On continuous maximum flow image segmentation algorithm

PhD. Thesis

Laszlo Marak[1]

ESIEE PARIS
Departement Informatique

2012-07-20

[1] PGP: A812 3708 B2E1 3B42 500E D50C 49A0 88FA C060 3F65,
Université Paris-Est, Département d'Informatique Gaspard-Monge,
Équipe A3SI, ESIEE Paris, Cité Descartes, BP 99, F-93162 Noisy-le-Grand Cedex, France
ujoimro@gmail.com

Ce n'est pas tous les jours qu'on a besoin de nous. Non pas à vrai dire qu'on ait précisément besoin de nous. D'autres feraient aussi bien l'affaire, sinon mieux.

Samuel Beckett: En attendant Godot

Copyright

Contents

Journal publications by the author

1. L. Marak, O. Tankyevych, and H. Talbot, "Continuous maximum flow segmentation method for nanoparticle interaction analysis," *Journal of Microscopy*, vol. 244, no. 1, pp. 59–78, 2011.
2. O. Le Bihan, P. Bonnafous, L. Marak, T. Bickel, S. Trépout, S. Mornet, F. De Haas, H. Talbot, J. Taveau, and O. Lambert, "Cryo-electron tomography of nanoparticle transmigration into liposome," *Journal of structural biology*, 2009.

Conference publications by the author

1. Ecole Normale Supérieure, *The Pink Image Processing Library*, (Paris), ESIEE Engineering, 2011-08-27. http://www.pinkhq.com.
2. L. Marak, J. Cousty, L. Najman, and H. Talbot, "4d morphological segmentation and the miccai lv-segmentation grand challenge," 07 2009.
3. O. Tankyevych, L. Marak, H. Talbot, and P. Dokladal, "Segmentation of 3d nano-scale polystyrene balls," in *International symposium on Mathematical Morphology 2007*, (Rio de Janeiro, Brazil), 2007.

Abstract

In recent years, with the advance of computing equipment and image acquisition techniques, the sizes, dimensions and content of acquired images have increased considerably. Unfortunately as time passes, there is a steadily increasing gap between the classical and parallel programming paradigms and their actual performance on modern computer hardware. In this thesis we consider in depth one particular algorithm, the continuous maximum flow computation. We review in detail why this algorithm is useful and interesting, and we propose efficient and portable implementations on various architectures. We also examine how it performs in the terms of segmentation quality on some recent problems of materials science and nano-scale biology.

Introduction

In this thesis we examine in depth the continuous maximum flow algorithm. This algorithm uses a separable evolutionary non-linear differential equation to find optimum curve and surface integrals on arbitrary integral measures. The algorithm is a continuous version of the well-known maxflow-mincut graph algorithm, which has been very popular in recent years in computer vision applications. It is able to find the optimum and avoid all the local minima of the formulation. The solution of the algorithm is isotropic. For the reason of separability the differential equation is extensible to accommodate fields (images) in arbitrary dimensions.

The solution of the differential equation is calculated by the broken line approximation method. The equation can be iterated on locally, so this gives space to a fine-grained parallelization in the implementations. Common architectures, however, are specialized in sequential execution, so a parallel implementation is not straight-forward. In this thesis we propose efficient parallel implementation on the x86 architecture. The details of the implementation as well as the challenges of the architecture are discussed in section 3.3. There are also other architectures, some of which have explicitly been designed to execute fine-grained parallel implementations. In section 3.4. we also propose an implementation on the GPGPU architecture. According to the profiler, the implementation saturates the GPGPU system up to 90%. With the implementations being as cross platform as possible we test both the x86 and the GPGPU implementation on the Cell Broadband Engine Architecture (CBEA). The details of the portability are discussed in section 3.4.3. With the optimal implementations we can achieve improvement of up to two orders of magnitude compared with an optimized but sequential implementation.

Finally in chapter 4. we propose segmentation methods using the maximum flow algorithm to segment silica beads, lipidic membranes and fibrous

materials. The images were reconstructed from a series of 2D tomographic projections with growing tilt degree.

However, due to physical sample preparation constraints, the range of projection angles is limited to 120 degrees (due to the thickness of the grid), and so some parts of the resulting images are not reconstructed precisely. The result of the microscopy are three-dimensional images, however the signal-to-noise ratio becomes very low near object's poles due to the so-called *missing wedge effect*. In the case of the silica beads we use the Hough transform to find the seeds for the objects, whereas we propose a special measure to avoid the acquisition noise. For the lipidic membranes, we propose a method for finding the seeds of the segmentation in case of higher noise-than-signal ratios. The seeds are derived from the cumulated data using incomplete slice segmentations. After the segmentation we carry out measurements of the physical properties of the objects. These measurements include curvature estimation for the extent of particle attraction and distance measurements.

We demonstrate that the continuous maximum flow algorithm is particularly useful in the segmentation of 3D nanotomographic and medical images. It is robust, little sensitive to noise. We show that the algorithm is parallelizable and we propose efficient implementations on wide variety of hardware.

Part I
Contemporary Image Segmentation

The general problem of image segmentation is coeval with the digital image acquisition. Segmentation can be defined as the problem of partitioning the image into regions of interest. This general subject covers a wide variety of potential applications [1]. Some applications, like bar-code recognition, are common, but not always appreciated as image segmentation problems. Others, like typical industrial applications, where cameras are installed on the production lines are also not as known to the general public. Cameras can record the products during fabrication. The acquired images can then help filter the end products for defects [2]. They can also guide the machines during the production. The product can have minor variations even within the same series. Automatic welders, for example, can be guided by the segmented image [3]. Cameras are frequently used in production of printed circuits. After the segmentation of the wire track on the circuit, broken wires can be detected by comparing the segmentation to the blueprint.

In materials science a sample of material is analyzed using X-ray or electron tomography, then important characteristics, like material degradation or the extent of corrosion can be automatically assessed. The quality of a composite material, like concrete, can be judged by detecting the proportion of its components. For this we first need to segment each type of object in the image.

In biological or agricultural applications companies often detect the distribution of the seed-size [4] or the proportion of broken seeds in the sample.

In medical application image acquisition is the general preferred way of diagnostics as it is non invasive. An example is mammographic images [5] or the diagnosis of a broken bone. White cell detection in blood samples is also a widely adopted technique [6]. With the development of 3D or 4D acquisition techniques, segmentation has become an even more important part of the procedure than with 2D images. 3D segmentation is generally carried out on 2D slices. In [7] authors demonstrate that this approach can be very time consuming. Moreover from 2D slices many of the typical 3D characteristics like topology or connection hierarchy are difficult to enforce and validate. Applications include tooth restoration [8], heart chamber segmentation, etc.

In the research area image segmentation has also been widely adopted. Image segmentation is both used as a research tool and research is carried out to improve existing segmentation techniques. For image data can represent huge quantities as well as 3D or 4D images, relying on computer vision techniques is in many cases more a necessity than a convenience.

Chapter 1
Image Segmentation Problems

From the mathematical and algorithmic point of view, image segmentation is a difficult, open problem. The problem can be divided in two general parts. As we mentioned in the introduction, segmentation is equivalent to partitioning the image into regions of interest. In the simplest case (in the sense of region cardinality) there are only two regions: the object and the background. The first problem is, therefore, to search conditions for an optimal partitioning. In other words, given two partitioning of the image we need to be able to decide which of them is a better segmentation. Human validation is sometimes possible for the end result, but it is practically impossible to incorporate the human factor in the mathematical formalisms.

Several models have been proposed, but most segmentation criteria try to formalize the properties of the Human Visual System (HVS) model. In this model, the objects are limited by edges, and objects appear in front of some background. The HVS can interpolate missing parts of the object or filter the noise in the image. This model has to deal with some aspects of human vision, which are not yet fully understood. Humans are good at edge interpolation, motion detection, face and handwriting recognition, color vision and raster vision[1].

The criteria for image segmentation can be as simple as the intensity of the light in the image. A simple threshold can provide sufficient, though not sophisticated results in some cases and it is also used as the last step of many image processing frameworks. More complicated criteria can be based on the connexity, the area, position and shape or the border of the objects to segment as well as any combination of these.

Once that the mathematical criteria have been formulated, the second problem is to propose algorithms for partitioning the image in order to meet the given criteria. Methods can include starting with a degenerated partition[2]

[1] we can distinguish many of different colors, but the number of different color sensors (cones) is much lower then the number of brightness sensors (rods)

[2] In a degenerated partition the whole image is either in the partition of the object or the whole image is in the partition of the background.

and successively erasing points until the searched criteria is met [9]. Other methods try to reformulate, project or convert the image field. After the reformulation the image can be thresholded into the optimal partition [10].

1.1 Minimum Surfaces

In this section we collect the basic notions and problems of image segmentation and the contemporary work designated to address these problems. In this thesis we deal with continuous image segmentation of grayscale images. We perceive the image I as a continuous $\mathbb{R}^d \to \mathbb{R}$ function. Here d represent the image dimensionality. The images we work with are often noisy and many feature acquisition artifacts. We seek the objects in these images which are limited by their borders. We will also demonstrate that segmentation is closely related to image denoising. Indeed, both deal with determining regions of the image that are similar. In the following sections we are going to collect the basic ideas and problems of continuous image processing.

Several authors have suggested continuous formulation for image processing [11, 12]. They have also demonstrated why is it important to use edges for segmentation. Scientists often use this formulations for image processing problems. Let us consider a digital grayscale image of size $[i_0, i_1, \ldots, i_{d-1}]$. In the computer memory representation the image I is a function $I : 0 \ldots i_0 - 1 \times 0 \ldots i_1 - 1 \times \cdots \times 0 \ldots i_{d-1} - 1 \to \mathbb{R}$. In the continuous domain, we interpret the same image as a function $I : \mathbb{R}^d \to \mathbb{R}$, for which $I[x_0, x_1, \ldots, x_{d-1}] = I(x_0, x_1, \ldots, x_{d-1}) \forall x_0, \ldots, x_{d-1}$. By abuse of notation where there is no confusion, we refer to both functions as I. Now we are going to describe the problems, that we can define with the continuous formulation.

The simple idea is to find an object in I. The object would be delimited by edges. The edges can, however, be damaged or incomplete. We generate a measure g which in every point $\overline{\mathbf{x}}$ will represent the *penalty* $g(\overline{\mathbf{x}})$ of the edge containing the point x. One intuitive choice could be the inverse of the gradient $(\frac{1}{\nabla I})$. We are now looking for a surface containing as many low intensity points as possible.

