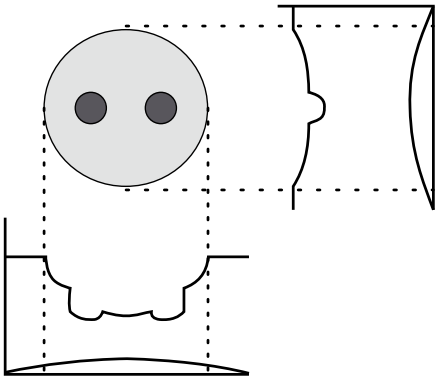

Figure 87: Scattered beams can lead to misinterpretations of the data. In this picture the both tubes could be recognized as one feature because of the highly scattered beam.

1. Quantum noise
2. Electronic noise
3. Pixel-noise

We will introduce each of the families separately.

Quantum noise
The energy in x-radiation is transmitted in the form of individual chunks of energy called quanta. Thus the x-ray detector detects a finite number of x-ray quanta. The number of detected quanta varies from one measurement to an other, not necessary because of errors in the detection apparatus, but because of statistical fluctuations that arise naturally in the counting process. Those fluctuations are inevitable, but the more quanta are detected the more the relative accuracy of each measurement improves.
Clearly quantum noise represents a fundamental limitation for a CT-scan process. The only way to reduce the quantum noise would be to increase the number of detected x-ray quanta. This can be done by increasing the number of emitted x-rays but this also means that the patient is exposed to more x-rays.

Electronic noise
In processing electric signals, electronic circuits inevitably add some noise to the signals. The difficulties of electronic noise suppression are up to the fact, that for some types of x-ray detectors, the electronic signals are very small and thus especially vulnerable to small errors. Furthermore the electronic components underlie certain unreliability when the environmental influences, such as humidity or temperature change, thus bringing

more or less noise in the data signals.

Pixel-noise
Those errors arise from the limited number of bits used to represent the measured values. Often pixel-noise is called round-off noise, because it also occurs when the data must be processed. For example, the product of two numbers must be round off to the least significant bit used in the computers representation for the number.
Normally round-off errors can be kept at an insignificant level, either through clever programming or by the use of a computer with enough bits per number.
When dealing with the display stage the name pixel-noise becomes clearer. Since CT-display units can display only a fixed limited number of discrete brightness levels, it can limit the observers ability to interpret the data.
An other limitation is the limited number of the reconstructed voxels. This can lead to edgy images, since it is not possible to display a ball with voxels without cutting it.

In the next section we will discuss what errors we will simulate and how it should be done.

3.3 Noise modelling

There are a few good works regarding the simulation of the noise during a CT-scan, see Tu et al. [TSC] for more information. However all of the simulations simulate the whole CT scanning process. Such a simulation would exceed the extent of this work regarding the time and capacities.
Since there are none attempts to simulate the noise of a CT-scan directly on the scalar 3D data, we have to choose a few of the errors we can simulate.
The errors we can simulate are: artefacts through scattered beams, partial-volume averaging and different noise influences.
In general the quantum noise can be and is simulated through the Gauss noise. The pixel-noise is a noise that is inevitable for all datasets. Since it does not change the position of the selected surface but only it's quality, say turning it in a stair like surface, we will not consider this kind of noise.
A quite interesting noise that can occur in image processing is the salt and pepper noise. This kind of noise is usually represented by completely white or black randomly distributed pixels in an image. In our case this means that there are randomly distributed voxels with uncommonly large or low intensity values.
An another noise type that is interesting to consider is the speckle noise. Originally observed in radar images, speckle noise results from random fluctuations in the return signal from on object that is no bigger than a single image-processing element. It increases the mean grey level of the local area. Although speckle noise is not existing in a CT-scan, it is an interesting noise to look at, because it can change the perception of the image greatly.

We will now shortly explain how each of the errors will be simulated and show the results afterwards.

Scattered beams
Scattered beams are usually simulated through Gaussian smoothing. The Gaussian smoothing or blur is done by convolving each point of the input data with the Gaussian function, also known as normal distribution from stochastic theory. The one dimensional Gaussian function is defined through

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2}{2\sigma^2}}. \tag{33}$$

In our case the three dimensional Gauss function is the product of three such one dimensional Gaussian functions, one in each dimension:

$$G(x, y, z) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2 + y^2 + z^2}{2\sigma^2}}. \tag{34}$$

Hereby x,y and z denote the distance from the origin in the corresponding dimension, the σ represents the variance. In [TSC] Tu et al. suggest a variance of $\sigma = 2.0$ mm. Note that we assume the same variance in each dimension.

Partial volume averaging
It is very easy to create datasets with overlapping features, for example we can use a small ball that overlaps with an ellipsoidal.
Since partial volume averaging had no influence on the algorithm at all, we will not present the data-set with overlapping features. As they don't provide an insight increase.

Quantum noise
As already mentioned the quantum noise is usually simulated through the Gaussian noise, or more specific the additive white Gaussian noise.
Additive means that we simply add noise values to the data-set. The characteristics of white noise is the constant energy density over the frequency. For us this means that the overall magnitude of the noise can be chosen the same for all voxels. The next term Gaussian is already known, and refers to shape of the probability density curve from equation 33. Gaussian noise always has a mean of zero, which means that we can add as well positive as negative values. In the noise simulation approaches the variance σ is choosen as the square root of the x-ray influence I, which is defined as the number of photons received in a unit cross-sectional area [BSL].
Since we can not simulate the whole CT-scan process, we have to use a different σ. In [TSC] Tu et al. derive a variation from the experimental measurements of $\sigma = 4.0$ mm. We will use this variation for the upcoming simulations.

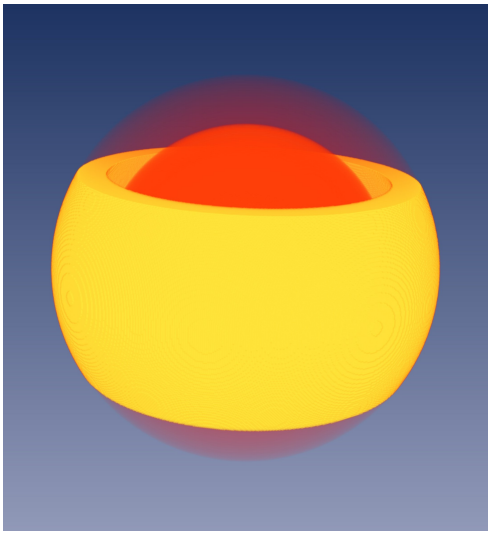

Figure 88: A data-set that represents the human head, with a ball with low density (brain) surrounded with a ball of lower density (liquor) that is enclosed in a strip of high density (skull).

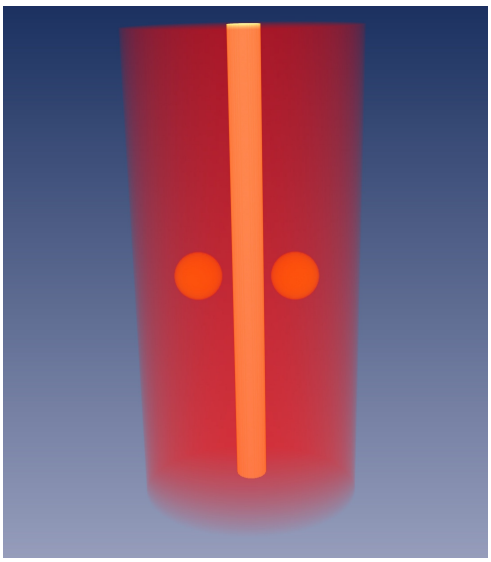

Figure 89: A data-set that represents the human torso, with a thin cylinder with a high density for the spinal column, two balls with lower density for the kidneys and a big cylinder that represents the surrounding tissue.

Salt and Pepper noise
Salt and pepper noise or "impulsive" noise is usually caused by analog-to-digital converter errors or bit errors in transmission. This kind of noise can be simulated by choosing a certain amount of pixels or voxels randomly out of the data-set and "color" those black and white.

Speckle noise
As mentioned before speckle noise can be encountered in conventional radar or synthetic aperture radar images. But when dealing with laser beams, one meet speckle noise again when looking at the intensity pattern of laser dots. Those patters are a result of the interference of many waves of the same frequency.
Speckle noise can be modelled by multiplying random values pixel- or voxel-wise with the data-set. For our model we will use a random value generator in the region of $[0, 1]$.

3.4 Quantum Noise and Blurring Analysis

For the first simulation series we will use two synthetic data-sets, a synthetic skull and synthetic torso see Figures 88 and 89. On those data-sets we will apply quantum noise and see how the algorithm handles the noise. Then the data-sets will be blurred and we will examine the picking algorithm on the blurred data-sets.

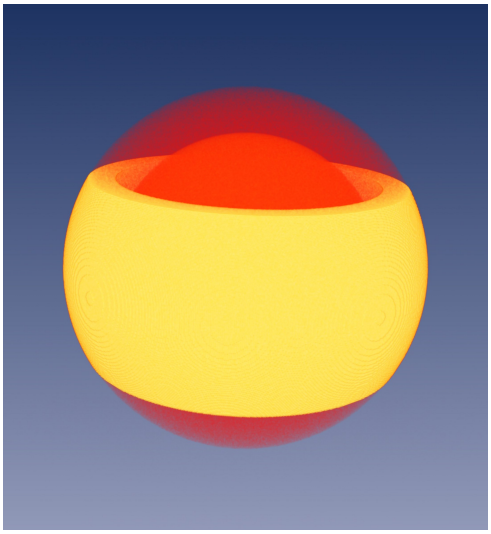

Figure 90: The synthetic skull with quantum noise.

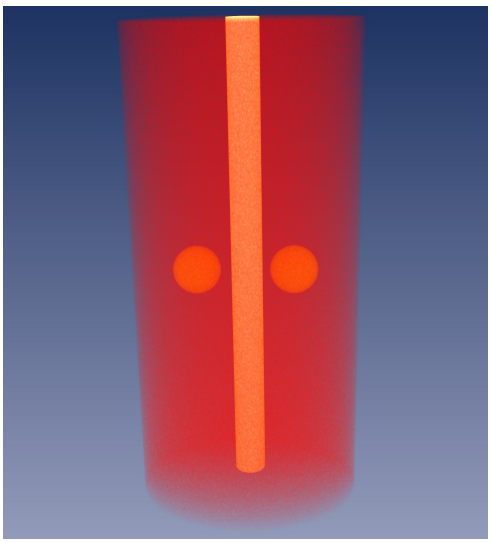

Figure 91: The synthetic torso with quantum noise.

When we use *surfseek* with the γ- or the β-criterion on the pure synthetic datasets both of the algorithms detect the surfaces correctly, which is not surprising, since there are no possible sources of error.
What happens when we use the algorithms on the noisy data-sets? For the noise simulation we used the quantum noise on each voxel with the maximal amplitude of 1. In Figure 90 for an illustration of the synthetic skull with quantum noise and Figure 91 for the illustration of the synthetic torso with quantum noise.
Let's see what direct picking methods will result in. When we choose direct picking with the β-criterion the resulting surface has several false detected points, see Figure 92. When we use direct picking with the γ-criterion the resulting surface is worse, see figure 93. This is caused through the already mentioned division of the features when the data-set is noisy.
When we use *surfseek* to compute the surfaces, the resulting surfaces are much smoother. With the β-criterion we obtain a surface as in Figure 94. We can see that the surface does not describe the surface of the ball perfectly, but there are no jumps or misfits in the surface. Using the γ-criterion we obtain a surface as in 95, here the noise crates some unwanted point in the surface. This is the already known behaviour described in section 1.2.4.
What happens when we blur the data? Will the resulting images be smoother, because the error is blurred too or will the blurring cause another error source?
For the blurring we used the already mentioned variation of 2.0 mm. We assumed the skull to have a diameter of 15 cm. The torso was assumed to be 35 cm in width.
After the blurring the data-sets look like in Figures 96 and 97.
Applying *surfseek* with the β-criterion on the inner ball we receive a surface as in Figure 98. Using the γ-criterion the surface gets some misfits, see Figure 99. Again this is due

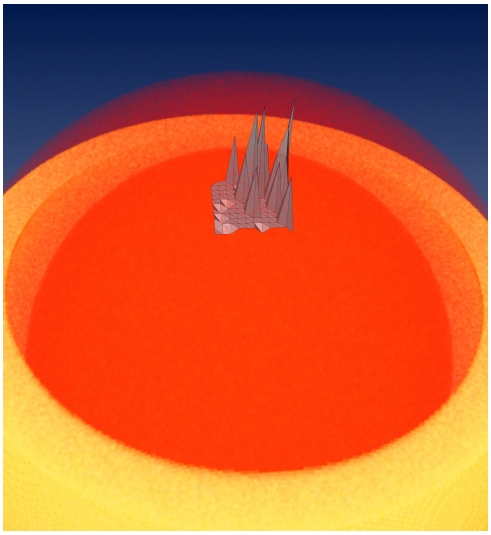

Figure 92: A surface computed with direct picking using the β-criterion on the synthetic data-set skull with quantum noise.

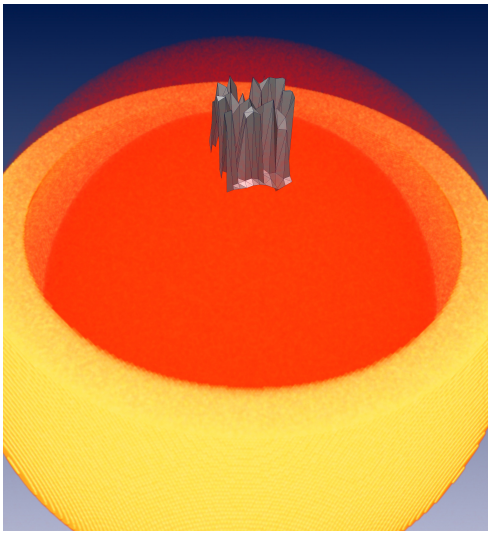

Figure 93: A surface computed with direct picking using the γ-criterion on the synthetic data-set skull with quantum noise.

