Contents

List of Figures

Chapter 1

Segmentation

Image segmentation is the process of dividing an image into meaningful non-overlapping regions or objects. The main goal is to divide an image into parts that have strong correalation with objects of the real world. Segmented regions are homogenous according to some property, such as pixel intensity or texture. Mathematically speaking, a complete segmentation of an image I is a finite set of regions $I_1, ..., I_S$ such that the condition (from [3])

$$I = \bigcup_{i=1}^{S} I_i, \quad I_i \cap I_j = \emptyset, \quad i \neq j \tag{1.1}$$

is satisfied. Image segmentation is one of the first steps leading to image analysis and interpretation. It is used in many different fields, such as machine vision, biometric measurements and medical imaging.

Automated image segmentation is a challenging problem for many different reasons. Noise, partial occluded regions, missing edges, lack of texture contrast between regions and background are some of the reasons. Noise is an artifact often found in images which makes the segmentation process harder. In the process of generating medical images noise is often introduced by the capturing devices. As a pre-processing step before segmentation the image can be smoothed to reduce noise. In the context of medical images segmentation usually means a delineation of anatomical structures. This is important for e.g. measurements of volume or shape. Low level segmentation methods are usually not good enough to segment medical images. Thus, higher level segmentation methods that are more complex and gives better results are used. The biggest difference between low-level segmentation methods and higher level segmentation methods is the use of apriori information. Low-level methods usually have no information about the im-

age to be segmented, while high-level segmentation methods can incorporate different types and amount of apriori information.

Traditional low-level image segmentation methods can roughly be divided according to the type of technique used:

- Global/Histogram based methods
- Region based methods
- Edge based methods

1.1 Histogram-based segmentation methods

Global knowledge about an image is usually represented by the histogram of the intensity values in the image. Histogram-based segmentation methods uses this information to segment simple images. These segmentation methods are usually much faster than other methods, but restricted to images with simple features.

1.1.1 Thresholding

The simplest segmentation approache is called thresholding. Thresholding is used to seperate objects from the background using a threshold value T. A threshold value splits the image in two groups, where all pixels with intensity value higher than T represents an object or the foreground, and the rest represents another object or the background. Choosing a good thresold value is important, as small changes in the value can significantly affects the resulting segmentation, which can be seen in figure 1.1c and 1.1d (described in more detail later). The threshold can be selected manually by either inspecting the image or the histogram of the image. But usually the threshold is selected automatically, and a variety of methods for automatically selecting T exists. When little noise is present, the mean or median intensity values can be selected as the threshold. The simplest method to select a threshold, apart from doing it manually, is iterative thresholding and is computed as follows:

1. Choose an initial threshold T_0 and segment the image.
2. The segmented image will consist of two groups, C_1 and C_2. Set the new threshold value T_i to be the sum of the mean intensity values from C_1 and C_2, divided by 2.
3. Segment the image using T_i.

4. Repeat steps 2 and 3 until $|T_i - T_{i-1}|$ is less than a predefined value.

By using multiple threshold values the image can be split up into several regions. Segmentation by thresholding is only suitable for very simple images, where the objects in the image does not overlap and their intensity values are clearly distinct from the background intensity values. If the threshold is poorly chosen, the resulting binary image would not be able to correctly distinguish the foreground from the background.

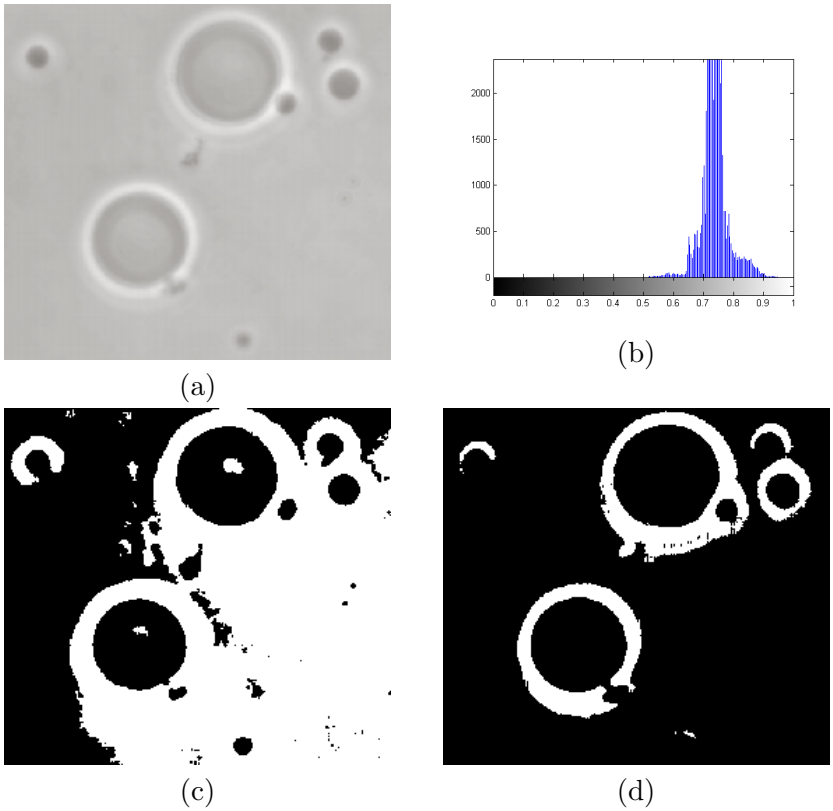
1.1.2 Otsu's thresholding method

Otsu's thresholding method assumes that the image contains two regions with the values in each region creating a cluster. Otsu's method tries to make each cluster (or class) as tight as possible, thus minimizing their overlap. The goal then is to select the threshold that minimizes the combined spread. The threshold that maximizes the between-class variance $\sigma_b^2(t) = \omega_1(t)\omega_2(t)\left[\mu_1(t)\mu_2(t)\right]^2$ is sought after. $\omega_1(t)$ and $\omega_2(t)$ are the weights (computed from the normalized histogram) of the two clusters, and $\mu(t)$ is the mean intensity value of the clusters. Otsu's method starts by splitting the histogram into two clusters using an initial threshold. Then $\sigma_b^2(t)$ is computed for that threshold value. The between-class variance $\sigma_b^2(t)$ is then iteratively computed for every intensity value, and the threshold that maximizes the between-class variance $\sigma_b^2(t)$ (or minimizes the within-class variance) is chosen as the final threshold value.

Figure 1.1 illustartes a gray-scale image and the segmentation results using both iterative global thresholding and Otsu's method. The threshold found using the iterative threshold method is 0.7332 where the range is from 0 (black) to 1 (white). The threshold found using Otsu's method is 0.7686. The image to be segmented is shown in figure 1.1a, and its histogram in figure 1.1b. As can be seen from the histogram, it is not possible to select a near perfect threshold by just looking at it. Figure 1.1c illustrates the segmentation result from the iterative global thresholding method and figure 1.1d is the segmentation result using Otsu's method. Even though the difference of the two threshold values is small, the segmentation results have a considerable difference, where Otsu's method gives a better result.