More generally we would be looking for an object in I delimited by some property in g. This property can be incomplete[3]. The general idea is that there should be no constrain on the measure we would like to consider. It can be based on any property, like edge detector or based on texture intensities. In any case we convert the image I in image g. The intensities of $g(\overline{\mathbf{x}})$ will represent penalty for $\overline{\mathbf{x}}$ to be on the border of a *partition*. We are looking for the hypersurface[4] which contains *low intensity parts* while still remaining a surface. We call this surface the minimum surface. Mathematically we can

[3] not defined or not available everywhere

[4] The hyper-surface is the $d - 1$ dimensional surface. It is a curve for 2D images and a surface for 3D images.

describe it as the result of the geodesic active contour (GAC)method.

$$\min_{S \subseteq C \subseteq \Omega} \oint_C g \mathrm{d}s \tag{GAC}$$

Here S is a subset of the image, which we constraint to be in the object. The (GAC) functional represents the optimal curve on the field. The methodology of curve integral minimization is well established in mathematics of differential equations. As a simple example we can consider a unit square in $\mathbb{R}^2$, and seeking the shortest path, which connects its two points A and B on opposite sides. We can now imagine the square to be a geodesic map of some mountains. We would like to find a path between A and B using the smallest energy possible. The general area of mathematics that concern itself with such problems is called the study of variations. The unfortunate fact is, that many of these problems do not posses a closed form solution. Therefore the idea is to approximate the curves with some numerical methods.

In this chapter we have seen that image segmentation problems can be solved with the continuous approach of minimum surfaces. The formulation of the minimum surfaces represents a close model of the Human Visual System and it can be mathematically formulated by simple means. In the next chapter we collect some specific notions about these minimization problems.

Chapter 2
Maximum Flows

The integral minimization with numerical methods can be attacked in two ways. These two ways are fundamentally different in their methods, precision, computation time and theoretical guarantees which they can provide. The first approach is to consider the image as a simple graph. The points are usually connected with the neighboring points. In this case the paths can be a set of edges or set of points meeting certain constraints. We generally refer to the graph based methods as the discrete approach. The advantages of this approach is, that we only have finite number of cases, so we generally do not have to worry about the existence of the solution. Naturally as the number of finite cases increases (possibly exponentially), many discrete methods quickly become unusable for images of bigger size.

The second approach is to consider the image as a continuous field, usually a rectangle or a prism, however higher dimensions are also appropriate. In this case our surfaces become topological planes and the function spaces uncountable (but separable in certain cases). As the explicit solution does not always exists, scientists are looking for iterative algorithms to approximate the solution of the problem. In the continuous case the problem of the algorithmic efficiency shifts from the classical big O efficiency to the speed of the convergence.

Also there is a newly raising issue concerning the speed of real implementations or certain algorithms. The big O efficiency measure of the algorithm usually does not anticipate the possible parallelization of the algorithm; much less the extent of parallelization. Local continuous iterative algorithms are often separable, so their parallelization is much easier than that of graph based algorithms. In this chapter we are going to walk through some examples of the curve integral minimization and demonstrate that the curve integral minimization is really a sub-problem of the family of *total variation minimization* problems.

2.1 Maximum Flows in the Discrete Domain

Historically image segmentation by optimization has evolved from the work of [13]. The original problem dates back to the second world war and was not motivated by image processing. It can be described as follow. Let us take a weighted directed finite graph $G(V, E, w)$, where w represents the edges' capacity. We separate this graph into two partitions A and $V \backslash A$ such a way, that A contains a special region S called the *source* and $V \backslash A$ contains another special region P called the *sink*. In the above setup, the edges which go from A to $V \backslash A$ could be interpreted as the "surface" of A. Here because the graph is directed, the direction $A \to V \backslash A$ is chosen. We denote this surface with ∂A. The smallest of these surfaces has a special interpretation. This was first noticed by Ford and Fulkerson and has later been known as the min-cut problem.

The extention of the problem for image processing can be described in the following fashion. Let us consider an image with an object of interest in front of some background. If the characteristics of the background are not rapidly changing, then the border between the background and the foreground can be seen as an unusual, rapid change in the local characteristics.

This change can be detected for example with a discrete gradient of the image. Low gradient values of the image will represent the interior of the background and the interior of the object. The high frequency parts; the high gradient values will represent the border between the object and its background. If we connect the points with high gradients, then we obtain a hyper-surface. If for some reason the border is not continuous, then the object will be limited by the hyper-surface which contains *most* of high frequency places (most of the border of the object). The area enclosed by the surface will segmentation. If the gradient of the image is properly calculated, then the Ford-Fulkerson method provides a way of finding the minimum cut, which is a curve traversing on the border of the object.

This general method has to be completed or improved for a given problem in several aspects. First of all, the gradient has to be calculated and filtered. Secondly, the source set has to be specified, which must be inside the object, but not necessarily on the border of the object. And finally as the method is discrete, it has some well known segmentation artefacts which have to be treated *a posteriori* or dealt witch during the segmentation process.

Nonetheless, the Ford-Fulkerson algorithm finds an optimal solution in some mathematical sense: Let us define a measure on G called the graph measure $♃$[1]. For a set $S \subseteq E$; $♃(S) = |S|$. Let w denote the weight of the edge. We define two dedicated sets $S \subseteq V$ and $T \subseteq V$ where $S \cap T = \emptyset$, referred as the *source* and *sink*. We can now define the original formulation of the min-cut problem:

[1] pronounce Jupiter

$$\min_{A \subseteq V} \int_{\partial A} w \mathrm{d}\mathcal{A} \qquad \text{s. t.} \qquad S \subseteq A \text{ and } T \subseteq V \backslash A \tag{FC}$$

We call (FC) the Ford-Fulkerson-cut. We do not need a special discussion on whether the minimum exists as our graph G is finite. This definition conforms the integral definition and the reader can easily verify that $\mathcal{A}$ is a measure.

For the minimization of (FC), [13] used a dual approach. They had defined a discrete flow F on G, which is represented on every edge by w_F with the properties from (FF). It essentially means, that we attribute each edge a flow w_F which is less then the *capacity* w_G of the edge in G. In each inner node of the graph the flow is *divergence* free.

$$\begin{aligned} &F(V,E) \cong G(V,E) \\ &\quad w_F(e) \leq w_G(e) && \forall e \in E \\ &\oint_{\partial v} w \mathrm{d}\mathcal{A} = 0 && \forall v \in \{V \backslash S \backslash T\} \end{aligned} \tag{2.1.1}$$

If F satisfies (2.1.1), than we call F a discrete flow. With this definition, the authors formulated the dual problem of (FC).

$$\max \oint_{S} w_F \mathrm{d}\mathcal{A} \qquad \text{s.t.} \qquad \text{F is a flow} \tag{FF}$$

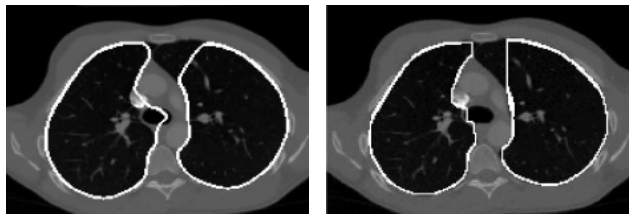
The authors presented a proof that (FF) is equivalent with (FC) in the sense, that the saturated edges of the discrete flow will form the minimum cut of the graph G.

The traditional approach of image segmentation with the discrete maxflow is to use a rectangular or parallelepipedic grid. The pixels of the image represent the nodes of the image and each node is connected with its four or six neighboring pixels[2]. Due to the discrete nature of the graph formalism the computation becomes both slower and more memory intensive as the image becomes larger. Also, as the surface normals can only be the edges of the graph, in a grid-graph the solution will likely be anisotropic[3]. On fig. 2.1(b). the vertical line has smaller integral, because it is only using one arrow path, whereas the desired border uses two. This artifact does not dissappear even if the resolution increases.

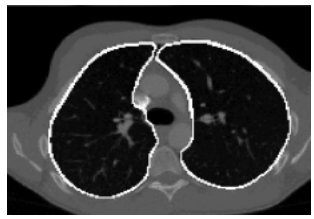
The problem can, however, be solved using carefully chosen special graphs. In [14] the authors have solved a more general problem on discretized fields. They have used a well known mathematical formulations from variational theory, the Euler-Lagrange functional. Let $L(x, u, \nabla u)$ be a function which depends explicitly on x some function $u(x)$ and its gradient $\nabla u(x)$. We call (EL)

[2] a pixel has four neighbors in 2D and six in 3D

[3] The horizontal and vertical planes will have lower integrals as their tilted counterparts.



(a) (b)



(c)

Fig. 2.1: The optimal curves on images. On image (a) there is a locally optimal curve. The optimum curves are presented on images (b) and (c). On image (b) the curve is optimized on the edges, while on image (c) the curve is optimal on the points.

the Euler-Lagrange functional.

$$E = \oint_0^1 L(x, u, \nabla u) \mathrm{d}x \tag{EL}$$

To avoid trivial solutions, usually u is subject to some limit conditions. Here we would like to remark that with (2.1.2), minimizing (EL) coincides with the (GAC) functional. In this way the optimum of (EL) will be the curve with the smallest integral, regardless on its position. The smoothness will not be constrained on the gradient of the curve, but it will still be enforced, through the length of the curve.

$$L_{\mathrm{GAC}}(x, u, \nabla u) := g \tag{2.1.2}$$

We want to minimize E with respect to u. Simply stated, we have an energy E which depends explicitly on x, u and ∇u. We want to find u for which (EL) is the minimum.

The authors of [14] then minimize (EL) on a discretized field. They start with the idea of creating a planar graph, which approximates (EL) continuously on the domain of F. In 3D they create a partition P_{kirs} of the field into polyhedra instead of a graph. After the graph or the partition has been

created, they can evaluate F on each segment (edge) or facet of each polyhedron of the partition, either analytically or numerically. Each edge or facet of each polyhedron can thus be attributed a weight. The authors proposed a formulation for choosing a dense partition, and thus any plane of reasonable regularity can be approximated with a subset of facets forming a topological plane.

From the partition the authors derive a dual graph P^*_{kirs}. Each partition from P_{kirs} represents a node in P^*_{kirs} two nodes are connected iff their corresponding partitions share a facet in P_{kirs}. The weight of each edge is the weight of the common facet. This weighted graph can then be used to find the optimum surface. In 2D, the surface is a path, in higher dimensions its a hyper-surface.

In the 2D case, as the optimal surface is a path, it can be found with the shortest path algorithm. In higher dimensions the surface or the hyper-surface can be calculated from the dual graph. The authors show, that the optimum surface is equivalent to the (FC) cut of P^*_{kirs}. The inconvenience of the method is, that its computational demands increase rapidly. For this the authors present a relaxed model in which the initial graph can be constructed in a more sophisticated way.

They experiment both with deterministic and random constructions. Their grids are generalizable to higher dimensions, but the number of nodes in they graphs increase heavily. Their function domain is generally $O(n^{2n+2})$, where n is the extent of discretization (not the image size). In the random graphs, they increase the density around anticipated solution thus decreasing the n required.

No algorithm to the author's knowledge has been proposed to find the optimum solution for the general at least 2D (EL) problem in the continuous domain. Some authors, however managed to find solutions for some subsets of the problem. Now we are going to collect some interesting propositions concerning some energy function minimizers in the continuous domain.