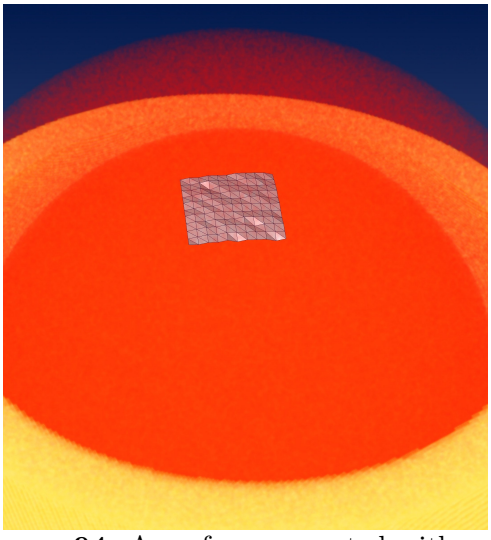

Figure 94: A surface computed with *surf-seek* using the β-criterion on the synthetic data-set skull with quantum noise.

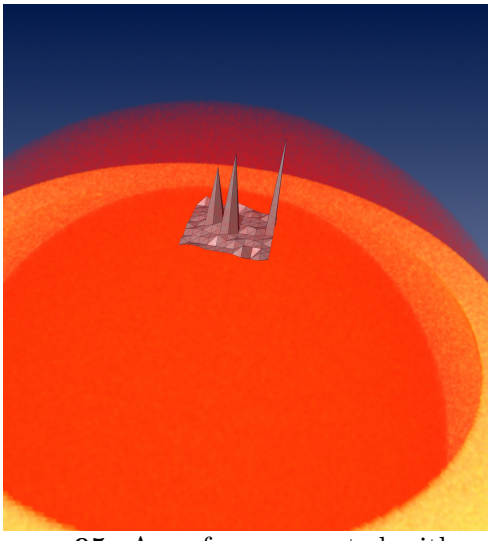

Figure 95: A surface computed with *surf-seek* using the γ-criterion on the synthetic data-set skull with quantum noise.

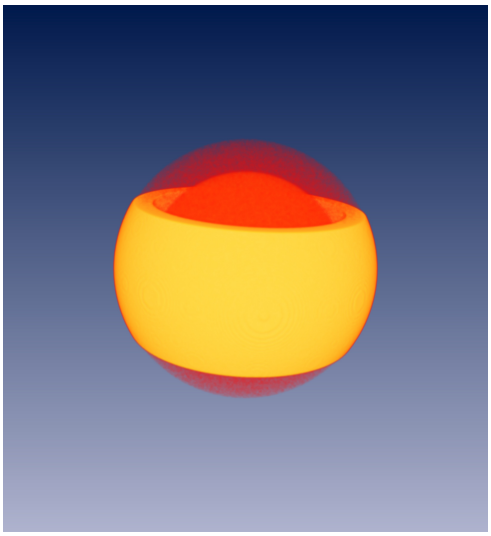

Figure 96: The synthetic skull with quantum noise and Gaussian blurring.

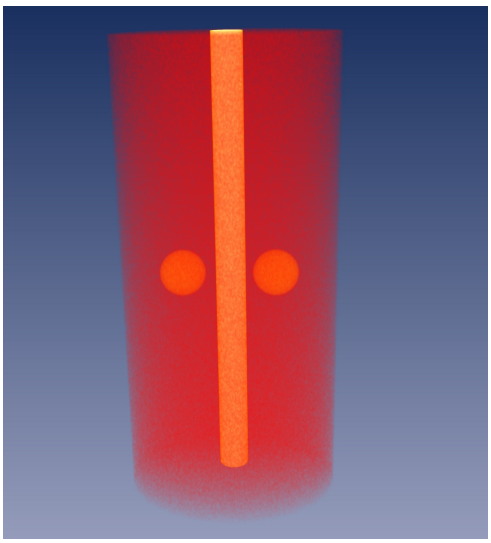

Figure 97: The synthetic torso with quantum noise and Gaussian blurring.

to the division of the features.
An interesting behaviour of the algorithm can be observed, when we try to project a surface patch on the outer belt, that represents the bone tissue. Here the reader can observe the difference between a local and a global choice of ϵ for the β-criterion. When we use a global ϵ, then only the bone tissue can be detected. Because the bone tissue has such a fast opacity acceleration, the other tissue is ignored when we use $\epsilon = 0.3$. The resulting surface can be seen in Figure 100. When we choose to use a local ϵ, then the features with lower densities are not ignored any more, but because of the proportionally high noise in the features with lower densities the algorithm detects unwanted feature borders, see Figure 101. Using the γ-criterion yields in a better result concerning the inner ball, see Figure 102.
What is about the synthetic torso? What makes the synthetic torso different from the synthetic skull is the thick non-zero feature surrounding the "kidney" and the "spine". This means, that the rays have a longer way through the tissue and thus collect more errors on that way. In this data-set we can explicitly observe the difference between a direct pick and the surface resulting from *surfseek*.
When the data-set has no noise, then the surfaces computed with the direct pick and *surfseek* with both the γ- and β-criterion are exactly the same. Therefore we will begin directly with the noisy dataset seen in Figure 91. We will present the result of the direct pick first. Since the quality of the surface did not change between the different patch projections on the "kidney" and the "spine", we will present the result of the kidney exclusively. The surface computed with the direct pick using the γ criterion can be seen in Figure 103. The surface computed with the β-criterion can be seen in Figure 104. In both cases the main part of the feature was detected correctly, only a few rays detected a non existing feature with higher opacity contribution.

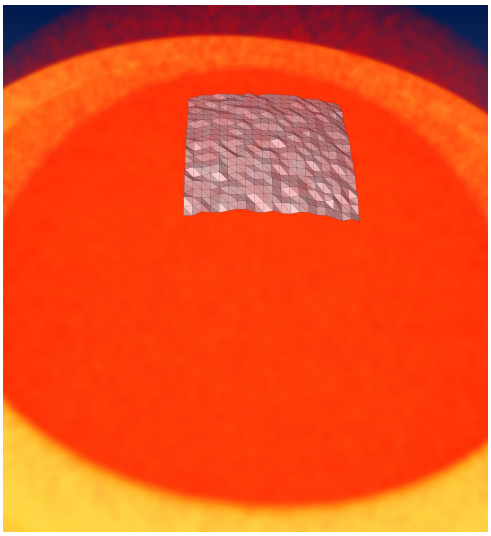

Figure 98: A surface computed with *surf-seek* using the β-criterion on the synthetic data-set skull with quantum noise and Gaussian blurring.

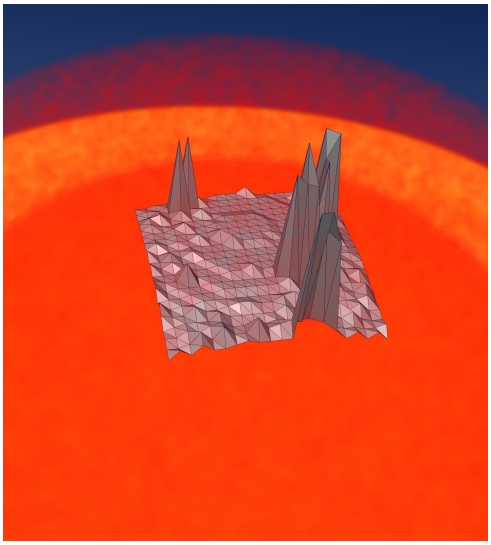

Figure 99: A surface computed with *surf-seek* using the γ-criterion on the synthetic data-set skull with quantum noise and Gaussian blurring.

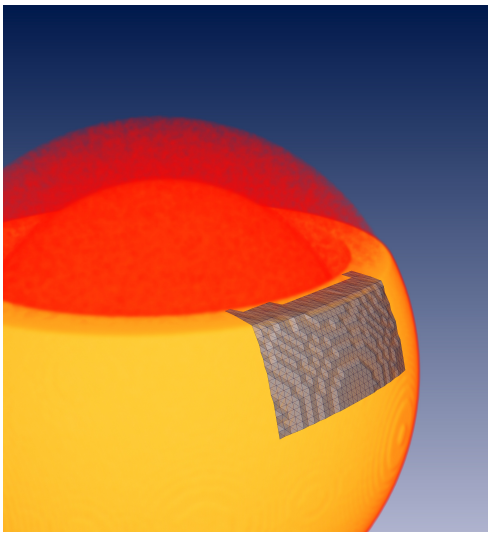

Figure 100: Using a global ϵ can cause to ignore the features with lower density.

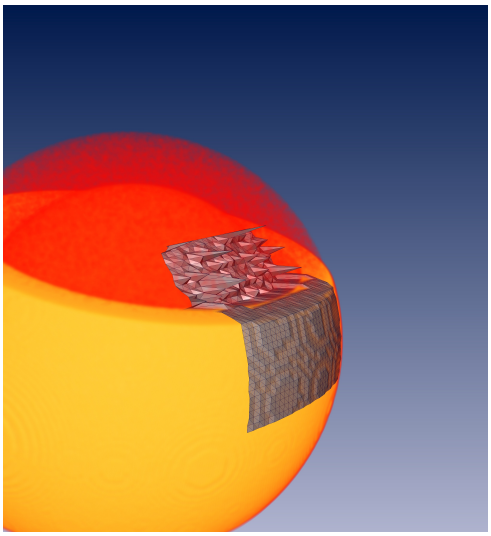

Figure 101: Using a local ϵ can lead to jaggy surfaces when the noise is big enough.

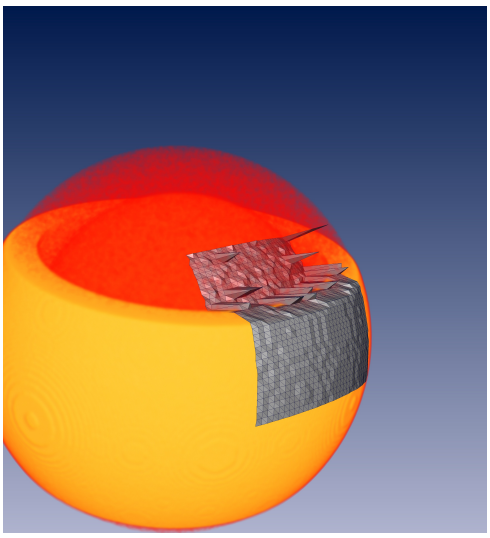

Figure 102: A surface computed with the γ-criterion. The feature borders are better detected than with the β-criterion.

Therefore it is not surprising that, both of the criteria lead to a good result when using *surfseek*, see Figures 105 and 106.
Now one could assume, that when the data-set is blurred then the surfaces computed with the direct pick should be smoother, because the errors are blurred and therefore the fluctuation averaged. Especial the γ-criterion should benefit from the blurring as it should minimize the division of the features. Surprisingly this is not the case. The Gauss smoothing lead to a data-set where the result of direct picking are much worse than those from only a noisy data-set. See Figure 107 for the surface computed with the γ-criterion. The surface computed with the β-criterion is not that different from the one from the noisy data-set. The reader can observe the surface computed with the β-criterion in Figure 108. Note that the noise in the images may appear to change from one image to another, this is not the case. The different appearance has its origin in the different angles during the snapshot, furthermore some snapshots were displayed a little bit blurry for a faster computation.
How do the surfaces look when we use *surfseek*? We can already expect that the β-criterion should provide some good results, since the surface of the direct pick had only a few "false" points. But surprisingly enough the surface computed with the γ-criterion is equivalently good. The reader can observe both surfaces in Figures 109 and 110. Figure 109 shows the surface computed with the γ-criterion and Figure 110 the surface computed with β-criterion respectively.

Conclusion

So far we can observe that both of the picking criteria provide good results. The results of direct picking and *surfseek* are the same when we deal with synthetic data-sets with-

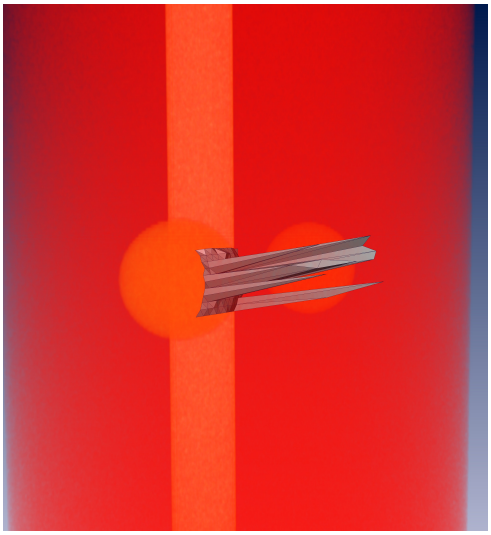

Figure 103: A direct pick using the γ-criterion on the synthetic torso data-set with quantum noise.