1.2 Region based segmentation

Region based segmentation methods tries to find homogenous regions based on gray-scale, color, texture or any other pixel based measure in an image. Pixels with similar properties are grouped together in regions I_i. The choice



(a) (b)

(c) (d)

Figure 1.1: (a): Image to be segmented, (b): Histogram of image, (c): Segmented using iterative global thresholding, with $T = 0.7332$, (d): Segmented using Otus's method with $T = 0.7686$.

of homogenity criteria is an important factor that affects the end segmentation result. In addition to the condition in equation 1.1, images segmented by region based segmentation also satisfies the two following conditions:

- All regions I_i should be homogenous according to some specified criteria: $H(I_i) = true,\ i = 1, 2, ..., S.$
- The region that results from merging two adjacent regions R_i and R_j is not homogenous: $H(I_i \cup I_j) = false.$

An example of a homogenity criteria for a region could be all adjacent pixels with intensity value within a range $\{x, y|x \pm y\}$. That is, if two adjacent pixels have intensity values in the range $x \pm y$ they are in the same region. Region based segmentation methods are usually better than edge based

segmentation methods in noisy images where the borders are difficult to detect.

1.2.1 Region growing

An example where thresholding is insuficient is when parts of the foreground have the same pixels values as part of the background. In this case, region growing can be used. Region growing starts at a point (seed point) defined to be inside the forground and grows to include neighbouring foreground pixels. This seed point is manually set at the beginning and consiste of one or more pixels. A small region of 4x4 or 8x8 can for example be chosen as a seed region. The regions described by the seed points grows by merging with their neighbouring points (or regions) if the homogenity criteria is met. This merging is continued until merging any more would violate the homogenity criteria. When a region cannot be merged with any of its neighbours it is marked as final, and when all regions are marked as final the segmentation is completed. The result of region growing can depend on the order in which the regions are merged. Thus, the segmentation result may differ if the segmentation begins, for example, in the top right corner or the lower left corner. This is because the order of the merging can cause two similar adjacent regions R_1 and R_2 not to be merged if an earlier merge of R_1 and R_3 changed the characteristics of R_1 such that it no longer is similar (enough) to R_2.

1.2.2 Region splitting

Region splitting is the opposite of region growing, and starts with a single region covering the whole image. This region is iteratively split into smaller regions until all regions are homogenous according to a homogenity criteria. One disadvantage of both region growing and region splitting is that they are sensitive to noise, resulting in regions that should be merged remaining unmerged, or merging regions that should not be merged.

1.3 Edge based segmentation

Edge based segmentation methods are used to find edges in the image by detecting intensity changes. The edge magnitude at a certain point is the same as the gradient magnitude, and the edge direction is perpendicular to the gradient. Thus, change in intensity at a point can be detected by using first and second order derivatives. There are various edge detection operators, and they all approximate a scalar edge value for each pixel in an image

based on a collection of weights applied to the pixel and its neighbours. These operators are usually represented as rectangular masks or filters consisting of a set of weight values. These masks are applied to the image to be segmented using discrete convolution.

1.3.1 First and second order operators

A simple second order edge detection operator is the Laplacian operator, based on the Laplacian equation:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{1.2}$$

This equation measures edge magnitude in all directions and is invariant to rotation of the image. Second order derivatives are commonly discretized by approximating it as $\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$ which is also how the Laplacian is discretized:

$$\nabla^2 f(x,y) = f(x+1,y)+f(x-1,y)+f(x,y+1)+f(x,y-1)-4f(x,y) \tag{1.3}$$

This Laplacian equation is represented by the mask in equation 1.4, and a variant of the equation that also takes into account diagonal elements is shown in 1.5.

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{1.4}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{1.5}$$

Since the Laplacian mask is based on second order derivatives it is very sensitive to noise. Moreover, it produces double edges and is not able to detect the edge direction. The center of the actual edge can be found by finding the zero-crossing between the double edges. Hence, the Laplacian is usually better then first order derivatives to find the center-line in thick edges. To overcome the sensitivity to noise problem, the image can be smoothed beforehand. This is the idea behind the Laplacian of Gaussian (LoG) operator. The LoG mask is a combination of a Gaussian operator (which is a smoothing mask) and a Laplacian mask. By convolving an image with a LoG mask it is smoothed at the same time as edges are detected. The smoothness is determined by the standard deviation of the Gaussian, which also determines the size of the LoG mask.

There are various masks based on first order derivatives, and two of them are the Prewitt and Sobel masks, represented in equation 1.6 and 1.7

respectively. These two are not rotation invariant, but the masks can be rotated to emphasize edges of different directions. The masks as they are represented in equation 1.6 and 1.7 highlights horizontal edges.

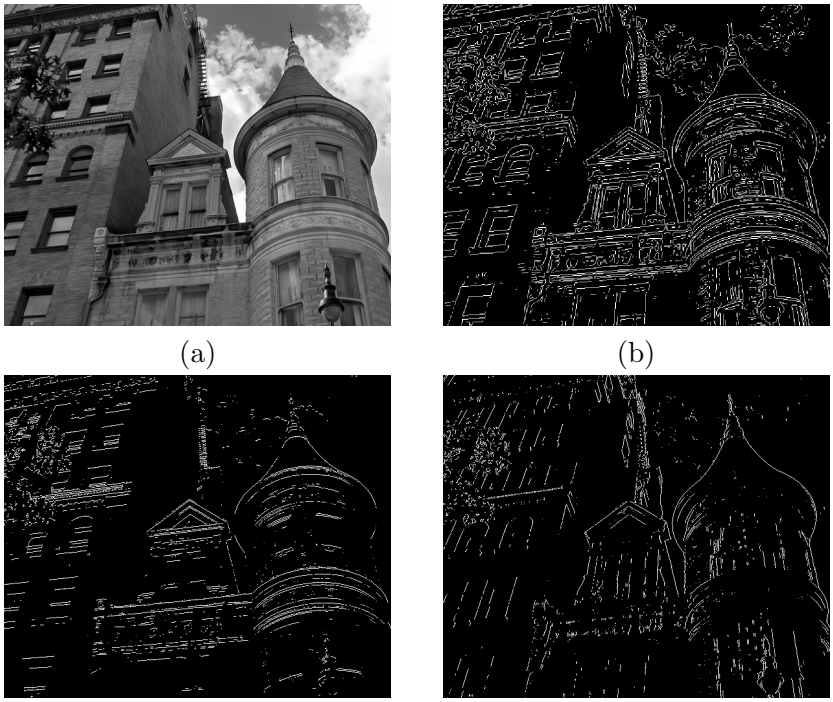
$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \quad (1.6) \qquad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad (1.7)$$

As can be seen from the above masks, the only difference between the Sobel and Prewitt is that the middle column (or row in a rotated version) in the Sobel mask is weighted by 2 and -2. This results in smoothing since the middle pixel is given more importance, hence, the Sobel is less sensitive to noise than Prewitt.