2.2 The Minimum Cut and Maximum Flow in the Continuous Domain

The max-flow–min-cut formulation can be interpreted in the continuous domain. One of the propositions can be found in [15]. The author, Strang, has used a continuous vector field F for representing the flow in the continuous domain Ω. He has drawn special attention to the border of Ω, but we omit the question of the borders in this thesis. In the formulation of Strang, F has been subject to the following constraints:

$$
\begin{aligned}
&|F| < g && \text{on } \Omega \\
&\operatorname{div} F = -\lambda S && \text{whereas } \begin{cases} S > 0 \text{ on the source} \\ S < 0 \text{ on the sink} \\ S \equiv 0 \text{ otherwise} \end{cases}
\end{aligned} \tag{2.2.1}
$$

From the above consideration Strang then defined the (MFS) problem.

$$\text{maximize } \lambda \qquad \text{s. t.} \qquad (2.2.1) \tag{MFS}$$

The dual formulation is also analogous to the discrete case, the $\mathbb{R}^2$ plane becomes a hyper-plane, and the minimum cut becomes the (GAC) formulation. Finally the author proves (2.2.2), namely that there is a similar equivalence between the max-flow and the min-cut than as in the discrete case.

$$\max \lambda = \inf_{\{S>0\} \subseteq C} \frac{\oint_{\partial C} g \mathrm{d}s}{\int_C S} \tag{2.2.2}$$

Nozawa in [16] goes even further by proving an equivalence, that is the min-cut is essentially a total variation minimization problem.

$$\inf_u \int_\Omega g|\nabla u| \tag{2.2.3}$$

(2.2.3) is minimized by a characteristic function $\mathbf{1}_C$ and ∂C is the optimal cut for (GAC).

2.3 The Algorithm for Finding the Optimum Flow

We have based our work on the algorithm described in [17]. In the continuous case the set of points is replaced by a continuous scalar field P and the set of edges is replaced by a continuous vector field $\overline{\mathbf{F}}$. $\overline{\mathbf{F}}$ is a flow if it satisfies two conditions:

- Conservation of flow: $\nabla \cdot \overline{\mathbf{F}} = 0$
- Capacity constraint: $|\overline{\mathbf{F}}| \leq g$

The conditions are analogous to the discrete case. Every closed surface around S limits the minimal surface. The continuous maximum flow system is described by the following equations:

$$\frac{\partial P}{\partial \tau} = -\nabla \cdot \overline{\mathbf{F}} \tag{2.3.1}$$

$$\frac{\partial \overline{\mathbf{F}}}{\partial \tau} = -\nabla P \tag{2.3.2}$$

$$|\overline{\mathbf{F}}| \leq g \tag{2.3.3}$$

Here P is analogous to a *pressure field* and $\overline{\mathbf{F}}$ a vector field of the *flow*. P is forced to 1 on the source and 0 on the sink. The solution of the equation will be discussed in chapter 3. For the purposes of section 2.5. we can reformulate the algorithm as:

$$\begin{aligned} P^{n+1} &= P^n - \partial\tau \nabla \cdot \overline{\mathbf{F}}^n \\ \tilde{F}^{n+1} &= \overline{\mathbf{F}}^n - \partial\tau \nabla P \\ \overline{\mathbf{F}}^{n+1} &= \tilde{F}^{n+1} \min\left\{1, \frac{g}{|\tilde{F}^{n+1}|}\right\} \end{aligned}$$

Now let us suppose that it converges for a given g. If the system is stable the following statements apply:

$$\nabla \cdot \overline{\mathbf{F}} = 0 \tag{2.3.4}$$

$$\nabla P = 0 \quad \text{if} \quad |\overline{\mathbf{F}}| < g \tag{2.3.5}$$

$$\nabla P = -\lambda \overline{\mathbf{F}} \quad \text{if} \quad |\overline{\mathbf{F}}| = g \tag{2.3.6}$$

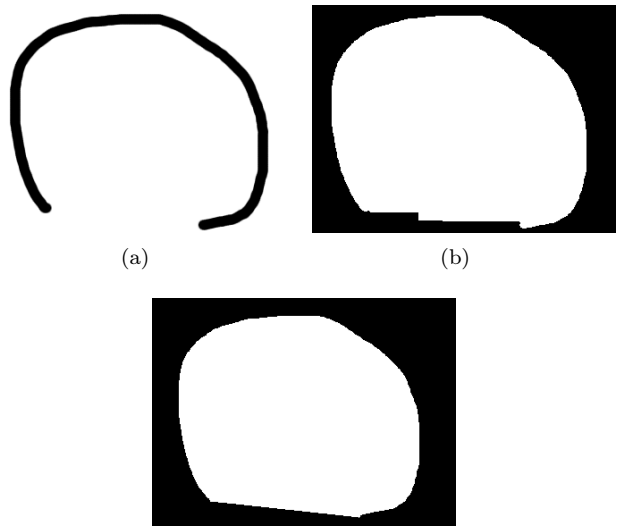
(2.3.4) simply restates the conservation of the flow. (2.3.5) applies if the flow have stabilized during the evolution without the (2.3.3) constraint. If without the constraint the flow would grow higher, still because the system is stable, its direction and magnitude remain constant. From (2.3.2,2.3.3) we can deduce that $\nabla P \cdot \overline{\mathbf{F}} \leq 0$, which means that P is a non strictly monotone decreasing function along the flow lines. If $\overline{\mathbf{F}}$ is dense, as it is divergence-free these flow lines can only initiate in the source and end in the sink. Now we define set $A = \{x | P(x) > p\}$ with $0 < p < 1$. On the iso-surface $Y := \partial A$ the $\nabla P \neq 0$ by construction, which means, that in these points (2.3.3) applies thus:

$$\int_A \nabla \cdot \overline{F_Y} = \oint_Y \overline{N_Y} \cdot \overline{\mathbf{F}} \mathrm{d}Y = \oint_Y g \mathrm{d}Y = \oint_Y \mathrm{d}G$$

This implies, that every iso-surface is a minimum. If there is only one minimum this also means, that the pressure field can be $0 \leq P \leq 1$ on a zero measure set. Before generalizing the integral formulation we would like to introduce another problem of image processing; the image denoising.

2.4 Discrete vs. Continuous Approach

If we consider the optimization approach to image segmentation we can find several key differences. In this section we are going to compare the continuous maximum flow algorithm with the Boykov-Kolmogorov version of the graph-cut algorithm. The first main difference is the quality of the segmentation. In the standard case the graph corresponding to the image is created the following way: one considers the pixels of the image as nodes of the graph and connects each node with its four or six neighbors in 2D and 3D respectively. This approach has a well known side effect of preferring the vertical and horizontal lines to interpolate the object at places, where the gradient is weak. An example can be observed in fig. 2.2.



(a) (b)

(c)

Fig. 2.2: An example of continuous maximum flow segmentation and graph-cut segmentation with standard discretization. The input measure 2.2(a) is set to zero on the black area and set to 1 on the white area. The graph-cut 2.2(b) shows segmentation artefacts when compared to the continuous maximum flows 2.2(c).

The above mentioned artefact can be avoided, as discussed in section 2.1, but the solution is not always applicable for an other inconvenience of the discrete approach: memory consumption. With the continuous maximum flow algorithm, we can segment a $[500 \times 500 \times 500]$ pixel size 3D image using a GPGPU with 3GB of memory. For the same image the Boykov-Kolmogorov

algorithm would consume 7 GB of memory (measured) using standard discretization. This fact in itself would rule out many embedded applications. Furthermore, if we would refine the graph further to avoid segmentation artefacts we would easily run out of the memory of any contemporary SMP machine.

As for the computation time, the Boykov-Kolmogorov algorithm performs well, however it is much less flexible when considering sub-optimal solutions. On fig. 2.3, which is a 3D image, the surface of the cumulated surface of the fibers is bigger than of a single cylinder encompassing all of the fibers. This means, the graph-cut will always return the cylinder if we specify all the sources at once. With continuous maximum flows, we can stop the segmentation before it reaches the steady state and we can get the fibers by thresholding the potential field. The problem can be solved optimally with both algorithms by segmenting the fibers one-by-one, but this would mean 620 segmentations, that is one for each fiber, in which case the segmentation time of the graph-cuts would be incomparably higher than that of the suboptimal segmentation with the maxflow algorithm.

Fig. 2.3: An example of the graph-cut segmentation artefacts in 3D. The yellow continuous maximum flow segmentation is superimposed by the red graph-cut segmentation. The red "bumps" are the segmentation artefacts of the discretization.

2.5 Total Variation in Convex Analysis and the Maxflow Algorithm

In the image denoising problem, we suppose that the image I is created by the real image and some noise as

$$I = I_{\text{real}} + Z$$

The problem is to determine the noise and reconstructing the image I_{real}. The key observation is that the noise can be modeled as the continuation of some kind of discrete i.i.d. probability variable z. This variable will have high total variation, that is

$$\int_\Omega |\nabla Z| \gg 1$$

The natural idea then follows, namely to find an image u which is close to the image, but has low total variation. There are two similar approaches to the problem. One formulation has been proposed in [18]. In this paper the authors were looking for a partition of the image into n regions, separated by piecewise smooth boundaries. Than they propose to minimize the Mumford-Shah energy functional on these regions.

$$E_{\mathrm{MS}}(u) = \lambda \int_\Omega (I-u)^2 \mathrm{d}x + \mu \int_{\Omega\setminus\Gamma} |\nabla u|^2 + |\Gamma| \qquad \text{(Mumford-Shah)}$$

The first term in E_{MS} is the fidelity term. It ensures that the optimal image will be close to the original in the integral metric, the second term is a regularity term, which assures, that the image will have low total variation in the interior of the partitions. The third term is also a regularity term. It assures that the total length of regions will remain small, that is we should not consider every pixel a separate partition. The disadvantage of the formulation is that the minimizers have to consider the curve integral and the total variation together.

For us, the interest in image denoising lies in its connection to the integral minimization. We can reformulate (GAC) using a Sobolev function u with sufficient regularity and codomain $[0,1]$ as

$$\min_{u|_S \equiv 1} \int_\Omega g\,|\nabla u| \tag{2.5.1}$$

that is the weighted total variation of u with u constrained to 1 on some set S. We are going to demonstrate the equivalence in $\mathbb{R}$. In (2.5.1) we consider the differential in the sense of distributions.

Definition 2.1. We consider the set of functions $\varphi \in \mathcal{C}_0^\infty(\Omega)$, i.e. the functions infinitely differentiable with compact support $\operatorname{supp}\varphi \subseteq \Omega$. For every $f \in L^1(\Omega)$ we define the linear operator (2.5.2) to be the *distribution* associated with f. We note the set of distributions with $\mathcal{D}$.

$$T_f := \langle f, \varphi \rangle = \int_\Omega f\varphi \tag{2.5.2}$$

Two distributions T_f, $T_g \in \mathcal{D}$ are equivalent iff the following holds:

$$\langle f, \varphi \rangle = \langle g, \varphi \rangle \qquad \forall \varphi \in \mathcal{C}_0^\infty(\Omega)$$

We consider that $f \cong g$ in the sense of distributions, if they behave similarly on the φ test functions. The interest of distributions lies in the differential operator and later in the integral generalization.