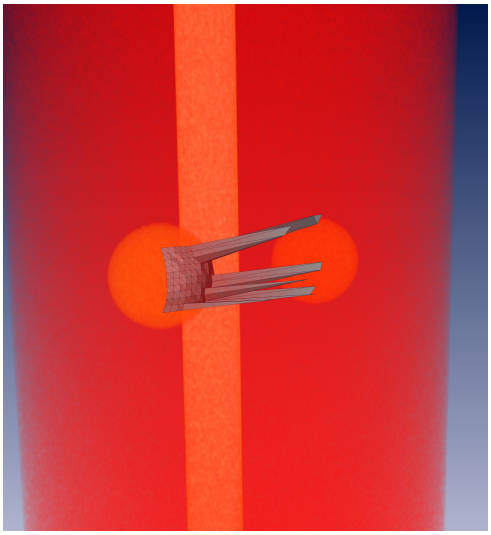

Figure 104: A direct pick using the β-criterion on the synthetic torso data-set with quantum noise.

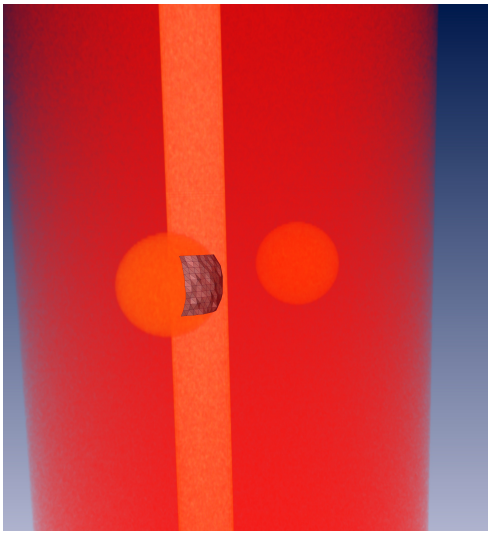

Figure 105: A surface computed with *surf-seek* using the γ-criterion on the synthetic torso data-set with quantum noise.

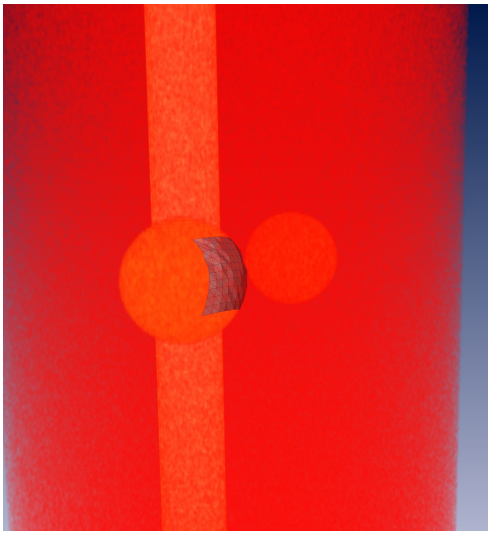

Figure 106: A surface computed with *surf-seek* using the β-criterion on the synthetic torso data-set with quantum noise.

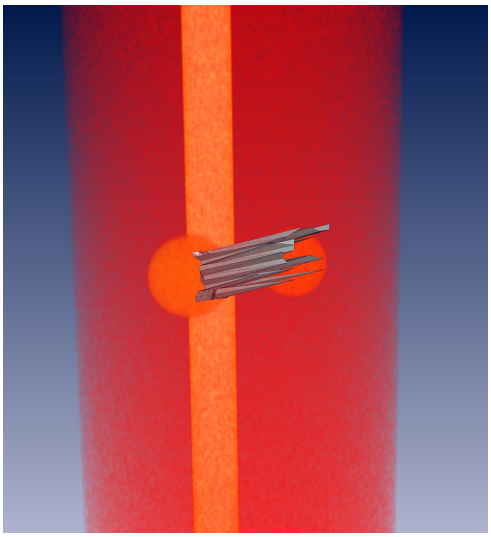

Figure 107: A surface computed with direct picking using the γ-criterion on the synthetic torso data-set with quantum noise and Gaussian blurring.

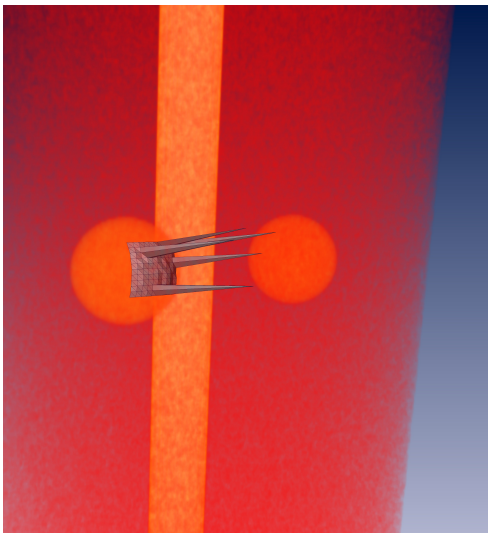

Figure 108: A surface computed with direct picking using the β-criterion on the synthetic torso data-set with quantum noise and Gaussian blurring.

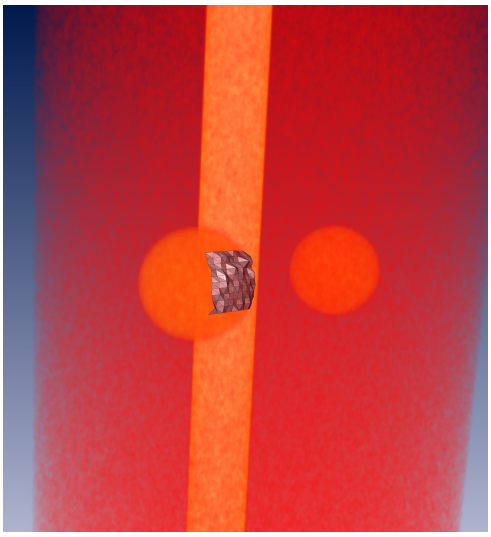

Figure 109: A surface computed with *surfseek* using the γ-criterion on the synthetic torso data-set with quantum noise and Gaussian blurring.

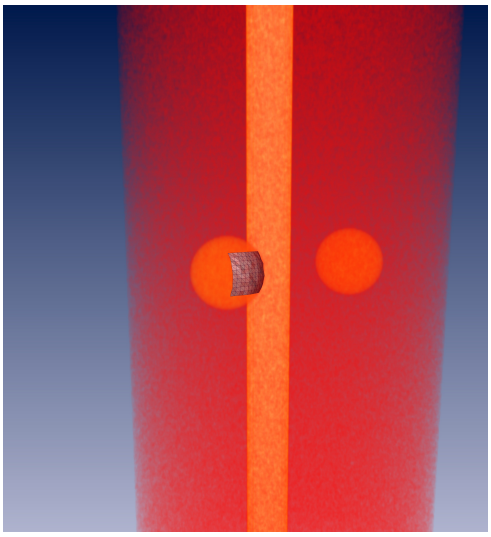

Figure 110: A surface computed with *surfseek* using the β-criterion on the synthetic torso data-set with quantum noise and Gaussian blurring.

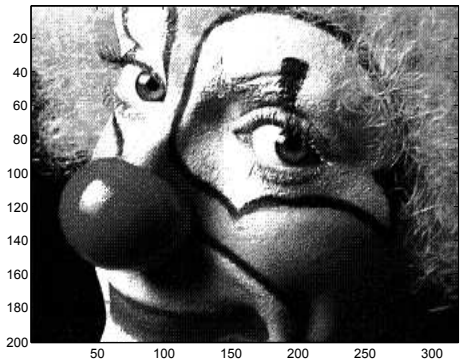

Figure 111: An image of a clown from the Matlab image library.

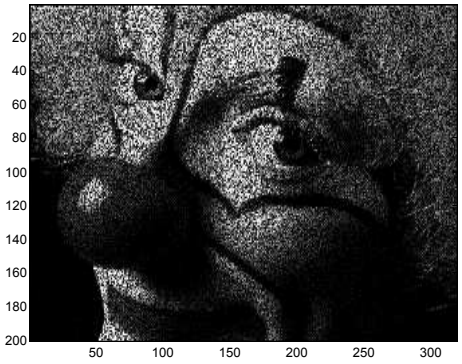

Figure 112: The clown image from the Matlab library with speckle noise.

out noise. As soon as noise is added to the data-set, the surfaces from direct pick get worse, because the quantum noise creates false feature boundaries. This effect is even stronger when the noise is blurred. When the data-set is noisy and blurred, then the surfaces computed with *surfseek* differ, depending on the picking criterion. The surfaces computed with the γ-criterion appear to be not that smooth as the ones computed with the β-criterion, see Figures 109 and 110. For each of the criteria the algorithm shows to be stable against quantum noise and blurring.

3.5 Speckle Noise Analysis

Speckle noise is by far more aggressive than quantum noise, because we don't add the noise but multiply with it, the changes on the data-set are much more crucial. In a 2D Image speckle noise can lead to misinterpretations. Images contaminated with speckle noise look like they have holes. The reader can examine a 2D picture that is contaminated with speckle noise in Figure 112, the original Image can be observed in Figure 111. The details like hair or iris are not recognisable in the noisy image.
In a 3D data-set we can observe the same behaviour. In Figure 113 the reader can observe the synthetic skull data-set, that was contaminated with speckle noise. The reader should be able to perceive the boundaries of the single features, but he should also see that the features have a sponge like structure. This means that the perceived boundary is not the "real" boundary but an interpolated result from our mind.
How does *surfseek* react on speckle noise? Sadly *surfssek* fails to detect a proper surface on such a noisy data-set. When we look at the accumulated opacity and its derivative this is not surprising. Since the original synthetic-skull data-set is very large and thus confusing, we provide an example of a smaller similar data-set contaminated with speckle noise for a better insight. In Figure 114 the reader can examine a the α^{acc}-, β^{acc}- and γ^{acc}-functions of a small data-set contaminated with speckle noise. Originally the data-set was build of two balls inside each other the inner ball had the density of 0.2 and the outer bigger ball a density of 0.1.

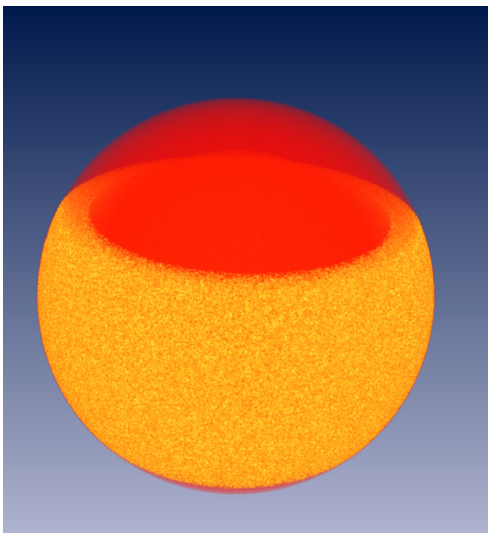

Figure 113: The synthetic skull data-set contaminated with speckle noise.

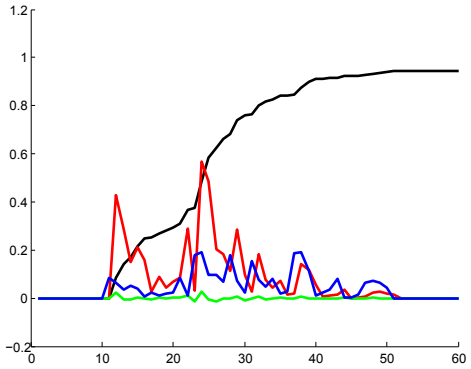

Figure 114: The opacity function (blue), the accumulated opacity (black) the β-function (red) and the γ-function (green) of a two ball data set contaminated with speckle noise.

Since this speckle noise has such a great influence, we can examine the influences of different leverage choices for the length leverage k. When we compare $k = 2$ and $k = 3$ for the β-criterion we can see a great qualitative difference in the surface. Using the γ-criterion the difference was not meaningful, hence we will not present the differences of different choices for the length leverage.
The surfaces that result from *surfseek* can be observed in Figure 115, 116 and 117. Figure 115 shows a surface computed with the β-criterion and a length leverage $k = 2$, Figure 116 shows a surface computed using the β-criterion and length leverage $k = 3$ and Figure 117 shows a surface computed with the γ-criterion.
Compared with the surfaces from a direct pick those are not real improvements, see Figure 118 for the surface computed with direct picking using the β-criterion and Figure 119 for the surface computed with direct picking using the γ-criterion.

Conclusion
We can see that *surfseek* has clear limits, as it does not calculate a satisfactory surface when the data is polluted with speckle noise. Hereby it was inessential if we used the γ- or the β-criterion. The surface that was computed using the β-criterion was smoother but it did not describe the desired surface sufficiently.

3.6 Salt and Pepper Noise Analysis

In the previous section we saw that *surfseek* reaches its limits when the data is contaminated with speckle noise. We can expect similar difficulties when we use *surfseek* on

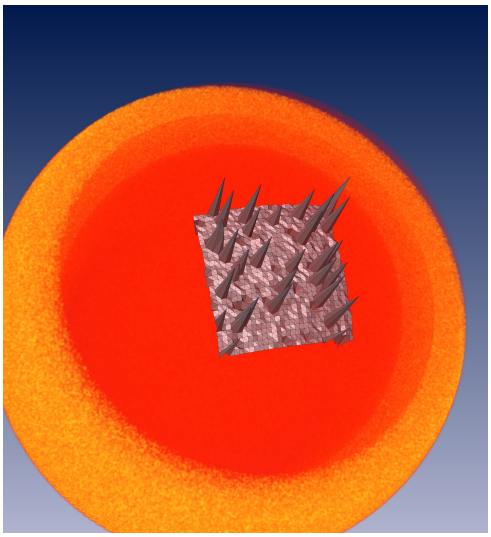

Figure 115: A surface computed with *surf-seek* using the β-criterion with the length leverage $k = 2$ on the synthetic skull data-set contaminated with speckle noise.