Figure 1.2a illustrates a gray-scale image and 1.2b is the edge segmented image based on LoG. 1.2c is the edge image resulted from the Sobel mask in equation 1.7b and 1.2d is the result from segmentation after rotating the mask 90°. The segmentations resulted by using the Prewitt operator to segment the image in figure 1.2a had no significant difference from the Sobel segmented images.

1.3.2 Canny edge detector

A more powerful edge detection method is the Canny edge detector. This method consist of four steps. The first step is to smooth the image based on a Gaussian filter with a given standard deviation σ. In the next step the derivatives in both directions are computed using any first order operator, and using these the gradient magnitude image and its direction are computed. The gradient magnitude image typically contains wide ridges around local maxima of the gradient. In order to get a single response to an edge, only local maxima should be marked as edges, and this process is called non-maxima suppression. A simple way for non-maxima suppression is to first quantize the edge directions according to 8-connectivity (or 4 connectivity). Then consider each pixel with magnitude > 0 as candidate edge pixels. For every candidate edge pixel look at the two neighboring pixels in edge-direction and the opposite direction. If the magnitude of the candidate edge pixel is not larger than the magnitude of these neighboring pixels, mark the pixel for deletion. When all candidate edge pixels are inspected, remove all the candidates that are marked for deletion. Now all the edges will contain a single response, but there still are lines/pixels that are not part of any continues edge. To remove these, hysteresis thresholding is used. Hysteresis thresholding consist of segmenting the image with two threshold



(a) (b)

(c) (d)

Figure 1.2: (a): Image to be segmented, (b): LoG, (c): Sobel - highlighting horizontal edges, (d):Sobel - highlighting vertical edges

values. First, the non-maxima supressed images is thresolded with a high thresold value T_h that determines which of the remaining candidate edge pixels are immediately considered as edge pixels (strong edges). The high threshold value leads to an image with broken edge contours. Therefore a low thresold value T_l is used to threshold the non-maxima supressed image again. The pixels in this segmented image that are connected to a strong edge are added to the final edge image.

The Canny edge detector gives different results based on the values of σ, T_h and T_l, but the derivative operator used to find the magnitude and how the non-maxima suppression was implemented also affects the final edge segmented image.

Chapter 2

Level Set Method

2.1 Introduction

Surfaces that evolves over time can be difficult to represent. Taking the surface in figure 2.1 as an example, assume that the red surface is heat and the arrows on the interface as the direction of its movement, which is normal to the interface itself. One way to represent the propagation of this interface is by the function $y = f(x, t)$, where t represents time and x, y are coordinates. The problem with this representation is that it cannot represent every concievable shape of the interface. If for instance the shape of the inerface has more y coordinates for a particular x coordinate (which is true for all closed interfaces), the intercane cannot be correctly represented using this notion. A better alternative is to use a parametric equation. The

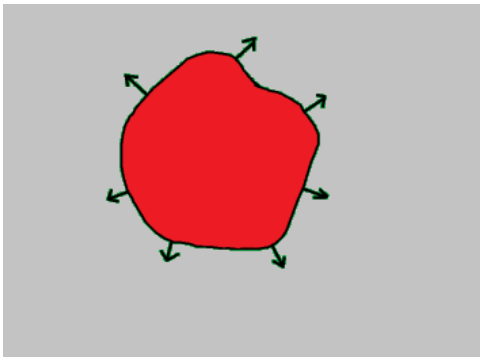


Figure 2.1: Interface of a moving surface.

problem mentioned above would then be solved because the interface would only depend on the time variable t. But parametric representation of evolv-

ing interfaces have its own difficulties. When a surface evolves, the model have to be reparemeterized, which, due to the computional overhead (especially in 3D) add limitations to what kind of shapes a parameterical model can represent effectively. Topological changes, such as splitting or merging parts during the propagation is difficult to represent using parametric models. Sharp corners, distant edges blending together and the complexity of representing boundaries in higher dimensions are some other reasons why an evolving surface is difficult to represent parametrically. A simple example is shown in figure 2.2, the two interfaces have to be represented as a single parametric function when merging and as two seperate againg when they split, and some sort of collison detection must be used to discover when the interfaces merge/split.

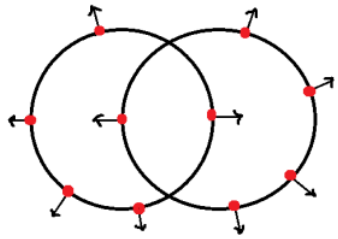


Figure 2.2: Interface evolution difficult to represent parametrically.

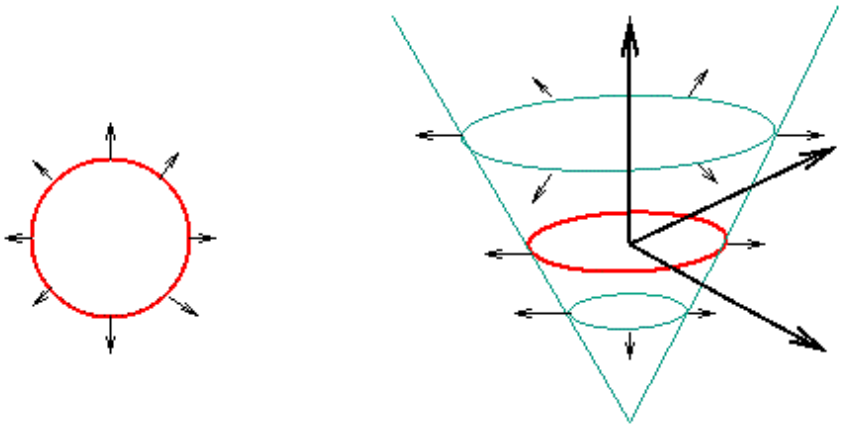
As a solution to all the problems mentioned above Osher and Sethian introduced the level set method in 1988 in [1]. The main idea behind the level set method is to represent the interface of a surface implicitly by using a higher dimensional function. Adding an extra dimension simplifies the problems mentioned above significantly. This higher dimension function is called the level set function, and a 2D interface (a curve) is represented by the 3D level set function

$$\phi(x, y, t) \tag{2.1}$$

where the additional dimension t represents time. Similarly any 3D or higher level function can be represented by a level set function by adding one dimension. At a given time step, the evolving surface/model can be represented as a closed curve by the boundary of the level set at that time step. This representation of the model is called the zero level set and is defined as the set of points where the level set is zero:

$$\Gamma(x, y, t) = \{\phi(x, y, t) = 0\}. \tag{2.2}$$

The initial curve is at the xy-plane, that is, at $\phi(x, y, 0)$. As an example, figure 2.3a depicts a circle with arrows pointing in the direction it is evolving, and figure 2.3b is the cone that represents the corresponding level set function with the start-position in red.