Definition 2.2. Let T_f be a distribution for some function $f \in L^1(\Omega)$. We consider (2.5.3) to be the differential of f in the sense of distributions.

$$\nabla T_f := -\langle f, \nabla\varphi \rangle \tag{2.5.3}$$

Now we consider the relation between ∇T_f and $T_{\nabla f}$. Substituting the definition we integrate *per partes*:

$$T_{\nabla f} = \langle \nabla f, \varphi \rangle = \int_\Omega \nabla f \varphi \overset{\text{p.p.}}{=} [f\varphi]_{\partial\Omega} - \int_\Omega f \nabla\varphi = -\langle f, \nabla\varphi \rangle$$

After the integration per partes, we consider $[f\varphi]_{\partial\Omega}$. In the definition we have restricted φ to have a compact support within Ω, therefore the product is zero everywhere on the border. From this demonstration we can see, that if f is differentiable in the classical sense, then $\nabla T_f \cong T_{\nabla f}$. If f is not differentiable in some point $x_0 \in \Omega$ we can still attribute a differential distribution to f. This means that the differential in the sense of distributions is an extension of the differential operator to L^1, that is every L^1 function is *differentiable* in the sense of distributions. If there is no confusion we note T_f and ∇T_f simply f and ∇f respectively. The integral can also be interpreted on $\mathcal{D}$. We note that $\mathcal{C}_0^\infty(\Omega)$ is dense in $L^1(\Omega)$. If we consider $f \in \mathcal{D}(\Omega)$ and $g \in L^1(\Omega)$. Then for the integral $\int_\Omega fg$ we can apply

$$\int_\Omega fg = \lim_{\varphi \to g} \langle f, \varphi \rangle$$

We note that the unicity comes from the continuity of the integral operator.

Here we would like to remark, that the product fg of two distributions f and g is not necessary a distribution. Therefore the differential equations cannot generally be solved using the distributions alone.

Next we are going to discuss the relation of (GAC) and (2.5.1) in $\mathbb{R}$. The 1D (GAC) formulation can be written as (2.5.4), where $a, b \in \mathbb{R}$, Ω and S are a connected intervals of $\mathbb{R}$. We can assume that $\Omega = [0, 1]$ without loss of generality.

$$\min_{S \subseteq [a,b] \subseteq [0,1]} g(a) + g(b) \tag{2.5.4}$$

In simple terms, we are looking for an segment $[a, b]$ which encloses S, and has the smallest possible weight. Now let $[a_0, b_0]$ be the optimum interval. We define u as:

$$u(x) := \begin{cases} 1 & \text{if } x \in [a_0, b_0] \\ 0 & \text{otherwise} \end{cases}$$

We are looking for the integral $\int_0^1 g|\nabla u|$. For this we consider first a simpler integral; that of the Heaviside function H.

$$H(x) := \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \qquad \text{(Heaviside)}$$

From the definition we can calculate the differential of H:

$$\nabla T_H = -\langle H, \nabla\varphi\rangle = -\int_0^\infty \nabla\varphi = [\varphi]_0^\infty = \varphi(0)$$

Now substituting $u = H(a_0 - x) - H(b_0 - x)$ we have equality

$$\begin{aligned} \langle|\nabla u|, \varphi\rangle &= \int_0^1 \varphi|\nabla u| \\ &= \int_0^1 \varphi|\nabla\left(H(a_0 - x) - H(b_0 - x)\right)| \\ &= \int_0^1 \varphi|\nabla H|(a_0 - x) + \int_0^1 \varphi|\nabla H|(b_0 - x) \\ &= \varphi(a_0) + \varphi(b_0) \end{aligned}$$

Note that in the final step, we have considered the differentials of the Heaviside function with the signs of their mollifiers. We can see, that if we substitute $\varphi_n \to g$ we obtain

$$\langle|\nabla f|, g\rangle = \lim_{n\to\infty} \varphi_n(a_0) + \varphi_n(b_0) = g(a_0) + g(b_0) \qquad (2.5.5)$$

With (2.5.5) we can see that in $\mathbb{R}$, for every g we can find an image with the same total variation as the weight of the minimum surface (2.5.4) on g. If we would like to show that this image has also the lowest total variation, we can use the Coarea formula.

$$\int_\Omega g|\nabla u| = \int_{\mathbb{R}} \oint_{u^{-1}(t)} g \mathrm{d}H_{n-1}\mathrm{d}t \qquad \text{(Coarea)}$$

In the Coarea formula g represents our measure, u is the image with the minimum variation, u^{-1} is the inverse of u, whereas H_{n-1} is the $n-1$ di-

mensional Hausdorff[4] measure [19]. A version of the Coarea formula has been proven for the continuous maximum flow problem in the space of the distributions $\mathcal{D}$, but here for the sake of simplicity we use a more restricted field. Let us consider $\psi \in \mathcal{C}_0^\infty$ with $\|\psi\|_{L^\infty} \leq 1$.

The *domains* in $\mathbb{R}$ degrade into intervals. As the *curves* are the border points of domains, the curves in $\mathbb{R}$ degrade into the endpoints of intervals. Let's therefore consider an image ψ. The weighted total variation of ψ can be written using the Coarea formula as

$$\int_\Omega g|\nabla\psi| = \int_0^1 \oint_{\psi^{-1}(t)} g \mathrm{d}H_0 \mathrm{d}t \tag{2.5.6}$$

For our case $H_0(p) = 1$ for all points of the interval. Now we are going to examine ψ^{-1}. $\psi^{-1}(t)$ represents the set of points $p_i(t)$, $i < N_t$, of ψ for which $\psi(p_i(t)) = t$. If $\int_\Omega g|\nabla\psi| \leq \infty$ then there are only zero measure values of t for which $N_t = \infty$. We can therefore continue (2.5.6) as:

$$\int_\Omega g|\nabla\psi| = \int_0^1 \sum_{i=0}^{N-1} g(p_i)\mathrm{d}t \geq \int_0^1 g(p_0(t)) + g(p_1(t))\mathrm{d}t \tag{2.5.7}$$

The inequality (2.5.7) holds because an interval has at least two extremal points. Also as ψ is continuous there will be at least one interval for every $t \in [0, 1]$. Therefore:

$$\left.\begin{array}{l} g(p_0(t)) \geq g(a_0(t)) \\ g(p_1(t)) \geq g(b_0(t)) \end{array}\right\} \Longrightarrow \int_\Omega g|\nabla u| \geq \int_0^1 g(a_0) + g(b_0)\mathrm{d}t = [t]_0^1 [g(a_0) + g(b_0)] \tag{2.5.8}$$

The last thing to consider is that if $a_0 \neq b_0$ then $p_0(t) \neq p_1(t)$ as the interval has to contain the source S.

Now we collect our observations. From (2.5.8) we know that on $\mathbb{R}$, the total variation is greater than the minimum surface for every admissible ψ and from (2.5.5) we know that there exist a minimizer which is the characteristic function of the interior of the minimum surface. From this we have shown or $\mathbb{R}$ that (GAC) and the weighted total variation (2.5.1) are equivalent.

In [16] a more general relation between (GAC) and (2.5.1) is proven, namely that the two relations are also equivalent in the space of distributions in arbitrary finite dimension.

[4] The Hausdorff measure is the generalization of the area in arbitrary dimensions. In trivial cases it coincides with the intuitive notion of area. The Hausdorff measure gives the surface in 3D, the curve length in 2D and the number of points in $\mathbb{R}$.

$$E_{ROF}(u,\lambda) = \int_{\Omega} |\nabla u| + \lambda \int_{\Omega} (u-f)^2 \tag{ROF}$$

Using the notion of distributions [20] proposed another energy function for image denoising. The authors in the paper have restricted u to be a Sobolev function and considered ∇u in the sense of distributions. Their formulation (ROF)[5] while similar, avoids referencing the total length of the regions. Naturally, if we choose a characteristic function χ_L for some set $L \subseteq \Omega$, the total variation $\int_{\Omega} \nabla L = \text{per}(L)$ will be the H_{n-1} perimeter of L. This property limits naturally the total length of the region- or regions in the image.

$$E_{\text{TV}}(g,\lambda) := \int_{\Omega} g\,|\nabla u| + \lambda \int_{\Omega} (Ku-f)^2 \tag{2.5.9}$$

We can connect (GAC) with (ROF) using weighted total variation as in (2.5.9). Here g represents the local cost measure. At higher values of g we want smoother image, generally in regions, whereas at low values of g we would like higher fidelity to the original image. We choose low values of g on the edges, where it permits us to have discontinuities between regions. λ represents the global fidelity. The higher the value of λ, the closer the solution will be to the original image, that is less noise will be removed minimizing (2.5.9). K is a linear operator which represents some image degradation, like blurring or tilted projections of the image. K can be, for example, the convolution of the image with the Gaussian kernel. An example is shown on fig. 2.4. Images with differend noise damage are reconstructed using the (ROF) energy function and the continuous maximum flows. We can see, that $E_{\text{TV}}(\mathbf{1}_{\Omega},\lambda) \cong$ (ROF) whereas $E_{\text{TV}}(g,0) \cong$ (GAC). With (2.5.9) we can see that the image segmentation with minimal surfaces and image denoising with total variation minimization are both a sub-problem of the total variation minimization. In the next section we are going to review the problem from the algorithmic perspective, which will lead us to the same conclusion. We also look at the numerical solution of the total variation minimization.

In addition to the similarity between the minimum surfaces and total variation denoising, there exists another set of algorithms, which raise interest in the continuous maximum flow method. We consider (2.5.10) from the framework of convex analysis.

$$E_{\text{conv}} := \min_{u\in\mathcal{H}} f_1(u) + f_2(u) \tag{2.5.10}$$

Here we assume that both f_1 and f_2 are convex problems. We consider u to be defined over some Hilbert space. We note that if we set $f_1 := \int_{\Omega} g\,|\nabla u|$ and $f_2 := \lambda \int_{\Omega} (u-f)^2$, we find E_{TV}.