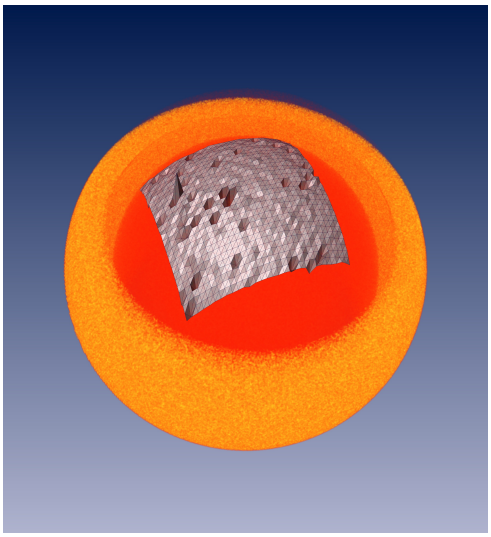

Figure 116: A surface computed with *surf-seek* using the β-criterion with the length leverage $k = 3$ on the synthetic skull data-set contaminated with speckle noise.

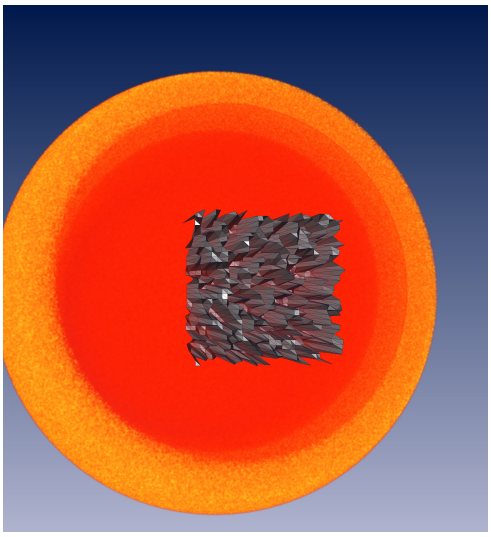

Figure 117: A surface computed with *surf-seek* using the γ-criterion on the synthetic skull data-set contaminated with speckle noise.

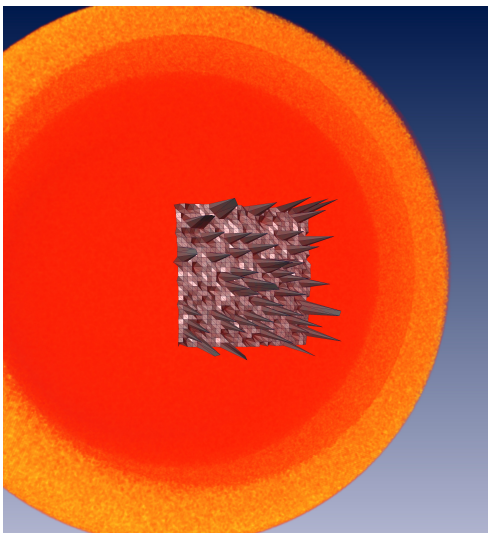

Figure 118: A surface computed with direct picking using the β-criterion on the synthetic skull data-set contaminated with speckle noise.

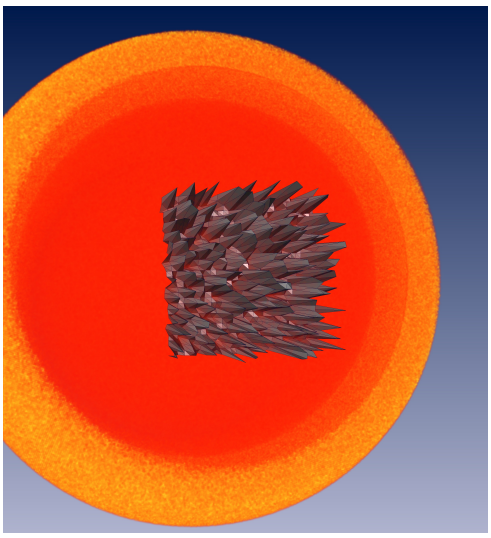

Figure 119: A surface computed with direct picking using the γ-criterion on the synthetic skull data-set contaminated with speckle noise.

data-sets with salt and pepper noise. Salt and pepper noise shows itself as totally or highly opaque voxels in the data. In the Figure 120 the reader can see the synthetic skull dataset with a salt and pepper contamination, where 2 percent of voxels are noisy. In Figure 121 the reader can see the same synthetic skull data set with 5 percent noisy voxels.
For the first Figure we randomly chose 1 percent of the voxels to have very low values and 1 percent of the voxels to have very high values. For the second Figure 2.5 percent respectively.
The difficulty with the salt and pepper noise is, that the opacity accumulation stops if a ray hits a totally or highly opaque voxel. Thus the algorithm can not detect features behind those voxels. In practice this leads to single misfits in the surface. The data-set with a total noise amount of 2 percent, did not lead to misfits, thus we will show the surfaces of the data-set with a total noise amount of 5 percent.
In Figure 122 the reader can observe a surface computed with direct picking and β-criterion, in Figure 123 the reader can see a surface computed using *surfseek* with the β-criterion.
In Figure 124 the reader can discover a surface computed with direct picking using the γ-criterion. In Figure 125 the reader can see a surface computed with *surfseek* using the γ-criterion.
We can observe that the surfaces computed with *surfseek* are smoother than those computed with direct picking, but some misfits are inevitable. Even if we use α-scaling, that is multiplying all opacities with a factor $0 < d_\alpha < 1$, there is no guaranty that we receive a smooth surface, since the noisy voxels are distributed randomly.

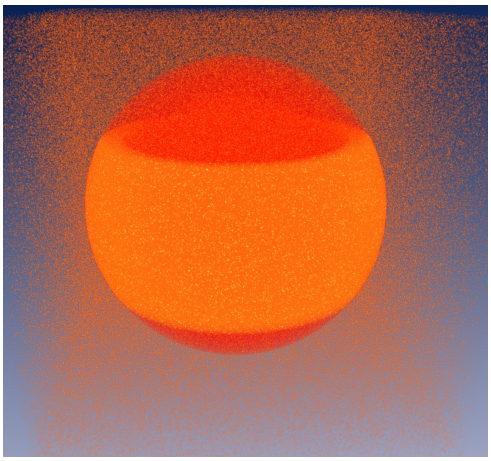

Figure 120: The synthetic skull data set with addition of salt and pepper noise of total contamination of 2 percent of all voxels.

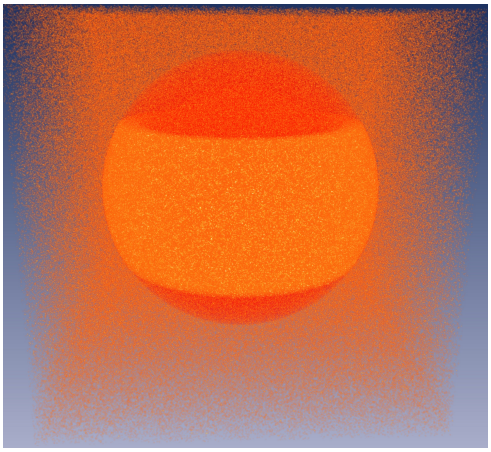

Figure 121: The synthetic skull data set with addition of salt and pepper noise of total contamination of 5 percent of all voxels.

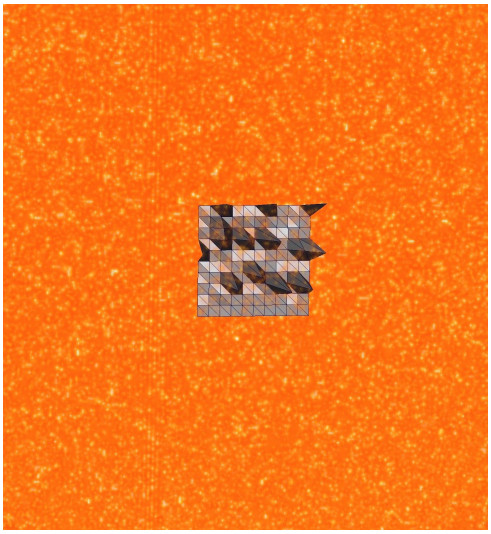

Figure 122: A surface computed using direct picking with the β-criterion on the synthetic skull data set with 5 percent salt and pepper noise.

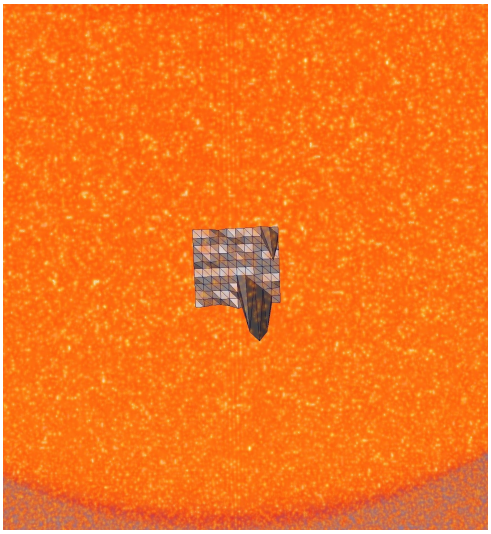

Figure 123: A surface computed using *surfseek* with the β-criterion on the synthetic skull data set with 5 percent salt and pepper noise.

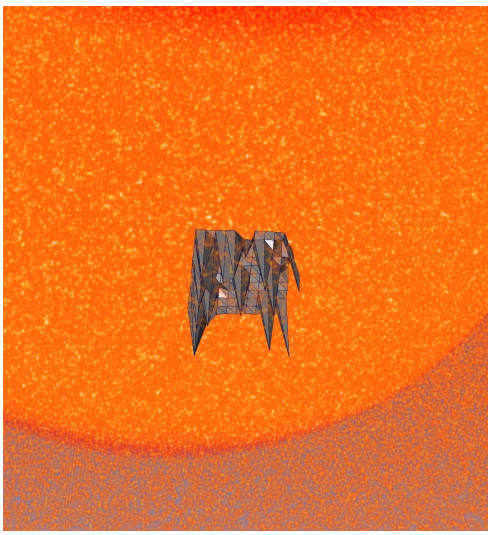

Figure 124: A surface computed using direct picking with the γ-criterion on the synthetic skull data set with 5 percent salt and pepper noise.

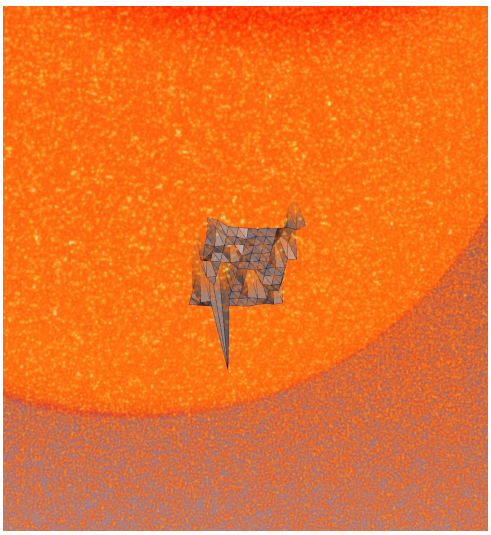

Figure 125: A surface computed using *surfseek* with the γ-criterion on the synthetic skull data set with 5 percent salt and pepper noise.

The most common way to reduce salt and pepper noise is to smooth the image. Again the most used smoothing function is the Gaussian function. When we blur the data-set with 5 percent noisy voxels using the Gaussian function with the variance of 2 mm, we receive a new dataset, that can be observed in Figure 126. In this Figure the reader can observe that the outer ball is fully transparent now, this is due to the low intensity of the ball, that is very sensitive to the noise.
When we try to compute a surface on the synthetic bone tissue we get a result as in Figure 127.
We can observe that the surface has improved, in the sense of reducing the misfits, but it also doesn't describe the original surface properly. The surface, that is computed after the Gaussian smoothing, shows some dents and is not that flat any more.

Conclusion
We have seen that salt and pepper noise represents a high difficulty for the *surfseek* algorithm. Even the Gaussian blurring could not eliminate the drawbacks completely. Although we have to consider that the salt and pepper noise in our example is very extreme as the noise values were extremely high, approximately 4 times higher than the maximal intensity values of the synthetic skull, this example shows that the algorithm is very sensitive to some extreme misfits in the data regardless of the picking criterion.

3.7 Surface quality as function of noise

In This section we will examine the quality of the computed surfaces as a function of noise. We already saw that the direct picking yields in worse results compared to surfaces

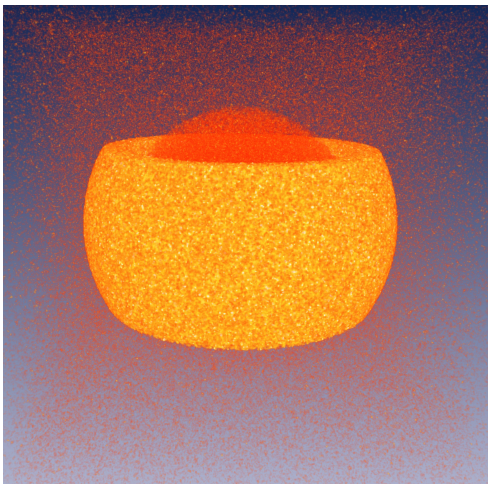

Figure 126: Using Gaussian smoothing on a with salt and pepper noise contaminated data-set, we reduce the noise but also loose visibility of small intensity data.