(a) (b)

Figure 2.3: (a): Circle with arrows pointing in direction of movement, (b): Corresponding level set function

Assuming that the zero level set moves in a direction normal to the speed F, then ϕ satisfies the level set equation

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| F \tag{2.3}$$

which is used to update the level set at each time step (iteration). Here $|\nabla \phi|$ represents the gradient of ϕ, and the speed function F describes how each point in the boundary of the surface evolves. The level set method is applied in many different contexts, such as image processing, fluid dynamics and other simulations, and the speed function F depends on the type of problem being considered.

An often used speed function for image segmentation that combines a data term and the mean curvature of the surface is [8, 7]

$$F = \alpha D(I) + (1 - \alpha) \nabla \frac{\nabla \phi}{|\nabla \phi|} \tag{2.4}$$

where $\nabla \cdot (\nabla \phi / |\nabla \phi|)$ is the normal vector that represents the mean curvature term which keeps the level set function smooth. $D(I)$ is the data function

that forces the model towards desirable features in the input data. The free weighting parameter $\alpha \in [0, 1]$ controls the level of smoothness, and I is the input data (the image to be segmented). The smoothing term α restricts how much the curve can bend and thus alleviates the effect of noise in the data, preventing the model from leaking into unwanted areas[7]. This is one of the big advantages the level set method has over classical flood fill, region grow and similar algorithms, which does not have a constraint on the smoothness of the curve.

A simple data function for any point (pixel, voxel) based solely on the input intensity I at that point[8, 7] is:

$$D(I) = \epsilon - |I - T| \tag{2.5}$$

Here T is the central intensity value of the region to be segmented, and ϵ is the deviation around T that should also considered to be inside the region. This makes the model expand if the intensity of the points are within the region $T \pm \epsilon$, and contract otherwise. The data function is gradual, thus the effects of $D(I)$ diminish as the model approaches the boundaries of regions with gray-scale levels within the $T \pm \epsilon$ range [7]. This results in the model expanding faster with higher values of ϵ and slower with lower values.

The level set algorithm is initialized by placing a set of seed points that represents a part inside the region to be segmented. These seed points are represented by a binary mask of the same size as the image to be segmented. This mask is used to compute the signed distance function which ϕ will be initialized to.

2.2 Signed Distance Transform

A distance function $D : \mathbb{R}^3 \to \mathbb{R}$ for a set S is defined as

$$D(r, S) = min(r - S) \quad for\ all\ r \in \mathbb{R}^3 \tag{2.6}$$

If a binary image have one or more objects, a distance function can be used to assign a value for every pixel (or voxel in 3D) that represents the minimum distance from that pixel to the closest pixel in the boundary of the object(s). That is, the pixels in the boundary of an object are zero valued, and all other pixels represent the distance to the boundary as a value. Using a distance transform was the idea of how to initialize ϕ in [1], where it was initialized as $\phi = 1 \pm D^2$. But in [5] it was showed that initializing ϕ to a signed distance function gives more accurate results. Signed distance transforms (SDT) assign for each pixel a value with a positive or negative

sign that depend on whether the pixel is inside or outside the object. The values are usually set to be negative for pixels that are inside an object, and positive for those outside. The pixels of the model, which represents the boundary (the zero level set), have values 0. A binary image containing an object is shown in figure 2.4a (the numbers in this image represent intensity values). Figure 2.4b is the signed distance transform of 2.4a where city-block (manhattan) distance have been used, and figure 2.4c is the signed Euclidean distance transform (SEDT).

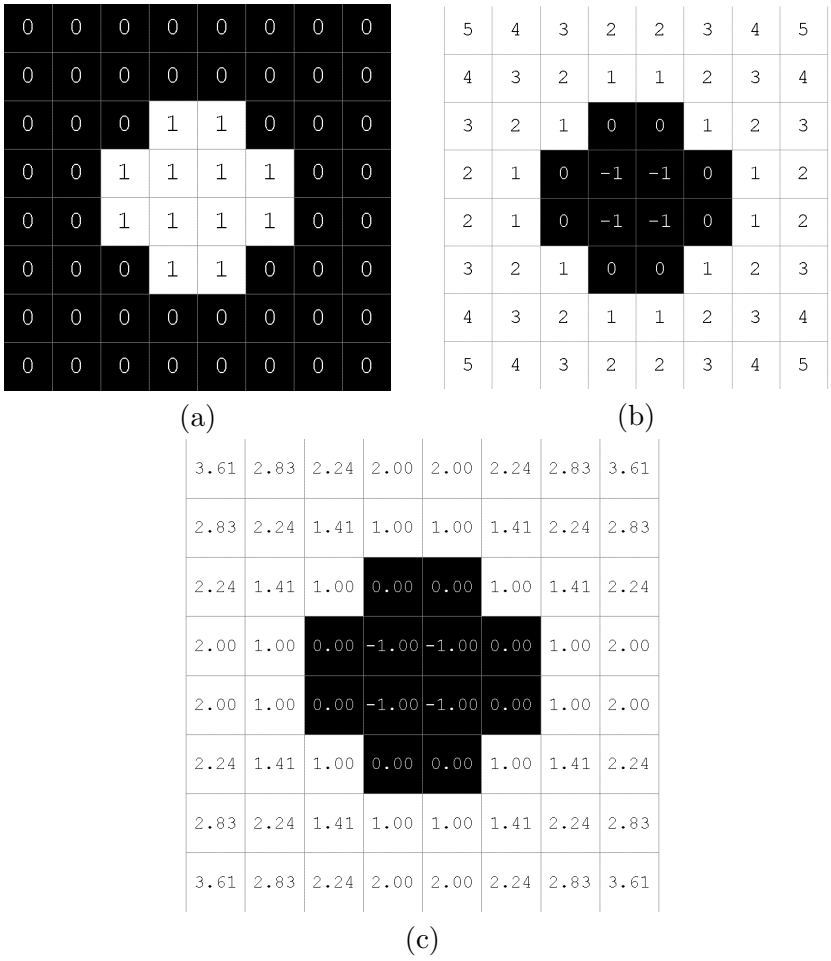


Figure 2.4: (a): Binary image, (b): SDT based on city-block distance, (c): SDT based on euclidean distance

As can be seen from the figures above, using different kind of functions

for the SDT can result in different distances. These differences effects the accuracy of the level set function, which may leads to different end-results of the segmentation, hence, the function used to represent the distance have to be carfully chosen. However, sometimes a less accurate SDT have to be used as a tradeoff for faster computation time.

2.3 Sparse Field

The narrow band method assumes that the computation of the SDT is so slow that it cannot be computed for every iteration (time step). The sparse field method introduced in [6] uses a fast approximation of the distance transform that makes it feasible to compute the neighborhood of the level set model for each iteration. In the sparse field method the idea of using a thin band is taken to the extreme by working on a band that is only one point wide. The points adjacent to the level set are called active points, and all of them together are referred to as the active set. At each iteration only a thin layer of points near the active set are visited and updated. Using only the active points to compute the derivatives would not give sufficient accuracy. Because of this, the method extends out from the active points in small layers to create a neighborhood that is precisely the width needed to calculate the derivatives needed.