[5] Rudin, Osher and Fatemi

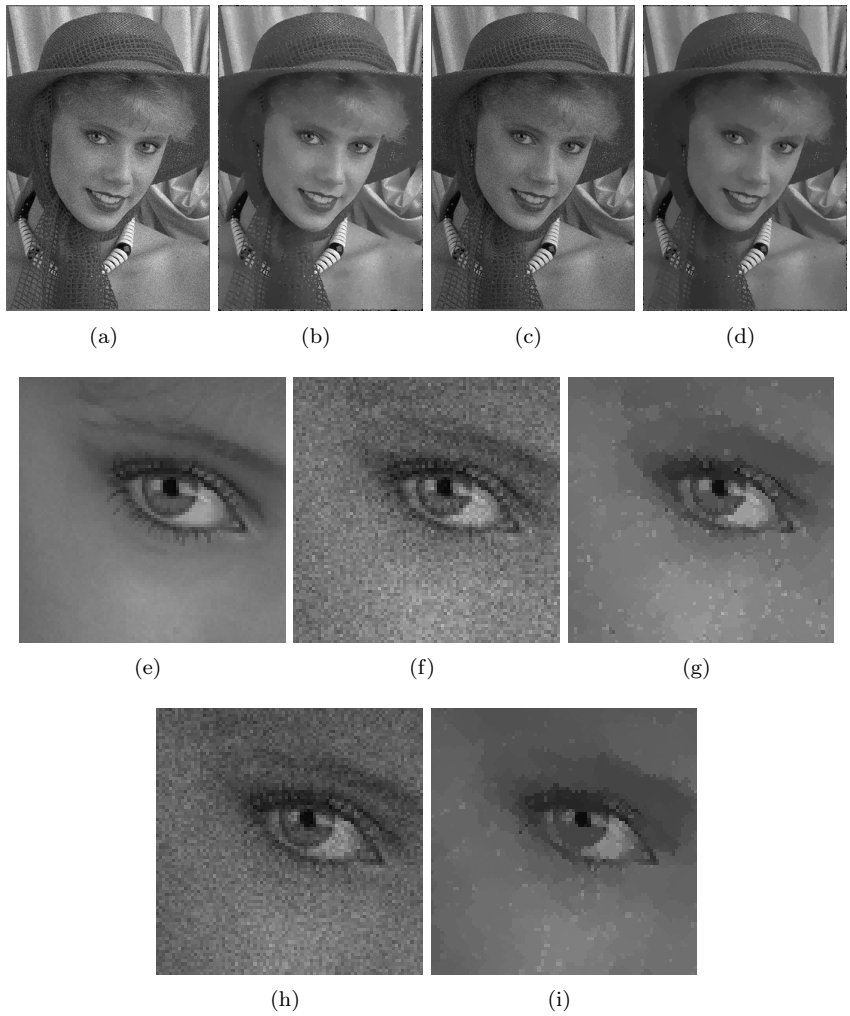


Fig. 2.4: Total variation denoising with continuous maximum flows. We minimize the (ROF) energy. Image 2.4(a) and 2.4(f) have been damaged by a gaussian noise. Their denoised versions are on subfigures 2.4(b) and 2.4(g). Images 2.4(c) and 2.4(i) have been damaged by a poisson noise. Their denoised versions are on subfigures 2.4(d) and 2.4(i). Subfigure 2.4(e) is a crop from the original images. It is shown for comparison.

As both f_1[6] and f_2 are convex, (2.5.10) is a convex optimization problem. Many authors have proposed algorithms to optimize E_{conv} [21, 22, 23, 24]. When both f_1 and f_2 are differentiable everywhere, the problem is largely classical and can be solved with variations of the gradient descent algorithm. If either f_1 or f_2 is not everywhere differentiable, some authors have proposed using the proximity operator to solve E_{conv}. In the optimizations, the authors use a proximity operator for the addends of E_{conv}. The proximity operator is the unique solution of:

$$\text{prox}_f(u) := \min_{u \in \mathcal{H}} f(y) - \frac{1}{2} \|u - y\|^2 \tag{2.5.11}$$

Proximity operators have convenient properties, which make them particularly well suited for iterative optimization. In particular, the operator has an explicit solution in many cases, and an algebra of prox operators exists. As several problems can be solved using convex analysis, an efficient algorithm and implementation of the proximity operators is considerably important.

An exact proximity iterator for the total variation was first proposed in [25]. Later the authors of [26] gave an explicit link between TV minimization and a variation of the maximum flow algorithm, which correspond to a projected gradient operator discussed along this thesis. In the remainder of this section we present the chain of thought from the two papers, that lets us use the continuous maximum flow algorithm for noise removal by minimizing the total variation.

As mentioned earlier, historically there have been two independent propositions for the maximum flow algorithm. From [27], we can deduce the following argumentation. We want to obtain the solution of the proximity operator for the total variation:

$$\min_u \int_\Omega g|\nabla u| + \frac{1}{2\theta} \int_\Omega (u - v)^2 \tag{2.5.12}$$

We observe that we can replace the total variation computation with the following convex maximization problem:

$$g|\nabla u| = \max_{\|F\| \leq g} \overline{\mathbf{F}} \cdot \nabla u \tag{2.5.13}$$

We can express $\overline{\mathbf{F}}$ explicitly as $\overline{\mathbf{F}} = g\frac{\nabla u}{|\nabla u|}$. After substituting (2.5.13) into (2.5.12) we obtain:

$$\min_u \max_{\|F\| \leq g} \int_\Omega \overline{\mathbf{F}} \cdot \nabla u + \frac{1}{2\theta} \int_\Omega (u - v)^2$$

[6] The convexity of f_1 has been proven in [16]

As the minimization is convex, we can exchange the min-max operators, and with $\int_\Omega \overline{\mathbf{F}} \cdot \nabla u = -\int_\Omega u \text{ div } \overline{\mathbf{F}}$, we can eliminate the gradient:

$$\max_{\|F\| \leq g} \min_u -\int_\Omega u \text{ div } \overline{\mathbf{F}} + \frac{1}{2\theta} \int_\Omega (u-v)^2 \tag{2.5.14}$$

With the corresponding Euler-Lagrange equation:

$$-\text{ div } \overline{\mathbf{F}} + \frac{1}{\theta}(u-v) = 0 \tag{2.5.15}$$

$$u = \theta \text{ div } \overline{\mathbf{F}} + v \tag{2.5.16}$$

Re-substituting (2.5.16) into (2.5.14) we can eliminate u from the equation:

$$\begin{aligned}
&\max_{\|\overline{\mathbf{F}}\| \leq g} \int_\Omega -\left(\theta \text{ div } \overline{\mathbf{F}} + v\right) \text{ div } \overline{\mathbf{F}} + \frac{1}{2\theta} \int_\Omega \left(\theta \text{ div } \overline{\mathbf{F}} + v - v\right)^2 \\
&= \max_{\|\overline{\mathbf{F}}\| \leq g} \int_\Omega -\left(\theta \text{ div }^2 \overline{\mathbf{F}} + v \text{ div } \overline{\mathbf{F}}\right) + \frac{1}{2\theta} \int_\Omega \theta^2 \text{ div }^2 \overline{\mathbf{F}} \\
&= \max_{\|\overline{\mathbf{F}}\| \leq g} -\int_\Omega v \text{ div } \overline{\mathbf{F}} - \frac{\theta}{2} \int_\Omega \text{ div }^2 \overline{\mathbf{F}} \\
&= \min_{\|\overline{\mathbf{F}}\| \leq g} \int_\Omega v \text{ div } \overline{\mathbf{F}} + \frac{\theta}{2} \int_\Omega \text{ div }^2 \overline{\mathbf{F}}
\end{aligned}$$

The Euler-Lagrange of which is:

$$v + \theta \text{ div } \overline{\mathbf{F}} = 0$$

From the above consideration and (2.5.16), we can deduce the following projected gradient descend algorithm:

$$\tilde{F}^{n+1} = \overline{\mathbf{F}}^n + \frac{\tau}{\theta} \nabla u^n \tag{2.5.17}$$

$$\overline{\mathbf{F}}^{n+1} = \tilde{F}^{n+1} \min \left\{ 1, \frac{g}{\left\| \tilde{F}^{n+1} \right\|} \right\} \tag{2.5.18}$$

$$u^{n+1} = v + \theta \text{ div } \overline{\mathbf{F}}^n \tag{2.5.19}$$

This proximity operator has been independently discovered by [27] and rediscovered by [10]. This provides a solution for the proximity operator of the total variation. Furthermore if we substitute $v := u^n$, then the system changes to:

$$E_n = \int_\Omega g|\nabla u_n| + \frac{1}{2\theta}\int_\Omega (u_n - u_{n-1})^2$$

Experimental results show, that the above modification is still stable and $E_n \to \min_u \int_\Omega g|\nabla u|$. This is the modification, that we use for segmentation in chapter 4. From the above argumentation we can see, that an efficient implementation of the maximum flow algorithm is important both from the segmentation and the total variation minimization point of view.

2.6 Summary

In this chapter we have discussed the mathematical approaches to surface minimization. From the two main approaches, the graph-based and continuous, we have discussed the most historically relevant and the most recent propositions. From the graph based approach we have reviewed the Ford-Fulkerson algorithm and its recent improvement by [14] We have seen that one of its major drawbacks for image processing is due to its preferred directions. The Ford-Fulkerson algorithm prefers the direction along the axis which often introduces segmentation artifacts. We have seen that the discrete approach also lacks the parallelization property, as well as it is very demanding on the computer memory as the desired precision of the results increase.

We have seen that the continuous approach is also actively researched area of image segmentation. From the historical formulations of Strang, we went through the algorithm of Appleton and Talbot, and we have also seen that the formulation does not end there. We went through a demonstration and we have seen that the proposed algorithm is also applicable to total variation minimization. We could then use the same algorithm for image denoising and image sharpening. The reviewed algorithm is separable and it has mild memory demand.

In the next part we look into the implementation of the projected gradient algorithm as well as its parallelization.

Part II
Continuous Maximum Flows: Implementation and Applications

There are only two kinds of programming languages: those that everybody's bitchin' about and those nobody's usin'.

Bjarne Stroustrup

Chapter 3
Implementing Continuous Maximum Flows

The general proposition of algorithms, tend to address the problems, that we encounter in real-world architectures. Until approximately 2004 the main improvement in computation power had been through the increase of processor frequency [28]. This was a convenient approach from the theoretical point of view, as the improvement of the speed of the implementations was roughly proportional to the increase of the processor frequency. Since 2004 the main source of theoretical performance improvement is in the number of processing units, cores. A modern x86-64 architecture can have as many as 16, while GPGPU architectures can have several hundreds of identical (but simpler) cores on a single die as of 2011. These are examples of homogeneous architectures. Apart from this path of architecture development, the computing industry has also begun to experiment with massively SIMD[1] and heterogenous architectures.

As for any algorithm, an efficient implementation is inevitable. The algorithm presented in [10] is a fine-grain parallel algorithm. We present here implementations on three different architectures. Particular attention will be given parallelization. We carry out estimations and the measurements for 2D and 3D images. The algorithm (and the implementation) supports images of arbitrary dimensions, however the majority of the problems (as of august 2011) still remain at most 3D.

To understand the main challenge let us imagine the following maxflow-like but simpler algorithm:

$$\begin{aligned} a_i &= b_i + b_{i+1} + b_{i+2} + \cdots + b_{i+k} \\ b_i &= a_i + a_{i+1} + a_{i+2} + \cdots + a_{i+k} \end{aligned} \tag{3.0.1}$$

(3.0.1) is an iterative update scheme. Let us say we want to parallelize this algorithm. If we would simply partition the indices between the threads A and B, than very likely one of the threads (A) would finish updating $\overline{\mathbf{a}}$ faster

[1] Single Instruction Multiple Data

than the other. Then A would proceed to update $\overline{\mathbf{b}}$ from the values of $\overline{\mathbf{a}}$ which would not have been updated by B yet. The only way to avoid the interference is if the two threads synchronize, that is to say the threads wait for each other, both after updating $\overline{\mathbf{a}}$ and $\overline{\mathbf{b}}$. (3.0.1) is therefore a fine-grain parallel update scheme.