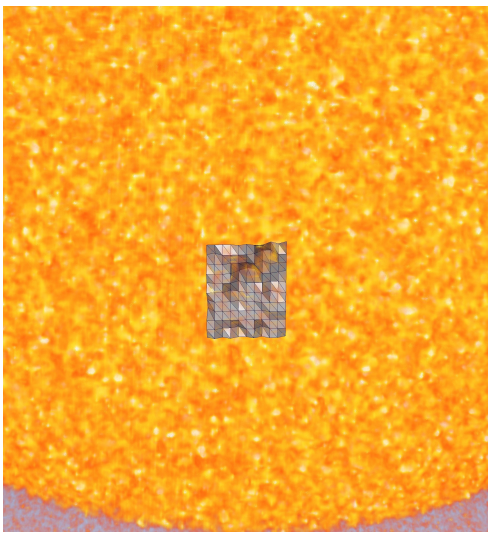

Figure 127: A surface computed using *surfseek* with the *beta*-criterion on the synthetic skull data set with 5 percent salt and pepper noise and Gaussian smoothing.

computed with *surfseek*, hence we will investigate the surfaces computed with *surfseek* only.
In the following we will ad white Gaussian noise to a ball data-set. The ball was created with the formula

$$(R \leq 0.5) \cdot 20. \tag{35}$$

This means that we have a ball with the diameter 1 and the density 20. In each iteration we add white noise to the data-set. We use the standard variation $\sigma = 1$, in each step we increase the noise amplitude, such that the maximal absolute amplitude of the noise corresponds to a specific percent value of the ball density.
The amplitude of the Gaussian function is increased in a way that the absolute maximum corresponds to 10, 15, 20, 25, 30, 35, 40 and 45 percent of the ball density.
For each noise we will apply *surfseek* with the γ- and β-criterion. Furthermore for each criterion we will compute the maximal, mean and median difference to the surface computed without noise.
To provide same conditions for each surface, we cast exactly the same rays on the ball for every computation.
When the data-set has no noise, then both picking criteria provide almost the same results. The result computed with the γ-criterion lie between 1 and 3 voxels in front of the surface, computed with the β-criterion. For the naked eye the difference was not really noticeable. In Figures 128 and 129 the reader can examine the surfaces computed with the β- and γ-criterion respectively. We will call those surfaces reference surfaces.
We will now examine the differences of the surface computed in noisy data-sets to the surfaces in Figures 128 and 129. In Figure 130 the reader can observe the maximal

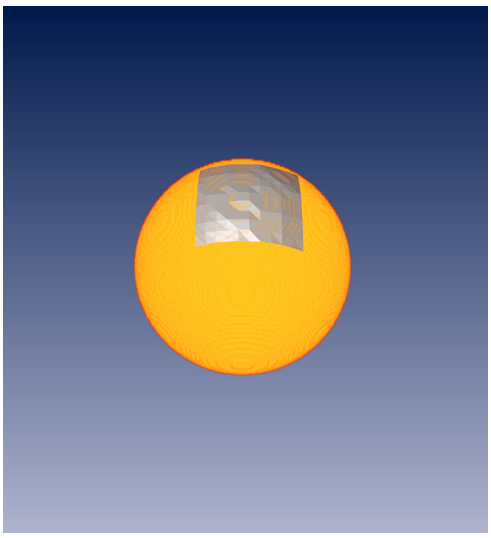

Figure 128: A surface computed using *surf-seek* with the β-criterion on a simple ball without noise.

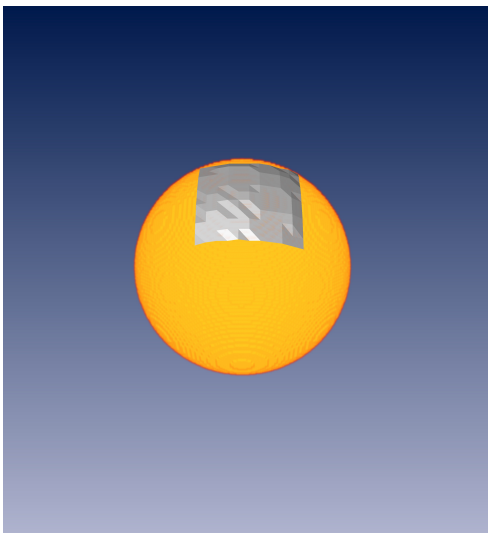

Figure 129: A surface computed using *surf-seek* with the γ-criterion on a simple ball without noise.

difference of the surfaces computed with the β-criterion in noisy data-set compared to the reference surface. We can clearly see that noise with maximal absolute amplitude up to 35 percent of the ball density does not influence the surface greatly. The same behaviour can be observed in Figure 131, in this Figure the mean difference is displayed over the noise. Again we see an increase of inaccuracy at a noise amplitude of 40 percent of the ball density.
The medians of the difference don't show such a behaviour, as the reader can observe in Figure 132. This is due to the fact, that the most surface positions are computed correctly, and only a few misfits impair the surface. Note that the one distinctive value in the middle of the graph corresponds to exactly one voxel size.
Let us take a look at the DVR-image at the noise level of 40 percent. The reader can observe the DVR-image in Figure 133. At such a noise the ball is not clearly visible any more. Using α-scaling, that is multiplying all opacity values α with a real number $d_\alpha \in [0, 1]$, with $d_\alpha = 0.6$, we can make the noise less visible for a better insight on the computed surface, see Figure 134.
Clearly at such nose stage it is impossible for the algorithm to compute the surface identical to the reference surface, since the noise ensures that some parts of the ball are not visible at all. Thus the algorithm has no chance to detect the features lying behind those fully opaque features. Using α-scaling we can make the ball visible again, in this case the algorithm computes the surface without errors again, regardless of the noise magnitude. For example the reader can examine the ball data-set with a maximal absolute noise amplitude of 45 percent of the ball intensity with an α-scaling using $d_\alpha = 0.6$ in Figure 135. We can clearly recognise that the computed surface corresponds the ball-boundary very good.

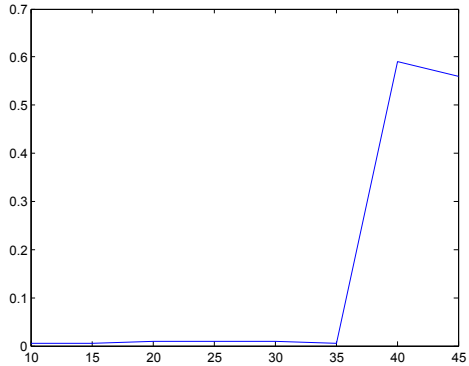

Figure 130: Maximum difference of the surfaces computed with the β-criterion in noisy data-sets over the maximal absolute amplitude of the Gaussian function as percentage of the ball density.

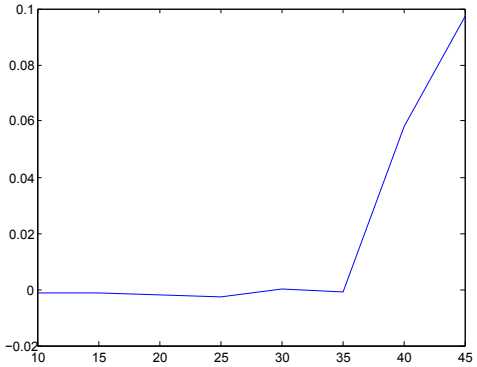

Figure 131: Mean difference of the surfaces computed with the β-criterion in noisy data-sets over the maximal absolute amplitude of the Gaussian function as percentage of the ball density.

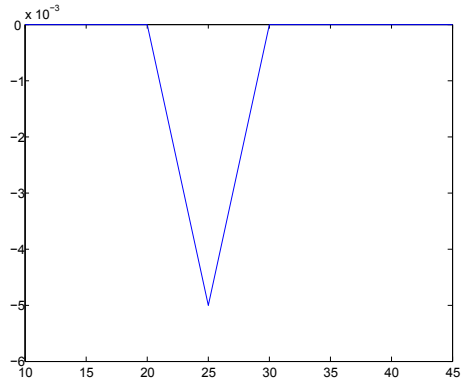

Figure 132: Median difference of the surfaces computed with the β-criterion in noisy data-sets over the maximal absolute amplitude of the Gaussian function as percentage of the ball density.

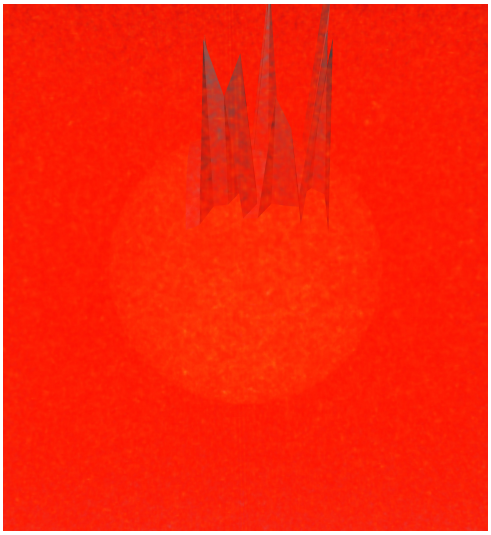

Figure 133: DVR-image of a surface computed with the β-criterion on the ball with Gaussian noise with maximal absolute amplitude of 40 of the ball intensity.

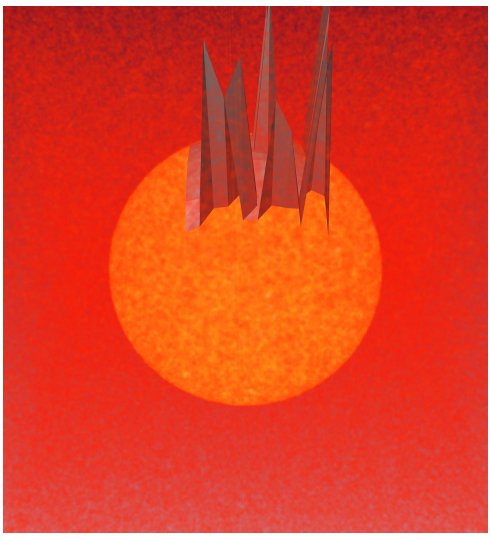

Figure 134: DVR-image of a surface computed with the β-criterion on the ball with Gaussian noise with maximal absolute amplitude of 40 of the ball intensity with α-scaling of $d_\alpha = 0.6$.

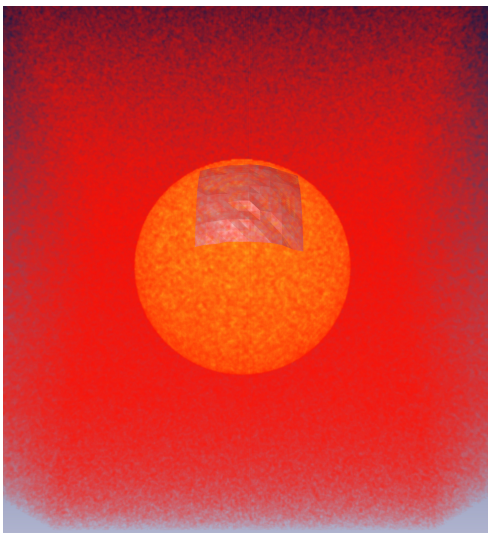

Figure 135: DVR-image of a surface computed with the β-criterion on the ball with Gaussian noise with maximal absolute amplitude of 45 of the ball intensity with α-scaling of $d_\alpha = 0.6$. Regardless of the noise amplitude the surface is computed correctly.

How does *surfseek* compute the surface with the γ-criterion? We already know that γ-criterion is more sensitive to noise, thus we can expect the surface quality to worsen faster compared to the β-criterion. In fact this behaviour is confirmed in Figure 136, where the reader can see the maximal difference of the computed surfaces to the reference surface over the noise amplitude in percent of the ball density.
In Figure 137 the reader can observe the mean difference as function of noise. And in Figure 138 the reader can observe the median difference to the reference surface over the noise amplitude in percent of the ball intensity.
We can see an increase of the difference to the reference surface at 30 percent. In Figure 137 we can observe that the mean difference grows continuously with the noise growth. In Figure 138 we cab observe that the median does not change over the noise, again the last high plateau in the graph corresponds to exactly on voxel of the data-set. Thus most surface points where computed correctly.
Let us take a look at the surface evolution from 30 to 40 percent. In Figure 139 the reader can examine a surface computed with the γ-criterion on a data set with a maximal absolute noise amplitude of 30 percent of the ball density. We can observe that only three points are computed wrong. When the noise is increased to 35 percent then the surface is computed in a way as can be observed if Figure 140. Note that for better visibility we applied α-scaling with $d_\alpha = 0.6$ after the surface computation. In this surface there are several points that are false detected but the majority is correct.
At a noise level of 40 percent we obtain a surface as in Figure 141. In this Figure we can again see that the noise reached a level, where it is not possible to compute the surface

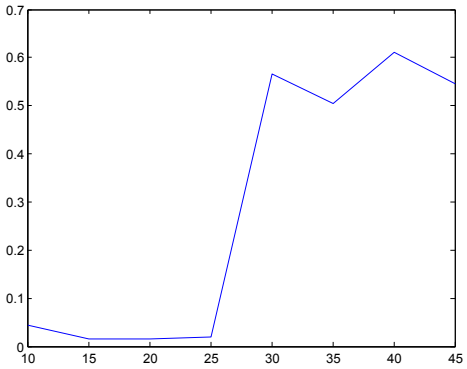

Figure 136: Maximum difference of the surfaces computed with the γ-criterion in noisy data-sets over the maximal absolute amplitude of the Gaussian function as percentage of the ball density.