Several advantages to this approach are mentioned in [6]. No more than the precise number of calculations to find the next position of the zero level set surface is used. The number of points being computed is so small that a linked-list can be used to keep track of them. This also results in that only those points whose values control the position of the zero level set surface are visited at each iteration. A disadvantage of the narrow band method is that the stability at the boundaries of the band have to be maintained (e.g. by smoothing) since some points are undergoing the evolution while other neighbouring points remain fixed. The sparse field method avoid this by not letting any point entering or leaving the active set affecting its value. A point enters the active set if it is adjacent to the model. As the model evolves, points in the active set that are no longer adjacent to the model are removed from the active set. This is done by defining the neighborhoods of the active set in layers and keeping the values of points entering or leaving the active set unchanged. A layer is a set of pixels represented as L_i where i is the city-block distance from the active set. The layer L_0 represents the active set, and $L_{\pm 1}$ reprsents pixels adjacent to the active set on both sides. Using linked lists to represents the layers and arrays (matrices) to represent distance values makes the algorithm very efficient. The exact steps of the

sparse field algorithm can be found in [6].

The sparse field algorithm is based on an important approximation, it assumes that points adjacent to the active points undergoes the same change in value as their nearby active set neighbors. But despite this, the errors introduced by the sparse field algorithm are no worse than many other level set algorithms. Since only those grid points whose values are changing (the active points and their neighbors) are visited at each time step the growth computation time is d^{n-1}, where d is the number of pixels in along one dimension of the image. This is the same as for parameterized models where the computation times increase with the resolution of the domain, rather than the range.

2.4 Discretization by upwinding and difference of normals

To use the level set method in image processing it have to be discretized, but simple forward finite difference schemes cannot be used because such schemes tends to overshoot and are unstable. To overcome this problem the up-winding scheme was proposed in [1]. To avoid the overshooting problems associated with forward finite differences the up-winding scheme uses one-sided derivatives that looks in the up-wind direction of the moving interface. Let ϕ^n and F^n represent the values of ϕ and F at some point in time t^n. The updating process consist of finding new values for ϕ at each point after a time interval Δt. The forward Euler method is used to get a first-order accurate method for the time discretization of equation 2.3, given by (from [4])

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + F^n \cdot \nabla \phi^n = 0 \tag{2.7}$$

where ϕ^{n+1} is ϕ at time $t^{n+1} = t^n + \Delta t$, and $\nabla \phi^n$ is the gradient at time t^n. This equation is expanded as follows (for three dimensions):

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + u^n \phi_x^n + v^n \phi_y^n + w^n \phi_z^n = 0, \tag{2.8}$$

where the techniques used to approximate the $u^n \phi_x^n$, $v^n \phi_y^n$ and $w^n \phi_z^n$ terms can be applied independently in a dimension-by-dimension manner [4]. When looking at only one dimension (for simplicity), the sign of u^n would indicate whether the values of ϕ are moving to the right or to the left. The value u^n can be spatially varying, hence by looking at only one point x_i in addition

to only look at one dimension, equation 2.8 can be written as

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + u_i^n (\phi_x)_i^n = 0, \tag{2.9}$$

where $(\phi_x)_i^n$ denotes the spatial derivative of ϕ at point x_i at time t^n. The values of ϕ are moving from left to right if $u_i > 0$, thus the points to the left for x_i are used to determine the value of ϕ at point x_i for the the next time step. Similarly, if $u_i < 0$ the movement is from right to left, and the points to the right of x_i are used. As a result, ϕ_x is approximated by the derivative function D_x^+ when $u_i < 0$ and D_x^- when $u_i > 0$. When $u_i = 0$ the term $u_i(\phi_x)_i$ equals zero, and approximation is not needed. Extending this to three dimensions, the derivatives used to update the level set equation are

$$\begin{gathered} D_x = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2} \quad D_y = \frac{\phi_{i,j+1,k} - \phi_{i,j-1,k}}{2} \quad D_z = \frac{\phi_{i,j,k+1} - \phi_{i,j,k-1}}{2} \\ D_x^+ = \phi_{i+1,j,k} - \phi_{i,j,k} \quad D_y^+ = \phi_{i,j+1,k} - \phi_{i,j,k} \quad D_z^+ = \phi_{i,j,k+1} - \phi_{i,j,k} \\ D_x^- = \phi_{i,j,k} - \phi_{i-1,j,k} \quad D_y^+ = \phi_{i,j,k} - \phi_{i,j-1,k} \quad D_z^+ = \phi_{i,j,k} - \phi_{i,j,k-1} \end{gathered} \tag{2.10}$$

which is taken from the appendix of [7]. This is a *consistent* finite difference approximation to the level set equation in 2.3, because the approximation error converges to zero as $\Delta t \to 0$ and $\Delta x \to 0$ [4]. In addition to being consistent, it also have to be *stable* in order to get the correct solution. Stability guarantees that small errors in the approximations are not amplified over time. The stability can be enforced using the Courant-Friedreichs-Lewy (CLF) condition which says that the numerical wave speed $\frac{\Delta x}{\Delta t}$ must be greater than the physical wave speed $|u|$,

$$\Delta t = \frac{\Delta x}{max\{|u|\}}, \tag{2.11}$$

where $max\{|u|\}$ is the largest value of $|u|$ on the model.

The gradient $\nabla\phi$ is approximated to either $\nabla\phi_{max}$ or $\nabla\phi_{min}$ depending on whether the speed function for a given point $F_{i,j,k}$ is positive or negative,

$$\nabla\phi = \begin{cases} ||\nabla\phi_{max}||_2 & F_{i,j,k} > 0 \\ ||\nabla\phi_{min}||_2 & F_{i,j,k} < 0 \end{cases} \tag{2.12}$$

where $\nabla\phi_{max}$ and $\nabla\phi_{min}$ is given by (from [7])

$$\nabla\phi_{max} = \begin{bmatrix} \sqrt{max(D_x^+, 0)^2 + max(-D_x^-, 0)^2} \\ \sqrt{max(D_y^+, 0)^2 + max(-D_y^-, 0)^2} \\ \sqrt{max(D_z^+, 0)^2 + max(-D_z^-, 0)^2} \end{bmatrix} \tag{2.13}$$

$$\nabla\phi_{min} = \begin{bmatrix} \sqrt{min(D_x^+, 0)^2 + min(-D_x^-, 0)^2} \\ \sqrt{min(D_y^+, 0)^2 + min(-D_y^-, 0)^2} \\ \sqrt{min(D_z^+, 0)^2 + min(-D_z^-, 0)^2} \end{bmatrix} \tag{2.14}$$