The most common computer architecture today is x86. Historically x86 is a sequential architecture. The commands are input into a *pipeline* and are expected to be executed in-order. More precisely, the result of the computation is supposed to be that of the in-order execution. In practical implementations the processor executes the commands out-of-order but only if it can guarantee that the result will be unaltered by the permutation. During the past two decades the x86 architecture has accumulated lots of instructions. Some of the instructions are composed of several simple operations. This can be an addition followed by a multiplication or a jump based on a condition. The natural way to implement these instructions is by reusing the existing operations. The simple operations are collected into *pipelines*, where an instruction traverses through units, in which the appropriate operations are carried out. Using this approach the simple operations can be carried out independently. For example we want to carry out an addition followed by a multiplication. After the first part of the instruction, the addition module is ready to compute for a different instruction. The accumulating number of processor instructions has made the computation pipes longer. Several reimplementations of the architecture have tried to reduce the length of the pipelines, but they are still longer than in the early years of x86. This has an inconvenient side effect for the fine-grained parallel computation. The pipe works as follows. We insert the commands into the pipe one-by-one. On the end of the pipe we can collect the results of the computation one-by-one. There are several instructions pending in the pipe in normal operations. Therefore if we want to synchronize the threads on different cores we need to stop inserting data into the pipe. In (3.0.1) we could not continue as the input may not be ready. As we stop feeding the pipe, as the pipe finishes the ongoing computations, it becomes empty. For example, if the pipe processes 10 instructions at the time, emptying the pipe takes 10 instructions. Therefore the synchronization on x86 is expensive. Generally x86 will be difficult to implement fine-grained parallel algorithms on x86 and the details will be discussed in chapter 3.3.

The GPGPU architectures are becoming more and more significant. Historically, the GPGPU architectures have begun as graphical processors in graphics cards. In a very simplified settings, the 3D scene is a set of triangles and light sources. In the raster graphic rendering the GPU has to calculate the color of the triangle based on the illumination. The triangles are then projected to the computer screen thus creating the *image*. The color of the triangle can be calculated using a simple and reduced instruction set[2]. As the GPU core is much smaller and simpler than an x86 core, the natural

[2] compared to the x86

way of acceleration was to put more of them on a single die. Today a GPU can contain hundreds of identical cores. Another advantage of the reduced instruction set comes from the simplicity of the circuit diagram. As the pipes are much shorter the GPU cores can be synchronized more strictly, that is to say finish every instruction in a single clock signal[3]. Therefore the GPU where naturally suited for fine-graining. With the increasing demand for graphical performance, the GPU cores have become more-and-more sophisticated up to the point where they have become suited for General Purpose computation. As an advantage of the reduced instruction set if an algorithm is implementable on a GPGPU unit it can have much higher performance. The details of the GPGPU implementation can be found in chapter 3.4.

An inconvenience of the GPU architectures is, that it lacks many of the important instructions for operating system execution. Context switches, protected instructions are not present in the GPU. Furthermore, while having a huge cumulated processing power, each single core is slow compared to even the older i386[4] processors. This means, that every program would need to be reimplemented for the GPU in order to use it as the main architecture in computers. The capability to run legacy software is one of the vast major strengths of the x86 architecture and the main reason of its ongoing popularity[5].

The natural extention of this idea is to design microprocessors, which contain both x86-like and GPU-like processing cores. This design is referred to as heterogenous processor architecture. The first mass produced heterogenous architecture was the Cell Broadband Engine Architecture (CBEA). This architecture encompasses Two PowerPC cores together with eight Synergic Processing Elements (SPE). The PowerPC cores run the operating system, while the SPEs accelerate numerical computations. At the time of the writing of this thesis the CBEA is being phased out, but it is not without interest for two reasons. Firstly because there are other upcoming heterogenous architectures[6] and secondly because the cross-platform programming environment is still an open question. As GPGPU does not support the full C or C++ languages, the new cross platform development language is of high scientific interest. The details of the implementation on this architecture can be found in chapter 3.4.3.

Another approach of parallel computation is to use a explicit instruction level parallelism. It is usually implemented in a massively SIMD instruction set. In a SIMD instruction the same mathematical or logical operation is

[3] In the x86 architectures the modules are not synchronized, so an instruction can take for example 2.3 clock signals

[4] The i386 is the smallest subset of the x86 instruction set that is capable of running a multitasking operating system.

[5] Compared to different but fully featured architectures like SPARC, PowerPC, MIPS, Itanium, etc.

[6] The AMD fusion is a current candidate for heterogenous processing.

performed on multiple data. In the classical implementation[7] the vectors are four float long.

$$\overline{\mathbf{a}} + \overline{\mathbf{b}} \Leftrightarrow \begin{cases} a_0 + b_0 \\ a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \end{cases} \tag{3.0.2}$$

In a more recent, massively parallel SIMD architecture[8] the vectors can be eight or sixteen float long. In each instruction the instruction is executed on all elements of the vector. Naturally for the algorithm to benefit from the SIMD instruction set each step has to be divisible in eight or sixteen independent operations. Unfortunately we did not have a massively parallel SIMD processor in disposition, but we have used the SIMD instruction set of the x86 processors to accelerate the computation. This implementation could be simply adapted on Itanium processors.

In this part we are going to detail efficient implementation of the maxflow algorithm as well as the real-word performance of the implementations. In the estimations and the subsequent measurements we will use two types of images. A 2D image [640×400] and a 3D image [500×500×500] will be used for benchmarking. All the source code used in this part is freely available under the CeCILL license. All but the source code for the CBEA is published as a part of the Pink Image Processing Library [29]. A short overview is presented in appendix 5.

For an efficient performance boost, the implementation of any algorithm has to be capable of dividing the computation to several isolated tasks. The extent of isolation will be discussed in section 3.3.1.3. Also while most of today's algorithms still treat the memory as a single linear block of values, the increasing size of the memory yields to higher latencies and lower proportional memory bandwidth. The memory architectures of modern middle class SMP computers will be discussed in section 3.3.1.1. For the benchmarking the following architectures will be used:

pc4356c Is a middle class workstation HP xw9400. This workstation features two Quad-Core AMD Opteron(tm) 2360 SE processors running at 2.5GHz. This system possesses two NUMA nodes.

blade13 Is an IBM HX5 blade server featuring four Intel(R) Xeon(R) L7555 octo-core processors clocked at 1.87GHz. This system possesses four NUMA nodes.

tesla Is an NVidia Tesla C2050 GPGPU card. It features 448 CUDA cores clocked at 3MHz and 3GB memory.

blade10 Is an IBM blade server featuring 2 Cell Broadband engines clocked at 3.2GHz.

[7] AMD's 3DNow! from 1998 or Intel's SSE from 1999

[8] like Intel Itanium architecture

With these systems we will be able to compare the maxflow algorithm proportional to the hardware cost as well as some of the drawbacks of modern x86-64 systems in implementing fine-grained parallel algorithms. The next chapter will deal with the theoretical considerations.

3.1 Theoretical Considerations

Before the efficiency of an implementation can be judged it is necessary to make some theoretical estimates of the computational power and memory bandwidth needed. For these estimations, we will assume the following setup: The simulation will be carried out on 4 byte floating point[9] arrays. We will test two images of size $[640 \times 400]$ and $[500 \times 500 \times 500]$. The 3D image used for the benchmarking can be seen on fig. 3.1. The number of iterations will be set to 3000[10] and 30000[11]. For the theoretical needs of the algorithm, we consider the memory bandwidth as the major barrier. We will demonstrate, the implementation scales proportionally to the memory bandwidth.

As for the memory bandwidth, we need to consider every access to a pixel of the image. A memory access can be cacheable or un-cacheable, depending on the size of the array on which the algorithm has to operate. In practical streaming algorithms, the memory bandwidth will be much lower than the computation power of the processor.

3.1.1 The Dynamics of the Iterations

In the total performance of the maxflow segmentation it is also important to consider the how the iteration time is distributed among individual iterations. The naïve idea would be to think of a uniform distribution, that is to say every iteration would take $t_0(I)$ time. This is not the case. In fact the iteration time depend on the image I, the sources and sinks S. Somewhat less intuitively the iteration time also depends on the field of the flow F. If we initialize $F \equiv 0$, then F will be a function of the number of the iteration. The final model can be represented as (3.1.1).

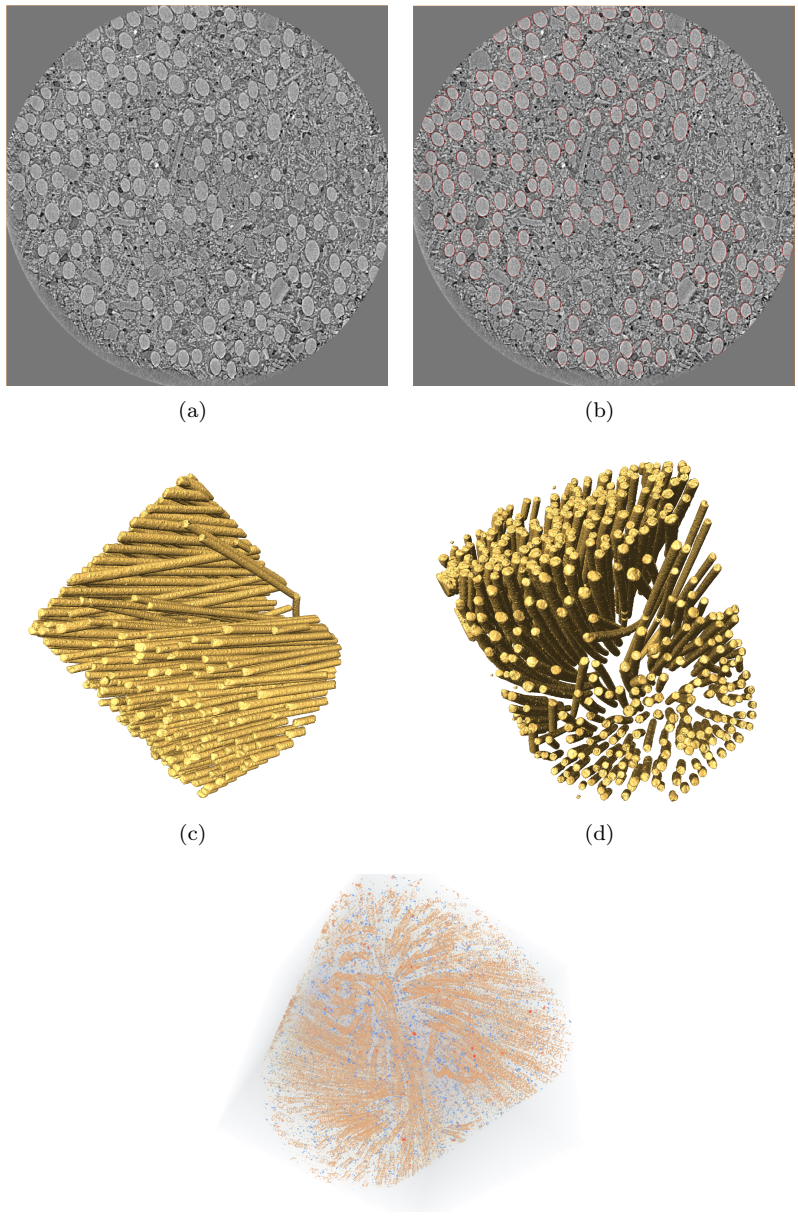
$$t_n(I, S, F) \Leftrightarrow t_n(I, S, F(n)). \tag{3.1.1}$$

We know from the profiling of the maxflow algorithm, that by far most of the computation time is used while updating the constraint. We know that the constraint is updated when the total flow exceeds the given constraint.