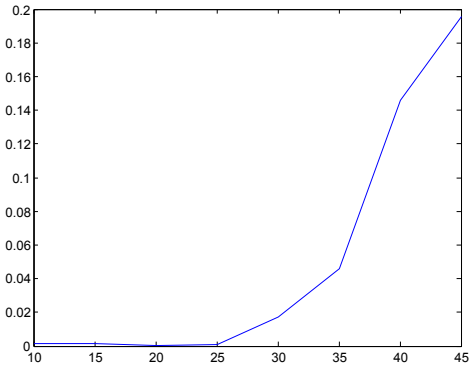

Figure 137: Mean difference of the surfaces computed with the γ-criterion in noisy data-sets over the maximal absolute amplitude of the Gaussian function as percentage of the ball density.

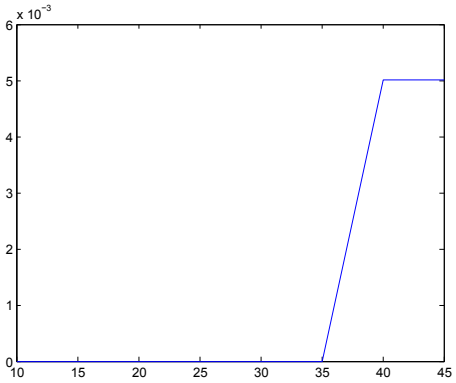

Figure 138: Median difference of the surfaces computed with the γ-criterion in noisy data-sets over the maximal absolute amplitude of the Gaussian function as percentage of the ball density.

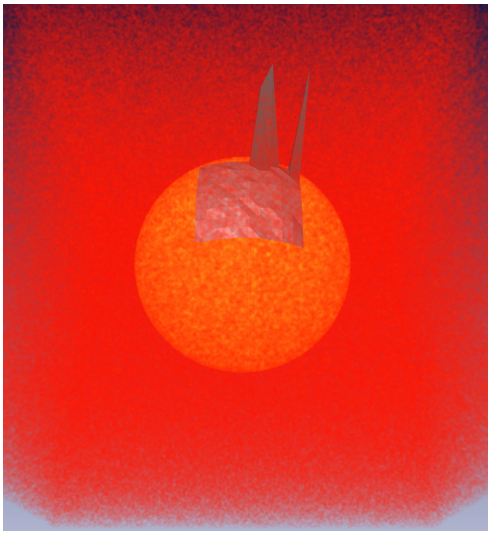

Figure 139: Surface computed with the γ-criterion on a noisy data-set with maximal absolute noise amplitude of 30 percent of the ball intensity.

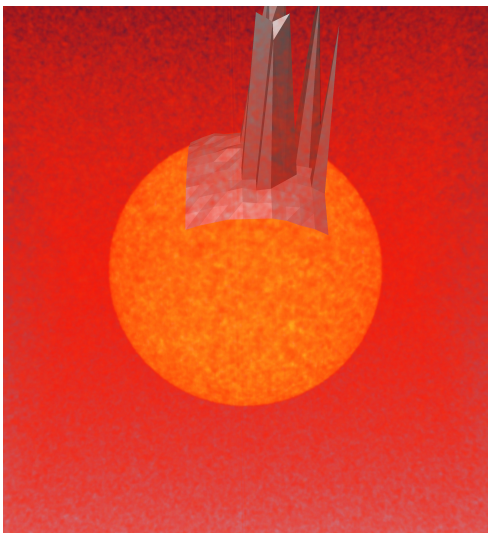

Figure 140: Surface computed with the γ-criterion on a noisy data-set with maximal absolute noise amplitude of 35 percent of the ball intensity. For better visibility the we applied α-scaling after the surface computation.

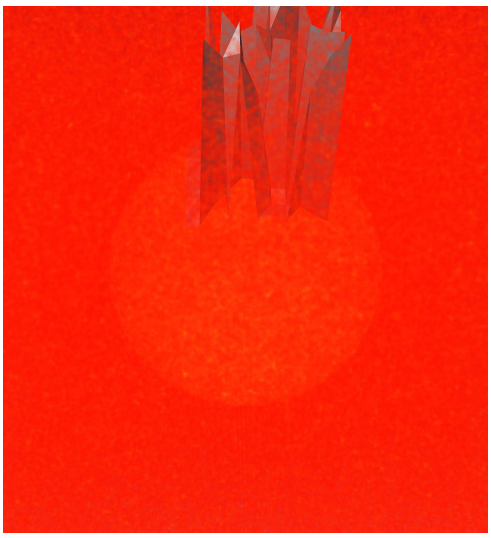

Figure 141: Surface computed with the γ-criterion on a noisy data-set with maximal absolute noise amplitude of 40 percent of the ball intensity.

correctly for all rays, since some run through fully opaque noise areas. Again if we use α-scaling before the surface computation we obtain a correct surface as can be observed in Figure 142.

Conclusion

We compared the quality of the two picking criteria in dependence of the maximal absolute amplitude of the Gaussian noise. We saw that the β-criterion fails at a noise level of 40 percent of the ball density and the γ-criterion started to compute a few false points at 30 percent of the ball density. Where at β-criterion we had a rapid increase of the false detected points, using the γ-criterion lead to a slow but continual increase of false detected points with the increase of the noise.

Both picking criteria can be improved greatly if we use α-scaling on data-sets with high noise amplitude before the surface computation.

We can derive from this test series that the algorithm *surfseek* is quite stable against white noise. This is a good result since most of the noise, that we encounter in experimental data is white noise.

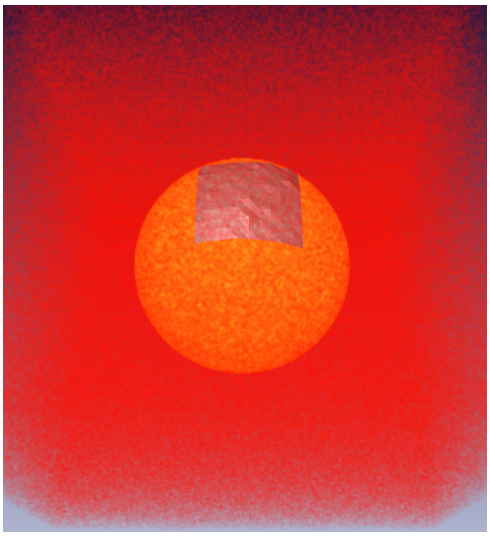

Figure 142: Surface computed with the γ-criterion on a noisy data-set with maximal absolute noise amplitude of 45 percent of the ball intensity after α-scaling with $d_\alpha = 0.6$.

4 Limitations, Conclusion and Outlook

In this section we will discuss the limitations of the presented algorithm *surfseek*,we will talk about an outlook and outline an outlook for further work.

4.1 Limitations of surfseek

We have seen that the algorithm *surfseek* is relatively stable concerning white noise and blurring in the data. The algorithm has difficulties with speckle noise and salt and pepper noise.
There are a few other cases where the algorithm *surfseek* has some difficulties to compute satisfactory surfaces. We will discuss those cases in detail.

4.1.1 Disappearing features

The algorithm has high difficulties when a feature becomes very thin or when the opacity of a feature becomes very small. In general we want, that the algorithm selects the most visible features, but there are cases, where it appears to be more appropriate to stay on one feature instead of jumping. In Figure 143 the reader can examine an illustration of a case where such a jump would occur. In the experimental data such a case can be observed in Figure 144, in this case the surface jumps between the kidney and the rib. This is due to the low opacity of the rib and the increased thickness of the kidney on the left side.
Another example was already seen in Figure 58. Here the bone tissue became so thin that it was displayed almost completely transparent. In such cases the algorithm *surfseek* would detect the surface behind or in front of the hole. Using a global ϵ when we compute the surface with the β-criterion leads to a hole in the surface, which is in some sense better than a false detected feature, but is still not the optimal result.

4.1.2 Crossing Features

An another case where the algorithm has difficulties to compute a proper surface is a uncommon and probably theoretical case where two very similar features overlap in some small area. In Figure 145 the reader can examine an illustration of such a case. In this Figure we have an overlapping of two features with exactly the same properties. Now there are four surfaces that are reasonable. The algorithm could follow the features, thus compute one of the two tilted surfaces, or it could compute the front faces of the features or the faces that lie behind.
There is no simple decision which of those four surfaces the algorithm should choose. Clearly we can say that the algorithm should not jump between the two surfaces, but what should happen at the overlapping point?
One suggestive criterion is that the algorithm should follow the surface of one feature. If we assume that the features contribute different amount of opacity to the surface

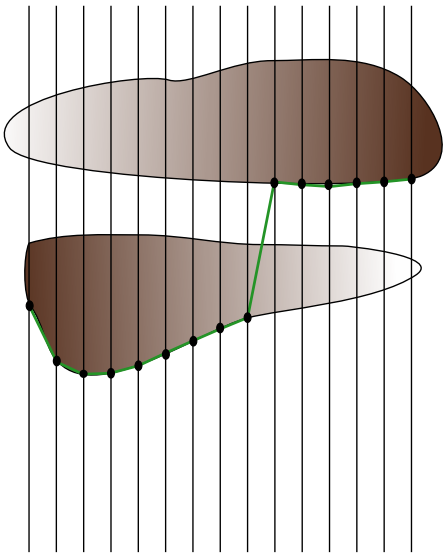

Figure 143: When a feature becomes thin or otherwise transparent then the algorithm jumps to another most visible surface.

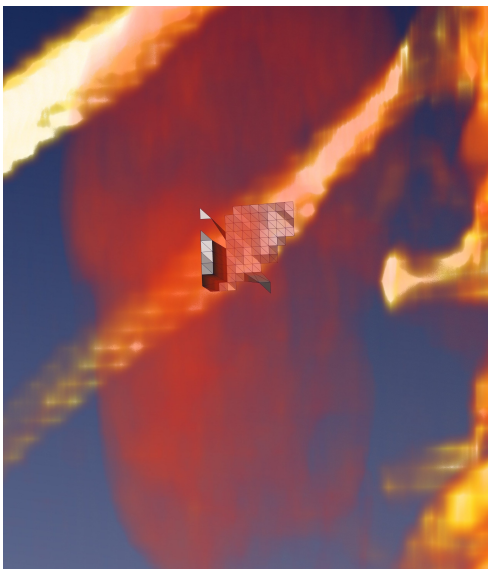

Figure 144: The algorithm jumps from the kidney to the opaque becoming rib.

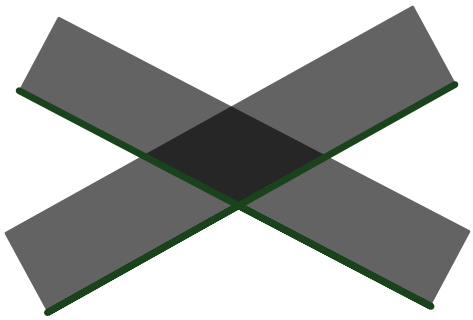

Figure 145: When two very similar features overlap, the algorithm has four possible surfaces that it can compute.

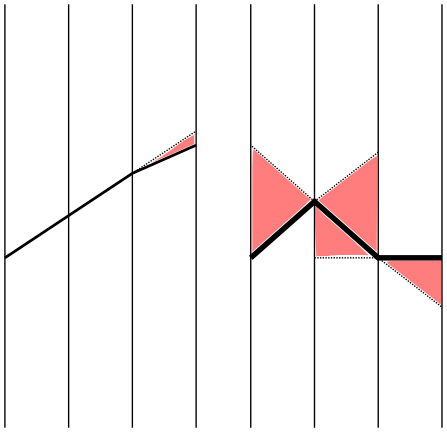

Figure 146: Additional weight component that is a measure for the surface orientation change could help to follow the surface of one feature. In this Figure the red area could be used as an weight for the orientation change.

area, the algorithm should follow the surface with the most contribution. This could be achieved by adding a third component to the edge weights. This component should contain some degree of the change of the surface orientation. In Figure 146 the reader can see an example of such a weight component. The red area is an indicator of the magnitude of the orientation change. With such an additional weight the algorithm should select one of the tilted surfaces in the example of Figure 145.
However on the other hand, this would lead to a neglect of a feature if the surface orientation changes naturally compared to a totally planar but less visible feature. Furthermore we can not assume that there are no features that consist of such a crossing, specifically: we can not disallow the existence of such complex features.
The current *surfseek* algorithm would select the front surface for both of the picking criteria.

4.1.3 Large surface patches and β-criterion

If we use the β-criterion with a global ϵ we obtain a good result for small surfaces. If the surface is very large and covers features with high opacity variance, it can happen, that the computed surface has holes. This is the case when a ray hits a feature with very fast opacity increase, that is surrounded with otherwise less opaque features. This can occur through noise or some small features with high opacity behind or in front of the wanted feature. Even if the feature does not change, it may have varying opacity along the computed surface.
Thus the more area the surface covers or the higher the ray density is, the higher are the chances that a ray detects a uncommonly high opacity increase.
To avoid such holes we can use a local ϵ or we use a semi global ϵ. By semi global we mean that for each ray we choose ϵ as

$$\epsilon = \max_{x,y,j} \beta_{x,y,j} \cdot 0.3 \text{ with} \tag{36}$$

$$|x - x_r| \leq 2,\ |y - y_r| \leq 2, \tag{37}$$

where x_r and y_r denote the position of the current ray.
Using a semi global epsilon would not guarantee that the surface is computed hole free, but rather limit the size of the holes.