The curvature term $\nabla \cdot (\nabla\phi/|\nabla\phi|)$ of the speed function F is discretized using the difference of normals method. The second order derivatives are computed first:

$$\begin{aligned} D_x^{+y} &= (\phi_{i+1,j+1,k} - \phi_{i-1,j+1,k})/2 \quad D_x^{-y} = (\phi_{i+1,j-1,k} - \phi_{i-1,j-1,k})/2 \\ D_x^{+z} &= (\phi_{i+1,j,k+1} - \phi_{i-1,j,k+1})/2 \quad D_x^{-z} = (\phi_{i+1,j,k-1} - \phi_{i-1,j,k-1})/2 \\ D_y^{+x} &= (\phi_{i+1,j+1,k} - \phi_{i+1,j-1,k})/2 \quad D_y^{-x} = (\phi_{i-1,j+1,k} - \phi_{i-1,j-1,k})/2 \\ D_y^{+z} &= (\phi_{i,j+1,k+1} - \phi_{i,j-1,k+1})/2 \quad D_y^{-z} = (\phi_{i,j+1,k-1} - \phi_{i,j-1,k-1})/2 \\ D_z^{+x} &= (\phi_{i+1,j,k+1} - \phi_{i+1,j,k-1})/2 \quad D_z^{-x} = (\phi_{i-1,j,k+1} - \phi_{i-1,j,k-1})/2 \\ D_z^{+y} &= (\phi_{i,j+1,k+1} - \phi_{i,j+1,k-1})/2 \quad D_z^{-y} = (\phi_{i,j-1,k+1} - \phi_{i,j-1,k-1})/2 \end{aligned} \tag{2.15}$$

Then these derivatives are used to compute the normals n^+ and n^- in equation 2.16, which is used to compute the mean curvature H in equation

2.17 taken from [7].

$$n^+ = \begin{bmatrix} \frac{D_x^+}{\sqrt{(D_x^+)^2+(\frac{D_y^{+x}+D_y}{2})^2+(\frac{D_z^{+x}+D_z}{2})^2}} \\ \frac{D_y^+}{\sqrt{(D_y^+)^2+(\frac{D_x^{+y}+D_x}{2})^2+(\frac{D_z^{+y}+D_z}{2})^2}} \\ \frac{D_z^+}{\sqrt{(D_z^+)^2+(\frac{D_x^{+z}+D_x}{2})^2+(\frac{D_y^{+z}+D_y}{2})^2}} \end{bmatrix}$$

$$n^- = \begin{bmatrix} \frac{D_x^-}{\sqrt{(D_x^-)^2+(\frac{D_y^{-x}+D_y}{2})^2+(\frac{D_z^{-x}+D_z}{2})^2}} \\ \frac{D_y^-}{\sqrt{(D_y^-)^2+(\frac{D_x^{-y}+D_x}{2})^2+(\frac{D_z^{-y}+D_z}{2})^2}} \\ \frac{D_z^-}{\sqrt{(D_z^-)^2+(\frac{D_x^{-z}+D_x}{2})^2+(\frac{D_y^{-z}+D_y}{2})^2}} \end{bmatrix} \tag{2.16}$$

$$H = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} = \frac{1}{2}[(n_x^+ - n_x^-) + (n_y^+ - n_y^-) + (n_z^+ - n_z^-)] \tag{2.17}$$

Finally, the level set equation is updated as

$$\phi(t + \Delta t) = \phi(t) + \Delta t F|\nabla\phi|. \tag{2.18}$$

2.5 Parallel implementation in GPU

A huge disadvantage with the level set method for segmentation is that it is very slow when working with big data volumes in 3D space. Implementations of level set algorithms for 3D in the graphical processing unit (GPU) parallelizes the level set method and makes it much faster. One of the first GPU based 3D implementations of the level set method was by Lefohn et al. in [8] in 2003. In this paper a modified sparse field level set method was implemented for the GPU using graphic APIs such as OpenGL and DirectX. In the past few years general purpose GPUs have made implementing level set methods and other non-graphical tasks in GPUs much easier. In [9] some simple medical segmentation algorithms was implemented using NVIDIAs CUDA technology, and in [10] CUDA was used to implement the level set method.

Chapter 3

Narrow Band - for lite?

3.1 Introduction

When working with the level set of a single interface a huge drawback with the originally proposed level set method is the computional inefficiency due to computing over the whole domain of ϕ. As a solution to this problem Adalstein and Sethian proposed the narrow band method in 1994[2]. The narrow band looks at the interface of a single level set instead of the whole domain, and thereby decreases the computational labor of the standard level set method for propagating interfaces considerably. Another reason the narrow band was proposed are problems where the velocity field is only given on the interface. In such cases the construction of an appropriate speed funcion for the entire domain made use of the classical level set method a significant modeling problem.

3.2 Overview of the Narrow Band method

Unlike the original level set method, which describe the evolution of an embedded family of contours, the narrow band works with only a single surface model[6]. That is, instead if calculating ϕ over the whole domain it focuses only on a small part surrounding the surface. There are many cases in which the description of the evolution of only one surface in the domain is needed, and in such cases the narrow band method operates much faster while delivering the same results. The method ignores points that are far away from the zero level set at each iteration and only looks at the points within a narrow band. This is possible because points far away from the zero level set do not have any influence on the result. That is, only the area of ϕ where $\phi \approx 0$ is important for accurate representation of the level set.

The narrow band method restrict the computation to a thin band of points by extending out approximately k points from the zero level set (shown in figure 3.1), and an embedding of the evolving interface is constructed via a signed distance transform. All points outside the band is set to constant values to indicate that they are not within the band and thus should not be used in the computation. This reduces the number of operations at each iteration from $O(n^{d+1})$ to $O(nk^d)$ [2] where d is the number of dimensions and n is the (average) number of points in one dimension. The points within

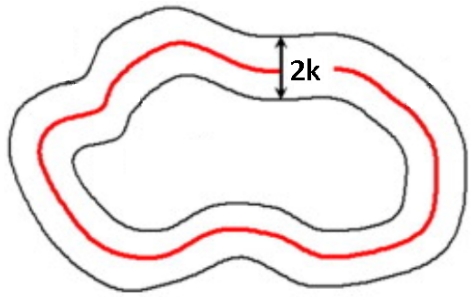


Figure 3.1: The narrow band extending out with a width of k from the level set.

the band is used to calculate the distance function and then to initialize ϕ to the signed distance. As the zero level set evolves, ϕ will get further and further away from its initialized value as signed distance. As this happens ϕ must be ensured to stay within the band. One way to do this would be to make a new band for each iteration. But determining which points are to be inside the band, and deciding how to take the differentials at the edge points makes the reconstruction process of the band time consuming. Thus a given band is used for several iterarions with the same initialization of ϕ. When the interface gets close to the band it has to be reset from the current position of the zero level set and ϕ must be reinitialized. Reinitializing ϕ at every iteration takes too much time and the alternative task of finding out if any of the pixels in the zero level set are getting close to the edge of the band (for every iteration) also takes time. Hence, ϕ is usually just reinitialized after a fixed number of iterations, which keeps ϕ approximately equal to the SDT.