[9] IEEE Standard for Floating-Point Arithmetic (IEEE 754)

[10] 3000 iterations are typically sufficient for practical segmentations

[11] 30 000 iterations are typically sufficient for convergence



(a) (b) (c) (d) (e)

Fig. 3.1: The image used for benchmarking the 3D segmentation. On image (a) we can see a 2D slice of the image. Image (b) shows the segmentation superimposed on the 2D slice. Images (c) and (d) show the result of the segmentation of the fibers. On image (e) we can see a 3D volume rendering of the original 3D image. We can see, that it is very difficult to recognize the objects without the segmentation.

We also know from theory that the flow will equal the g constraint on the optimal hyper-surface. In the beginning when the flow is zero no constraint is enforced, so the iterations are naturally faster. Later on, when the hyper-surface begins to formulate, the constraint has to be enforced on higher and higher areas of the image, so the iteration time t_n increases. The slowdown can be measured, but whether the iteration time sequence t_n is monotone or quasi monotone in n is yet to be determined.

3.1.2 Memory Bandwidth

We are going to follow the maxflow algorithm detailed in section 2.4. For the calculation we need an array for the P, d arrays for F and an array for the g constraint. In a practical implementation, the limit conditions (the source and the sink) have to be enforced as well. For the efficiency of comparisons, the indicators of the source and the sink are also kept in a floating point array. These arrays represent for a 2D image $5 \times N$ floats, and for a 3D image $6 \times N$ floats where N is the number of voxels (3D). This is the pure memory consumption. If an iterative streaming algorithm consists of N iterations, each pixel of the image has to be accessed proportionally N times. If we know the memory bandwidth of the system, the scale-up can be constrained as

$$t_p \leq t_1 \frac{m_1}{m_p} \tag{3.1.2}$$

where t_p is the time needed for the algorithm using p threads, t_1 is the time on a single thread, m_p is the memory bandwidth achievable on p threads and m_1 is the memory bandwidth achievable on a single thread. As each pixel has to be processed in each part of the iteration, the data needed by an iterations is:

- **Updating the potential**
  The potential is updated from the flow arrays, however the limit conditions are also enforced in this step. For this we need to access each value of the potential, the flow and the srcsink array. This means, that for updating the potential, we need to access $4 \times N$ floats for a 2D and $5 \times N$ for a 3D image.
- **Updating the flow**
  The flow is updated from the potential. This means, that $3 \times N$ floats for a 2D image and $4 \times N$ for a 3D image, as the limit conditions need not be updated.
- **Updating the constraint**

2D image

iterations	**1**	**3000**	**30000**
flow	2.93 MiB	11.44 GiB	114.44 GiB
pot	3.9 MiB	8.58 GiB	85.83 GiB
cons	2.93 MiB	11.44 GiB	114.44 GiB
total	**10.74 MiB**	**31.47 GiB**	**314.71 GiB**

3D image

iterations	**1**	**100**	**1000**
flow	2.5 GiB	250 GiB	2.44 TiB
pot	2 GiB	200 GiB	1.95 TiB
cons	2.5 GiB	250 GiB	2.44 TiB
total	**7 GiB**	**700 GiB**	**6.83 TiB**

Fig. 3.2: Estimated memory bandwidth consumption of maxflow for streaming

The constraint is updated on every arrow of the flow. The reference constraint g is used to decide weather the flow exceeds the limit, so we need to access $3 \times N$ floats for a 2D image and $4 \times N$ floats for a 3D image. In summary, we need $(4+3+4) \times N = 11 \times N$ floats for a 2D image and $(5+4+5) \times N = 14 \times N$ floats for a 3D image. For our reference images these are collected in table 3.2.

3.1.3 Synchronization

The maxflow algorithm consists of three major steps: 1. updating the potential 2. updating the flow 3. enforcing the constraint on the flow. In each three steps one has to use the neighbors of the point to update its value. This means, that the flow cannot be updated before all the potential values are put in place. This results in three synchronizations, barriers, for each iteration. If a barrier is called, the thread must suspend its calculation until all threads have finished with their calculation. In practice this has several ramifications, which will be discussed in section 3.3.1.3.

3.2 The Reference Implementation

The reference implementation will be the one from [30]. In that report we have carried out a detailed analysis of the prefetching, and have discussed out first intent to eliminate the branches from the implementation. In the reference implementation, as well as in all the subsequent implementations, we have added some constraint on the sink to eliminate special points in the image.

We require in the implementation that all the border points belong to the sink[12] and border points are constrained to 0 in g. With this constraint all the border points can be treated as internal points. From the point of view of the memory representation, the image can be interpreted as a d dimensional matrix. It is stocked as a linear, one dimensional array. Let the size of the image I be $\overline{\mathbf{d}} := [d_0, d_1, \ldots, d_{d-1}]$. I is a linear array of size $[0 \cdots \prod d_i]$. Every point is represented in the array as

$$p[p_0, p_1, \ldots, p_{d-1}] = I\left[p_0 \cdot \frac{\prod d_i}{d_0} + p_1 \cdot \frac{\prod d_i}{d_0 \cdot d_1} + \cdots + p_{d-1}\right] \tag{3.2.1}$$

The advantage of this representation is that the calculation can be carried out iteratively. If, for example we want to update the flow, then we need the two end points of the potential $p = P[i_0]$, $p^{+1} = P[i_1]$, $f = F[i_2]$. Here P and F represent the field of the potential and the flow. The update, $f = f - \tau(p^{+1} - p)$ can be calculated for every point as $F[i_2 + q] = F[i_2 + q] - \tau(P[i_1 + q] - P[i_0 + q])$. Iterating q from 0 to $\prod d_i$ will update every point in the flow. The only thing to consider is what happens to the border points of the image. If $I[q]$ is a border point, $I[q + 1]$ is also a border point on the other end in the consecutive line or the top of the next plane (see fig. 3.3.). In this setup even if we calculate the pressure and the flow at the border points, they will always be 0. This way there is no interference between the opposite borders of the image.

The reference implementation collects the times of segmenting the reference images. The function is implemented as `maxflow` function from Pink [29]. The times are then collected and the values are compared with the optimized versions.

3.3 Maxflow on x86-64 Architecture

The x86 is the most widespread computer architecture in the world today. Its ongoing success is mainly due to its backward compatibility. Indeed the

[12] in the implementation the sink can be any set of points that includes the points of the border

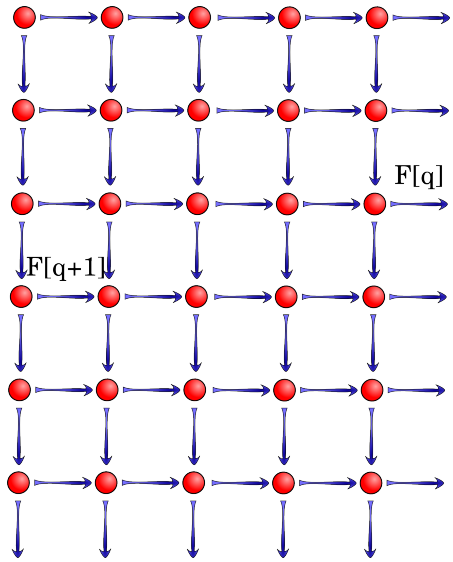


Fig. 3.3: The representation of the maxflow system in the memory. The fields (images) are represented with linear arrays. In the conceptual model the flow is shifted right and down with half pixel. The red dots represent the points and the blue arrows represent the flow. The consecutive memory block of $F[q]$, representing a border flow of the image, is the first border flow of the next line, named $F[q+1]$.

contemporary instruction set is a superset of the original i386 instruction set introduced in 1985. The x86 is using a complex instruction set. This means that the processor supports relatively many instructions. The instructions can belong to different categories and one instruction can perform several operations[13]. As the instructions can be of different type, the processor is divided into different *modules* or instruction units. Also, as the instructions can be complex, the processor is using *instruction pipes*. The internal implementation is using several optimization techniques, however in any case the result of the computation is guaranteed to be that of the in-order execution. This means that the result of the computation could have been obtained by executing the instruction in the order of reception.

[13] like (multiply and increment) or (conditional jump)

3.3.1 Parallelism on x86

Historically the x86 architecture has been a sequential architecture. Its memory and synchronization subsystem had been added only after its performance increase started to rely on the multi-core paradigm. Even if the architecture has been reimplemented several times since its first publication, as its instruction set remains similar, there are some properties which make an efficient parallel implementations difficult. The parallel computations on x86 are divided into threads. Threads are essentially fully featured processes which share common memory. Also if the number of threads n equals the number of physical cores on the system, the execution can be understood as n independent processors executing n individual processes. If we want to synchronize the processes, we have to use inter-process communication. The synchronization penalty is discussed in section 3.3.1.3. The different cores also access the memory with different speeds and latencies. The memory model will be discussed in section 3.3.1.1.

3.3.1.1 Non-Uniform Memory Access

The CPU is connected with the memory through several busses. Each of the busses can be considered as a communication channel between different parts of the system[14]. The function of the bus can be modeled as follows. The time of the system is divided by a periodical signal, the clock. The bus can only change its state when a clock signal comes. Between the signals the bus connects exactly two items in the system. The rest of the system is *de facto* isolated. The two parts can be the processor and the memory, or the processor and the hard-disk. On systems with more than one processor die, the processors themselves communicate via busses. On a system with several CPU dies the bus speed and hence the memory bandwidth is divided. Imagining a computer with four CPU dies, the effective memory bandwidth would be $\frac{1}{4}$ of the bus speed. This would be very wasteful, and therefore many systems possess a more complex memory architecture. As can be seen on fig. 3.4, the memory is divided in different nodes and each node is connected with a different CPU die. Therefore the CPU can access its directly connected memory much faster than the memory of the neighbor nodes, for which it would first had to have the processor of the other node to fetch the memory and then transfer the information on the CPU interconnect bus. In the ideal case the dies are fully connected to each other. This means that there is a separate bus between each CPU die on the system. On bigger systems and on possible future systems the dies will not necesserily be fully connected. The *distance* between two CPUs is anticipated in the NUMA API. This means, that there is a symmetric distance matrix which represents the distances

[14] for example the CPU and the system memory

between the CPUs. A parallel program therefore has to take into acount the access times to different nodes. Ideally a program should allocate the memory in its own node.