4.1.4 Similar feature opacities

Though the algorithm *surfseek* is robust against noise, it still has some difficulties with foggy data-sets. When we try to compute a surface patch for a vague visible feature with some foggy area in front of the feature, the algorithm can detect several false points. Such a behaviour can be observed in Figures 147 and 148.
For both of those surfaces the rays run through a foggy area with similar opacity values as the kidney. Thus the algorithm computes several false points. Interestingly such a

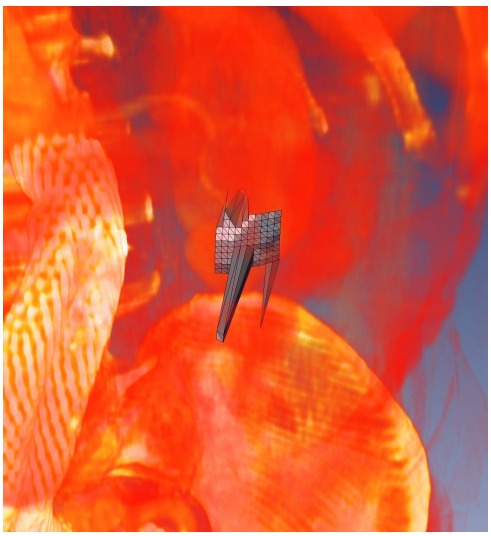

Figure 147: A surface computed with *surfseek* using the β-criterion. The rays had to go through a foggy area with similar opacity values as the kidney.

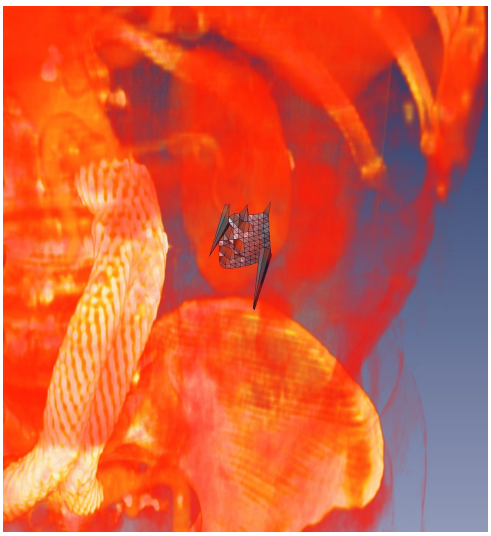

Figure 148: A surface computed with *surfseek* using the γ-criterion. The rays had to go through a foggy area with similar opacity values as the kidney.

foggy area appeared more strongly on one side of the torso then the other, when we try to compute the surface of the right kidney the result is much better, see Figure 149.

4.2 Conclusion

We presented a new algorithm *surfseek* that computes the surfaces of the most visible features in DVR. For the algorithm we used two different picking criteria, the β- and γ-criterion.
We compared the surfaces computed with *surfseek* with those from a direct picking using WYSIWYP. We saw that the surfaces computed with *surfseek* are qualitatively better then those computed with direct picking.
In the section three we studied the surface qualities in dependence of noise. We saw that the algorithm *surfseek* is very robust against white noise and Gaussian blurring. We also saw that *surfseek* is vulnerable against speckle and salt and pepper noise.
At the end of the third section we showed the surface quality as a function of white noise. We compared the computed surfaces in noisy data-sets to a reference surface in a noise free data-set. Hereby we saw that the β-criterion computes the surfaces correctly up to a noise level of 35 percent of the feature intensity, at 40 percent noise intensity the amount of false detected points is suddenly very high and grows only little from that point on. Whereas the γ-criterion computes the surface fully correctly up to a noise level of 25 percent and then the amount of false detected points grows slowly with the noise amplitude.
To sum it up, we can say that both picking criteria are robust against white noise.
Previously in this section we discussed the limitations of *surfseek*, we reported that the

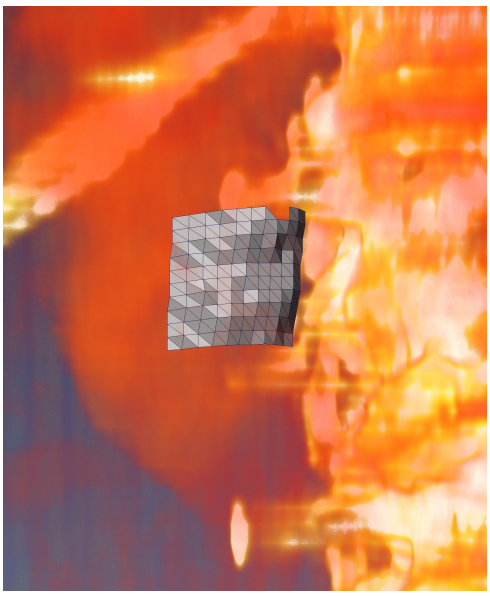

Figure 149: A surface computed with *surfseek* using the β-criterion. The rays had to go through a foggy area with similar opacity values as the kidney. But the foggy are was thinner compared to the foggy are in Figure 147

algorithm can fail if the opacity of a feature decreases greatly and if the wanted feature is surrounded by a foggy area with a similar opacity contribution.
During this work the algorithm was tested on a variety of different data-sets, synthetic data-sets and data-sets from CT-scans. For the surface quality it was irrelevant if the data-set was a synthetic one ore an experimental one.
The main goal, to select the most visible feature was achieved in both data-set types. In Figures 150 and 151 the reader can examine the different surface positions in dependence of the transfer-function. In Figure 150 the reader can see the inner ball clearly through the less visible bigger outer ball, thus *surfseek* computes the surface on the inner ball. In Figure 151 the inner ball is not recognisable through the outer ball, thus the algorithm computes the surface on the outer ball. Hereby it was not important which picking criterion was used.
The same behaviour can be observed in experimental data. In Figure 152 the reader can observe a part of the torso data-set, the transfer-function was adjusted that the thigh bone is clearly visible. In Figure 153 the reader can observe the same data-set with a different transfer-function, now the muscle tissue covers the bone almost completely.
If we apply *surfseek* with the β-criterion we receive the following surfaces, see Figures 154 and 155. Note that we used global ϵ for this computation, the usage of local ϵ leads to a different surface in Figure 154. Using a local epsilon would cause the algorithm to select the muscle tissue for the rays that do not hit the thigh bone.

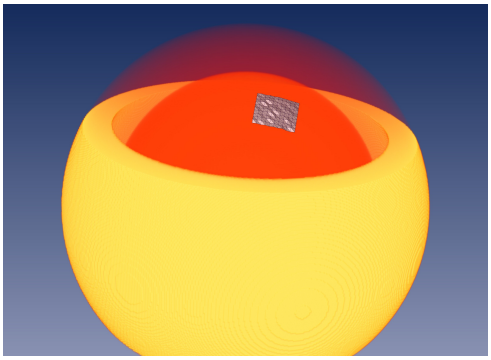

Figure 150: The smaller ball is more visible in this image, thus the algorithm selects the smaller ball, regardless of the picking criterion.

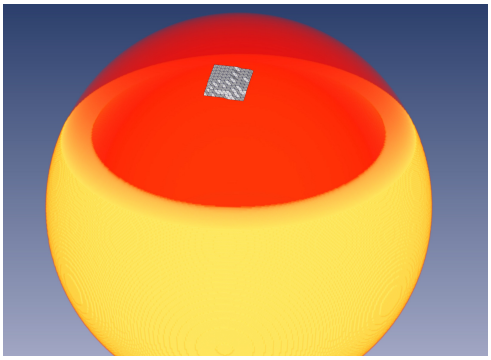

Figure 151: The inner ball is not recognisable through the outer ball, thus the algorithm selects the outer ball, regardless of the picking criterion.

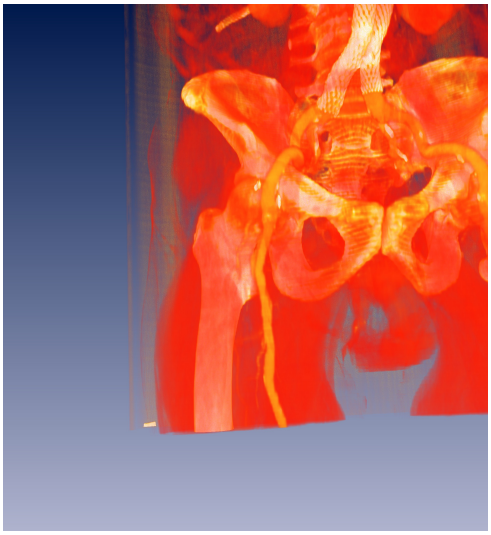

Figure 152: A part of the torso data-set, the transfer-function was adjusted in a way that the thigh bone is clearly visible.

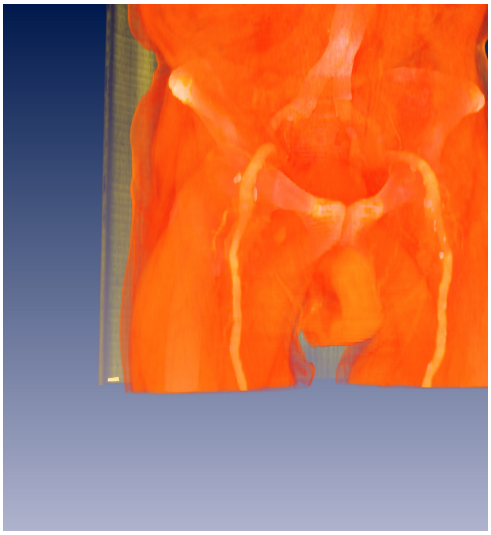

Figure 153: The torso data-set again, with a different transfer-function. The muscle tissue has a high enough opacity that it cover the bone almost completely.

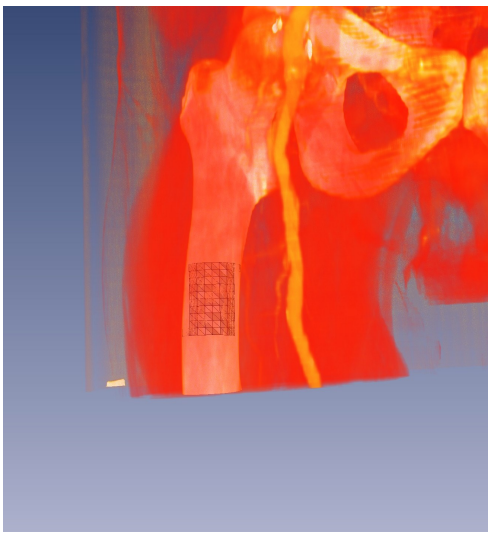

Figure 154: A surface computed with *surfseek* using the β-criterion on the data-set seen in Figure 152.

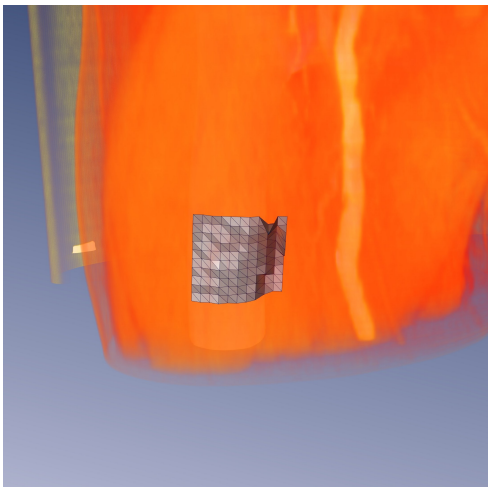

Figure 155: A surface computed with *surfseek* using the β-criterion on the data-set seen in Figure 153.

If we use the γ-criterion we obtain surfaces as in Figures 156 and 157. We can see that the surface on the bone has multiple false detected points, this is due to the natural noise of the CT-scan that divides the bone feature up in sub-features. But in general most of the points are computed correctly.
The drawback of the *gamma*-criterion can be observed in all CT-scans, with human bodies as well as with artificial objects. The noise in CT-scans is inevitable. In Figure 158 the reader can observe a surface computed with *surfseek* using the β-criterion in a CT-scan of a full backpack. In Figure 159 the reader can see the corresponding surface computed using the γ-criterion on the same data-set. We can see that there are some false detected point in the surface, that was computed with the γ-criterion. Note that we had to cast the rays through a foggy area to achieve this misfits. The foggy area can be better seen in Figure 160, where the whole backpack is displayed.
Even though both of the picking criteria showed to be robust against big white noise amplitudes, the γ-criterion showed to be vulnerable to minor changes in the accumulated opacity function sometimes. The amount of false detected points is usually very small, less than 5 percent of the points, but they lie far apart from the wanted surface so that the computes surface looks worse that it actually is.
A good post filter would be useful to correct the false detected points, especially because the misfits are displayed isolated.
In conclusion I made the experience that the algorithm detects the surfaces correctly, although the β-criterion leads to less misfits than the γ-criterion, both criteria lead to wanted surfaces in the most cases. In particularly the algorithm *surfseek* helped me to gain insight of the feature surfaces that was not provided by the DVR-image. For a final example I refer to the Figures 161, 162 and 163. In those figures we show the DVR-image

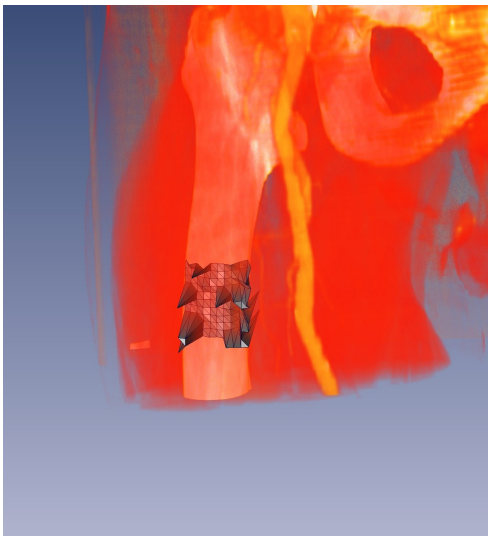

Figure 156: A surface computed with *surf-seek* using the γ-criterion on the data-set seen in Figure 152.