As metioned in the section about signed distance transforms, different SDTs can lead to slightly different end-results and must be carefully chosen. If the technique used to approximate ϕ to a signed distance function is too sensitive, ϕ needs to be reinitialized accurately and often. If it is less sensitive, it does not have to be initialized so often and a less accurate

method can be used, but this may lead to noisy features [4].

The narrow band, despite its improvements over the original level set method, is not optimal. The band used being too wide is the main reason. Even if k=2 is enough to compute the necessary derivatives, the band have to be of a certain width (k=12 was used in the test of topological changes in [2]) because of two competing computional costs[6]. The first is the cost of computing the position of the curve and the SDT, and reset the band. The second is the cost of computing the evolution process over the entire band.

Chapter 4

Sparse Field

4.1 Theory

//Theory eller introduction som seksjons-overskrift?

The narrow band method assumes that the computation of the SDT is so slow that it cannot be computed for every iteration. The sparse field method introduced in [6] uses a fast approximation of the distance transform that makes it feasible to compute the neighborhood of the level set model for each iteration. In the sparse field method the idea of using a thin band is taken to the extreme by working on a band that is only one point wide. The points (immediately?) adjacent to the level set are called active points, and all of them together are referred to as the active set. At each iteration only a thin layer of points near the active set are visited and updated. Using only the active points to compute the derivatives would not give sufficient accuracy. Because of this, the method extends out from the active points in small layers to create a neighborhood that is precisely the width needed to calculate the derivatives needed.

Several advantages to this approach are mentioned in [6]. No more than the precise number of calculations to find the next position of the zero level set surface is used. The number of points being computed is so small that a linked-list can be used to keep track of them. This also results in that only those points whose values control the position of the zero level set surface are visited at each iteration.

A disadvantage of the narrow band method is that the stability at the boundaries of the band have to be maintained (e.g. by smoothing) since some points are undergoing the evolution while other neighbouring points remain fixed. The sparse field method avoid this by not letting any point entering or leaving the active set affecting its value. A point enters the

active set if it is adjacent to the model. As the model evolves, points in the active set that are no longer adjacent to the model are removed from the active set. This is done by defining the neighborhoods of the active set in layers and keeping the values of points entering or leaving the active set unchanged. A layer is a set of pixels represented as L_i where i is the city-block distance from the active set. The layer L_0 represents the active set, and $L_{\pm 1}$ reprsents pixels adjacent to the active set on both sides. Using linked lists to represents the layers and arrays (matrices) to represent distance values makes the algorithm very efficient.

The sparse field algorithm is based on an important approximation, it assumes that points adjacent to the active points undergoes the same change in value as their nearby active set neighbors. But despite this, the errors introduced by the sparse field algorithm are no worse than many other level set algorithms. Since only those grid points whose values are changing (the active points and their neighbors) are visited at each time step the growth computation time is d^{n-1}, where d is the number of pixels in along one dimension of the image. This is the same as for parameterized models where the computation times increase with the resolution of the domain, rather than the range.

The Up-Winding scheme gives the curvature in an area surrounding a point in the active set. This scheme uses both first and second order derivatives, and to calculte them it needs a 3x3x3(3D) grid of points surrounding the point for which it is calculating the speed.

4.2 Implementation

We implemented the Sparse Field method using C/C++. Out implementation used lists, a C++ struct, to maintain the different

Cutouts and working progress stuff:

The neighbours of the active set are tracked in lists, L1, L2, L-1 and L-2, and in addition we have the active set L0. These lists keep track of the coordinates of where the different layers reside in each iteration.

Chapter 5

Sparse Field - Implemented code

5.1 Introduction

The sparse field level set method was implemented in C++ for the project, and the implemented code is mainly based on the pseudocode in [11], which again is based on Whitaker's introduction to the sparse field method in [6]. The implemented code can be found in appendix B, and its pseudocode can be found in appendix A. In this chapter the pseudocode in [11] will be explained first, along with how it works. Secondly, the differences between the pseudocodes in appendix A and [11] will be described. And finally there will be a detailed explanation of the implemented code.

5.2 TODO

As previously mentioned, the sparse field method can be implemented using linked lists to hold the pixels being used in the calculations. Pixels in this context does not mean the pixels in the original or segmented image, but the points in the matrix that represents the ϕ.These pixels are seperated into five layers, each represented by a linked list. One of the lists holds the active points, i.e the zero level set, and is referred to as the Lz list. The rest of the needed pixels are seperated according to their closeness to the pixels in Lz and which side of the Lz pixels they are located. The Ln1 list contains the pixels that are adjacent to Lz pixels on the inside of the object being segmented. Similarly Lp1 contains pixels that are adjacent, but on the outside. All pixels that are adjacent to those in Ln1 except for those

in Lz are elements in the Ln2 list, and similarly the ones adjacent to Lp1 on the opposite side of Lz are part of Lp2. This becomes more clear when looking at table 5.1 and figure 5.1.

List Name	Range
Lz	[-0.5, 0.5]
Ln1	[-1.5, -0.5¿
Lp1	¡0.5, 1.5]
Ln2	[-2.5, -1.5¿
Lp2	¡1.5, 2.5]

Table 5.1: Range of lists used in [11]

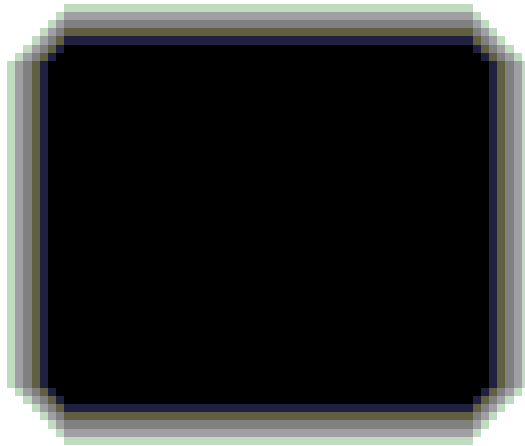


Figure 5.1: Label image: image showing the different layers under segmentation.

Figure 5.1 represents the 5 different layers with different colors. The black colored part is defined to be inside the object being segmented, the white part as outside, and these two parts are not used in the computation for the current iteration. The dark blue pixels around the dark part are the pixels contained in Ln2, and the brown pixels are those on Ln1. The dark-purple colored pixels are Lz elements, light-purple are Lp1 and light-blue are pixels in Lp2. This type of image will henceforth be referred to as the label image, because it shows the labels of the image being segmented.

5.2.1 Datastructures and types used - elr noe lignende

In addition to the five layers represented as five lists, there are some other structures implemented in the code. One of them is the label image described above, which is used to track where the pixels containing the different layers are on the image domain. Given a pixel, to find out which layer (if any) that pixel is a member of, a simple lookup to the label is enough. Another excellent feature of the label image is that it can be used to verify that all the layers are correclty aligned and if there are any pixels of any layers that are poorly placed. It can thus be used to find artifacts that might have resulted from code errors. An example of a label image (zoomed in) which clearly states that there is something wrong with how the layers are handled in the code is shown in figure 5.2. How the label image actually should have been is depicted in figure 5.3.