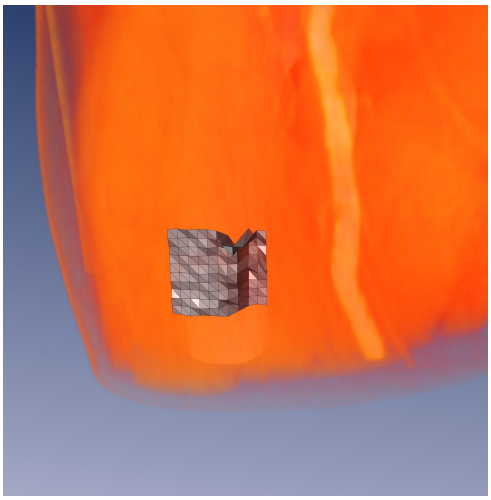

Figure 157: A surface computed with *surf-seek* using the γ-criterion on the data-set seen in Figure 153.

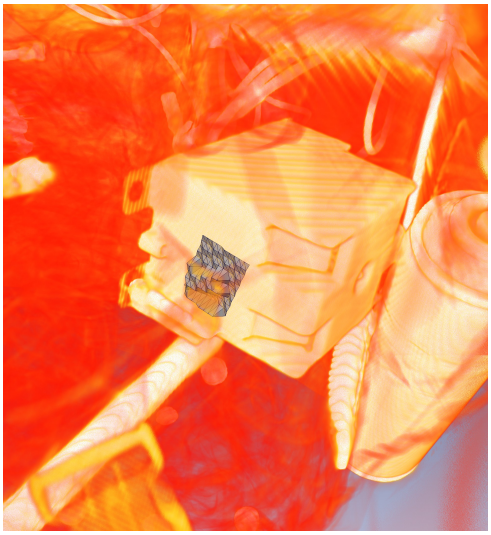

Figure 158: A surface computed with *surf-seek* using the β-criterion on a CT-scan of a full backpack. The rays for the surface computation were casted through a foggy area.

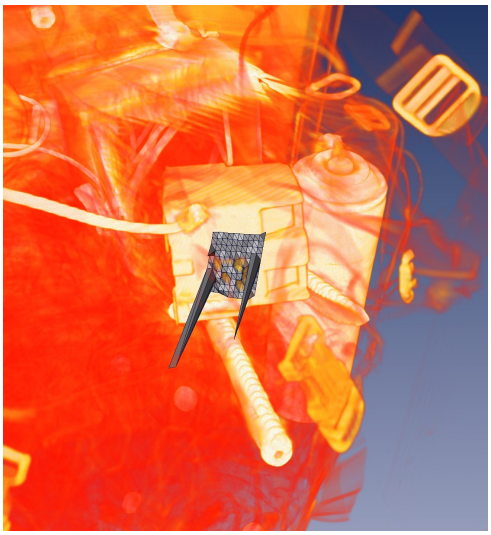

Figure 159: A surface computed with *surf-seek* using the γ-criterion on a CT-scan of a full backpack. The rays for the surface computation were casted through a foggy area.

Figure 160: A DVR image of a full backpack.

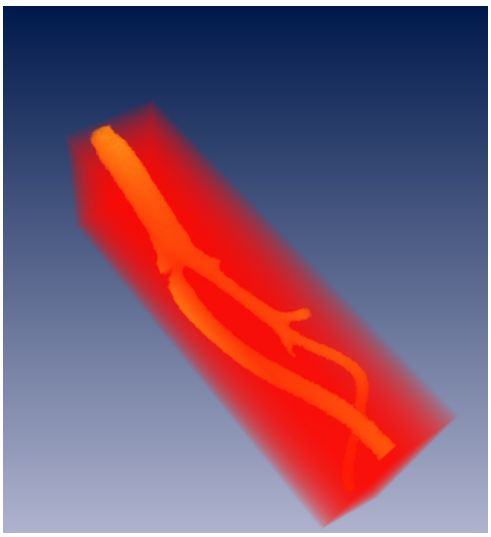

Figure 161: A DVR-image of a blood vessel. The foggy area and the jaggy surface make it hard to gain insight over the feature properties.

of a blood vessel, the DVR-image with the computed surface and the computed surface only. The surface was computed with *surfseek* using the β-criterion. I found that the extraction of the surface helped to gain insight of the feature properties. In many cases the computed surface improved the perception of feature properties that were hard to perceive using the DVR-image only. For example the depth perception or the perception of the surface structure was improved greatly after applying *surfseek* on the dataset.

4.3 Outlook

Even though the surfaces in this paper were projected from a rectangle, the algorithm can be slightly modified to compute an irregular but closed surface that is drawn by the user. For this the algorithm must not be changed greatly as it already automatically counts the number of rays for each ray separately.
As already mentioned in the section about the limitations of *surfseek* a new parameter might be useful that supports the tracking of one feature. A useful parameter has yet to be found, as it can also neglect curvy surfaces.
There are already many algorithms that recover a surfaces of a given amount of points, for example I refer to the paper of Shraf et al. [SLSK]. It makes sense to create a post-filter algorithm that deletes the misfits and computes the missing points with a surface reconstruction algorithm.

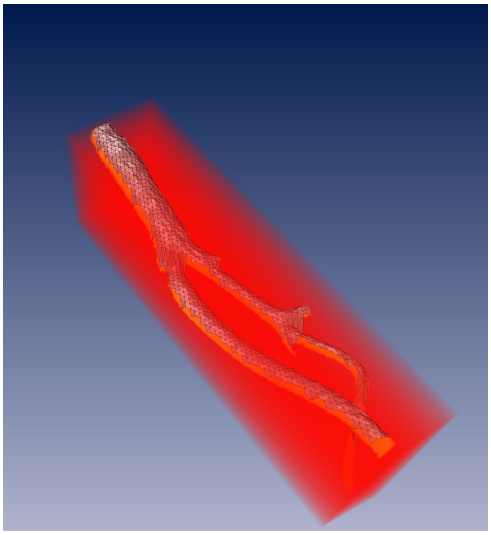

Figure 162: A surface computed with *surfseek* on the blood vessel from Figure 161, computed with the β-criterion.

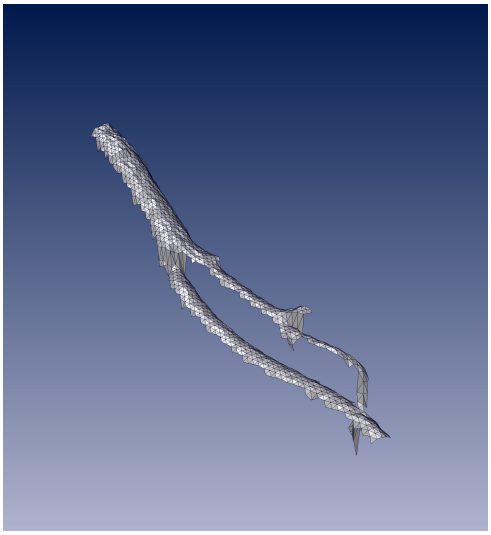

Figure 163: The extracted surface from Figure 162. Such an image can help to gain insight over the feature properties.

5 Appendix

5.1 Acknowledgements

I want to thank both of my supervisors Prof. Dr. Polthier (Freie Universität Berlin) and Prof. Dr. Wiebel (Hochschule Coburg) for the excellent care during my work on this thesis. I want to thank Prof. Dr. Wiebel for introducing me to the highly exciting field of Scientific Visualization through his lecture.
I also want to thank the researchers at Zuse Institut Berlin for providing help whenever I needed some.
I want to thank Prof. Dr. Klaus Lehnertz (Klinik für Epileptologie, Universität Bonn) for providing images of noise and measurement artefacts in a CT-scan.
All DVR images were produced with the software Amira.
All Graph images where produced with the software MATLAB (www.mathworks.de).
All illustrative images of the algorithm where produced with the software Inkscape (inkscape.org).
The skull data-set and the torso data-set were downloaded at "www.volvis.org".

5.2 Terminology

DVR
Direct Volume Rendering, a method to visualise 3D scalar data sets by casting rays through the data set and evaluate the rays. See [PS] for more information.

Feature
By feature we mean a part of the 3D-data that is perceived as an unit by the user, such as organs, blood vessels or machinery parts.

Ray
Rays that start from the viewpoint are cast through the volume to gather information about the data. The information is gathered using (usually equidistant) samples along the ray according to equation 2 and equation 3.

Ray casting
A method where rays are cast through the volume to gather information for the resulting image or further computation.

Sample
In this context a sample denotes a position on a ray with the corresponding color and opacity values.

Slice
An image of a 2D-part of the data. Imagine a thin slice that is cut out of the volume.

Transfer function
A function that maps density values to color and opacity values. The transfer function allows different visualisations of the same data, and hence is able to provide a greater insight.

Volume rendering integral
The volume rendering integral describes the accumulated intensity in one color along a ray. It is based on the emission-absorption-model, where each sample emits and absorbs light.

WYSIWYP
What You See Is What You Pick, an algorithm introduces in [WVFH].

$\alpha^{acc}, \beta^{acc}, \gamma^{acc}$
α^{acc} is the accumulated opacity function for a certain ray with the value $\alpha^{acc}(i)$ for a sample i and the first derivative β^{acc} and the second derivative γ^{acc}.

References

[BK] http://www.boost.org/doc/libs/1_47_0/libs/graph/doc/boykov_kolmogorov_max_flow.html; (Jul. 23, 2013)

[BK04] Yuri Boykov and Vladimir Kolmogorov: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, In IEEE Transactions on PAMI, Vol. 26, No. 9, pp. 1124-1137, Sept. 2004

[BNG] Tom Brunet, K. Evan Nowak, and Michael Gleicher, Integrating dynamic deformations into interactive volume visualization, Eurographics / IEEE-VGTC Symposium on visualization, Citeseer, 2006, pp. 219-226.

[BSGHBYD] S. Bruckner, V. Šoltészová, E. Gröller, J. Hladůvka, K. Bühler, J. Yu, and B. Dickson. Braingazer - visual queries for neurobiology research. IEEE Transactions on Visualization and Computer Graphics, 15(6):1497–1504, Nov. 2009

[BSL] Bushberg, J.; Seibert, J.; Leidholdt, E.; Boone, J. The Essential Physics of Medical Imaging. 2nd edn. Lippincott Williams and Wilkins; Philadelphia, PA: 2002.

[DW] Douglas B. Westt: Introduction to Graph Theory, Pearson: 2 edition (September 1, 2000), ISBN-10: 0130144002.

[GPZT] E. Gobbetti, P. Pili, A. Zorcolo, and M. Tuveri. Interactive virtual angioscopy. In Proc. of the conference on Visualization '98, VIS '98, pages 435–438, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press

[KS] A. C. Kak and Malcolm Slaney, Principles of Computerized Tomographic Imaging, Society of Industrial and Applied Mathematics, 2001, page 177-201.

[LG] Lea Grady: Computing Exact Discrete Minimal Surfaces: Extending and Solving the Shortest Path Problem in 3D with Application to Segmentation, Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)

[L] Prof. Dr. Klaus Lehnertz (Klinik für Epileptologie, Universität Bonn), Lecture on CT-scans:
http://epileptologie-bonn.de/cms/upload/homepage/lehnertz/CT2.pdf

[LC] William E. Lorensen, Harvey E. Cline, „Marching Cubes: A High Resolution 3D Surface Construction Algorithm", ACM Computer Graphics Vol. 21 No. 4 (SIGGRAPH 1987 Proceedings)

[LWCS] Kang Li, Xiaodong Wu, Danny Z. Chen and Milan Sonka: Optimal Surface Segmentation in Volumetrix Images - A Graph-Theoretic Approach. Ieee transactions on pattern analysis and machine intelligence, Vol. 28, No. 1, January 2006

[MG] Jirí Matousek, Bernd Gärtner: Understanding and Using Linear Programming, Springer Berlin Heidelberg (2. Juni 2010), ISBN-10: 3540306978.

[N] Gregory M. Nielson. On Marching Cubes. IEEE TVCG 9(3):283-297, 2003

[PJP] Philipp Johannes Preis: Sichtbarkeitsorientiertes Picking in Direct Volume Renderings mit Beleuchtungsmodellen. (Diplomarbeit an der Freien Universität Berlin; Dec. 13, 2012)

[PS] P. Sabella: A rendering algorithm for visualizing 3D scalar fields. SIGGRAPH Comput. Graph., 22: 51-58, June 1988

[SLSK] Andrei Shraf,Thomas Lewiner, Ariel Shamir,l Leif Kobbelt and Daniel Cohen-Or: Competing fronts for coarse–to–fine surface reconstruction. Found at: www.cs.bgu.ac.il/ asharf.

[SO] Issa Al-Shakhrah and Tariq Al-Obaidi: Common artifacts in computerized tomography: A review, Applied Radiology Volume 32, Number 8, August 2003, page 25-30

[TSC] Shu-Ju Tu, Chris C Shaw, and Lingyun Chen: Noise simulation in cone beam CT imaging with parallel computing, Published in final edited form as: Phys Med Biol. 2006 March 7; 51(5): 1283–1297.

[WM] Jerold W. Wallis and Tom R. Miller, Three-dimensional display in nuclear medicine and radiology, The journal of nuclear medicine 32 (1991), no. 3.

[WPFV] Wiebel A., Preis P., Vos F.M. and Hege H.-C.: Technical Report, Zuse-Institut Berlin, Computation and Application of 3D Strokes on Visible Structures in Direct Volume Rendering. (April 21, 2013)

[WVFH] Wiebel A., Vos F. M., Foerster D. and Hege H.- C.: WYSIWYP: What you see is what you pick. IEEE Transactions on Visualization and Computer Graphics 18, 12 2236-2244.(Dec. 2012). doi:10.1109/TVCG.2012.292.