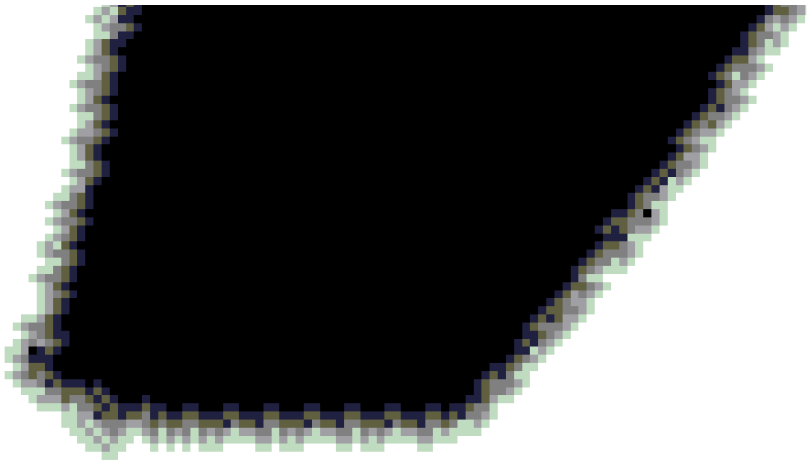


Figure 5.2: A label image with pixels in places they should not be.

The level set (ϕ) is also represented as an image...TODO

5.3 Forskjeller fra vr kode og pseudokoden til lankton

BlaBla ... blabal ... . Table X1 describes the ranges used for the different lists in the implemented code. By comparing table X with table X1 a small difference in the list ranges can be seen (Nevn hvilke forandiringer her). These small changes in the ranges of the lists may seem unsignificant but they are quite important. Figures X2 and X3 depicts the segmentation

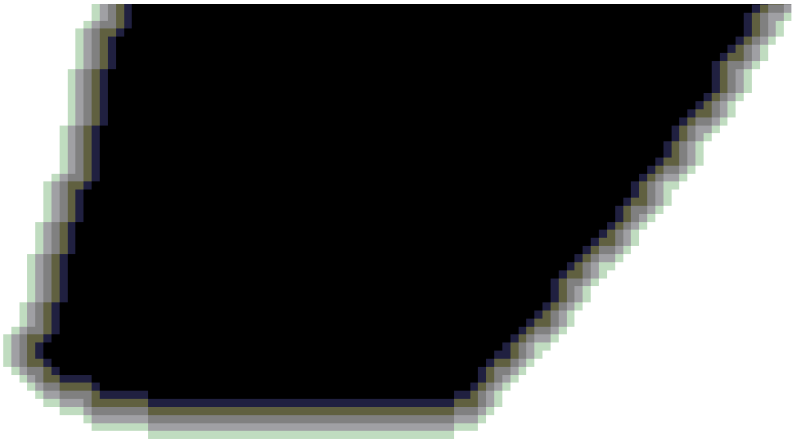


Figure 5.3: How the label image should have been.

result of the same original input image with equal number of iterations completed, with the ranges from table X and X1 used resprectively.

List name	Range
Lz	[-0.5, 0.5¿
Ln1	[-1.5, -0.5¿
Lp1	[0.5, 1.5¿
Ln2	[-2.5, -1.5¿
Lp2	[1.5, 2.5¿

Table 5.2: Range of lists used in the implementation

Som vi ser s ser X3 mye bedre ut blabalba.

5.4 Om koden vr

TODO

5.5 Problems met

5.6 Performance

Nevn at vi frst brukte vector i 2D versjonen, men at det gikk veldig tregt i 3D versjonen. Forskjellige ting ble brukt for minske farten, deriblant

bruke float istedenfor double, noe som senket kte farten med X. Vi brukte ogs OpenMP som en midlertidig lsning for speedup da vi testet koden p forskjellige datasett. Grunnen til at vi brukte vector var fordi det var nevnt i [11] p en mte alik at det kunne misforsts vre linked-list. Ved begynnelsen av implementeringen ble det ikke tenkt gjennom at vector kanskje ikke var den beste lsningen bruke. Men siden en full segmentering av et 512*512*265 bilde tok over 20min begynte vi se nrmere p hva som kan endres p. Fant da ut at det er stor forskjell mellom vector og list i C++. (Nevn noen av de strste forskjellene her.) Endret s til bruke list istedenfor vector, og fikk en speed increase p en faktor av 10 i 3D. MER!! (Nevn CUDA ogs hvis det blir implementert.. burde vel ha eget kapittel elr section for det.)

Bibliography

[1] S. Osher & James A. Sethian, *Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulation.* Journal of computational physics 79.1, 1988.

[2] David. Adalsteinsson & James A. Sethian, *A fast level set method for propagating interfaces.* Journal of Computational Physics, 1994.

[3] Dzung L. & Chenyang Xu & Jerry L. Prince, *A Survey of Current Methods in Medical Image Segmentation.* Annual review of biomedical engineering 2.1, 2000.

[4] Stanley Osher & Ronald Fedkiw, *Level set methods and dynamic implicit surfaces.* Vol. 153. Springer, 2002.

[5] W. Mulder & S. Osher & James A. Sethian, *Computing interface motion in compressible gas dynamics.* Journal of Computational Physics 100.2, 1992.

[6] Ross T. Whitaker, *A level-set approach to 3D reconstruction from range data.* International Journal of Computer Vision 29.3, 1998.

[7] Aaron E. Lefohn & Joe M. Kniss & Charles D. Hansen & Ross T. Whitaker, *A streaming narrow-band algorithm: Interactive computation and visualization of level sets.* IEEE Transactions on Visualization and Computer Graphics, 2004.

[8] Aaron E. Lefohn & Joshua Cates & Ross T. Whitaker, *Interactive, GPU-based level sets for 3D segmentation.* Medical Image Computing and Computer-Assisted Intervention, 2003.

[9] Lei Pan & Lixu Gu & Jianrong Xu, *Implementation of medical image segmentation in CUDA.* Information Technology and Applications in Biomedicine, 2008.

[10] M. Roberts & J. Packer & Mario C. Sousa & Joseph R. Mitchell, *A work-efficient GPU algorithm for level set segmentation*. Proceedings of the Conference on High Performance Graphics, pp. 123-132, Eurographics Association, 2010.

[11] Shawn Lankton, *Sparse Field Methods - Technical Report*. Georgia institute of technology, 2009.

Appendix A

Sparse Field - Pseudocode

The code written for this thesis (which can be found in appendix B) is based on the following pseudocode. This pseudocode originates from [11], with a different speed function and some other modifications.

TODO: skriv pseudocoden for koden vr her (ikke skriv direkte av lankton09)

Appendix B

Implemented Sparse field method

TODO: Kopier inn all koden her.