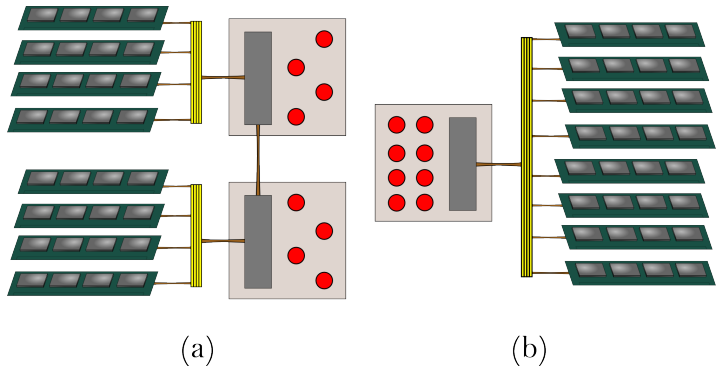


Fig. 3.4: This is a conceptual simplified schema of the Non-Uniform Memory Architecture. The memory modules are marked with green. The processor cores are marked with red. Every module on the system is connected with the FSB (Front Side Bus) marked in yellow; note that we assume that the processor is directly connected to the RAM modules, whereas in reality usually an interface chip is used, often called the NorthBridge, which is not represented here. Between any two clock signals the bus connects exactly two parts of the system. The other parts are practically isolated. In the conceptual schema of a non-NUMA system (b). As the processors can only be connected to a single memory module at a time, the effective memory bandwidth is the bus speed divided with the number of the memory modules. On a NUMA system (a) the computer has the same theoretical processing power (number of CPUs or cores), but it has two front side buses. The effective optimal theoretical memory bandwidth is twice as much as the compared system, but if a core fetches memory from a neighbor node, the processor of the neighbor node will be kept busy during the transfer.

3.3.1.2 Measuring the Memory Bandwidth

The apparent memory bandwidth can be measured with a simple program. We choose the array size M; we create N floating threads, and each thread allocates an array of size $\frac{M}{N}$. Then the arrays are copied front-and-back and

the measured wall time[15] is averaged. In the end, we divide the measured time with the quantity of the copied data to get the memory bandwidth.

We have carried out the measurements repeatedly on arrays of different size. The measurements are summarized on fig. 3.5. We can note several interesting phenomenon in the memory access. The first is that the apparent memory bandwidth is huge if the size of the array is small enough. As expected after a given array size, the memory bandwidth stagnates. This is caused by the cache. The cache is a small amount of memory close to the processor. It has a very fast bandwidth, but it is limited in its size. The cache represents a data in the memory. If the data has not changed in the memory, than the processor can read the data from the cache. For the small arrays, which fit into the cores' caches the apparent memory bandwidth is closer to the caches' speed. For this, the processor has to be aware that the data has not changed.

The second observation is, that the memory speed increases linearly only up until at most four threads after which it reaches a plateau. This is due to the fact that the memory bandwidth is much lower than the processing power, that is to say, the processor core would be able to process much more data than the memories can feed them. We could say in streaming applications the cores can be *starving* for data.

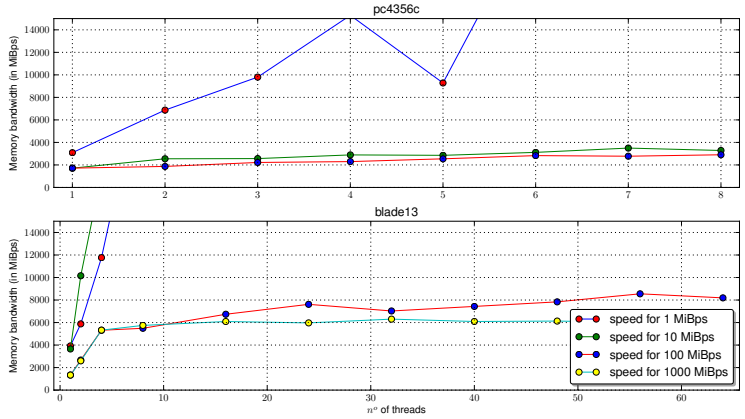


Fig. 3.5: The simple memory bandwidth graph on pc4356c and blade13.

[15] Wall time is the perceived time for the task from start to finish. It includes sleeps and waits for the resources. One can imagine the wall time as the time that passes by on the clock that is hanging on the wall in the room.

The memory bandwidth can also vary upon the distribution of the memory within the system. As we mentioned earlier, a core can access the memory on its own node much faster than on the neighbor nodes. The NUMA bandwidth can be measured a NUMA-conscious operating system. The current Linux kernel supports NUMA. There are dedicated system calls, but the recommended approach is to use the `libnuma` shared library. We will demonstrate the effects of the memory hierarchy on the memory bandwidth. We use the same function as above (which copies M bytes front-and-back), then we have created several threads and had them call the copy function with $\frac{M}{T}$ bytes each, where T is the number of threads. However, with the `libnuma` API we perform the tests twice with different scheduling policies. In the first part of the test we ask the scheduler to only schedule threads on cores belonging to the same NUMA-node. In the second part we create T threads as well, each thread to copies $\frac{M}{T}$ bytes, but we divide the threads in N groups, where N is the number of nodes on the system. This time we ask the scheduler to restrict the threads of the group on the node of their specific groups.

In practice the cumulated memory bandwidth is the sum of that of the nodes and can reach up to 6GiBps. The concrete bandwidth tests are collected in chart 3.6. It is therefore desirable to place the data to the memory node of the processor on which the calculation will run. We have developed a class named `pink::numa::poly_array_t` which behaves like an array, but distributes the array equally between the NUMA nodes on the system. Therefore if the points of the iterations are distributed equally among the threads, we can maximize the likelihood that a thread accesses the memory in its own node. Also our class behaves like a conventional array, so if the memory is displaced, typically on the borders of the chunk, the class will implicitly return the correct reference.

From the memory speed we would expect, that smaller the image is, faster the iterations runs. In practice however the memory speed is counter-balanced by the side effect of synchronization, which we will discuss in the next section.

3.3.1.3 Synchronization

During the maxflow iteration the threads have to be synchronized after updating the potential, the flow and the constraint. This means, that the algorithm is synchronized three times per iteration. In a typical segmentation of 3000 iterations, this means $3 \cdot 3000 = 9000$ synchronizations. These synchronizations are expensive by themselves. The reason is the complex instruction set of the x86 architecture. Namely, the commands are executed in *pipelines*. We can imagine the pipe as a queue, we insert the instructions in-order and we receive the results of the instructions in-order. However, the pipe can contain several instructions at once in various states of execution. When a thread arrives to a barrier, it can no longer insert any command into the pipe, as the data for those commands is not yet ready. Therefore the pipe becomes

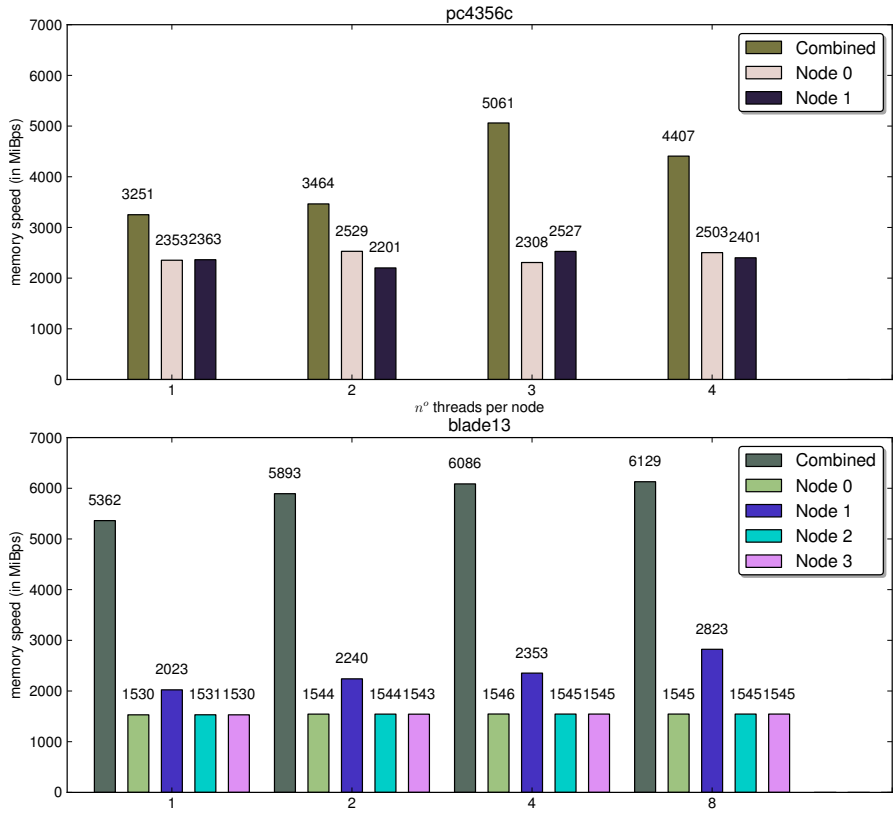


Fig. 3.6: The effects of the NUMA hierarchy on the apparent memory bandwidth.

empty. When all the threads are finished with the execution, and the signal arrives, that the thread can continue, the pipe needs to be filled again, this means, that if pipe contains 10 instructions, it will be idle after the barrier for 10 instruction times.

To demonstrate this, the code 3.1. is executed on the test computers with increasing thread concurrency. The test times are collected in fig. 3.7. The test results show, that even if the threads stay completely idle, the synchronization consumes a lot of time. From these measurements we can do some estimations concerning the performance of the maxflow. The estimations will be carried out in the next section.

To demonstrate the neccesity of the synchronization, we carry out the following experiment. In the normal iteration, we have to synchronize the threads three times per iteration. We will omit some synchronization steps and examine the result as well as the iteration time.

Listing 3.1: Time needed for synchronization

```
template <class T0, class T1>
void synchro( T0 iterations, T1 barrier )
{
  for (index_t q=0; q<iterations; q++)
  {
    barrier->wait();
  }
}
```

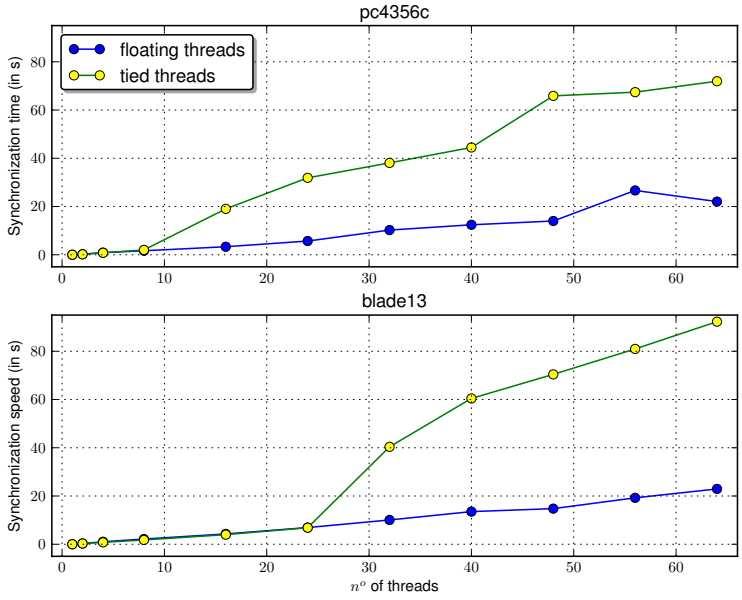


Fig. 3.7: Synchronization speed. A given number of threads is created and the threads synchronize 90000 times simulating the fine-grained parallel maxflow algorithm with 30000 iterations.

3.3.1.4 The Overhead of the Synchronization

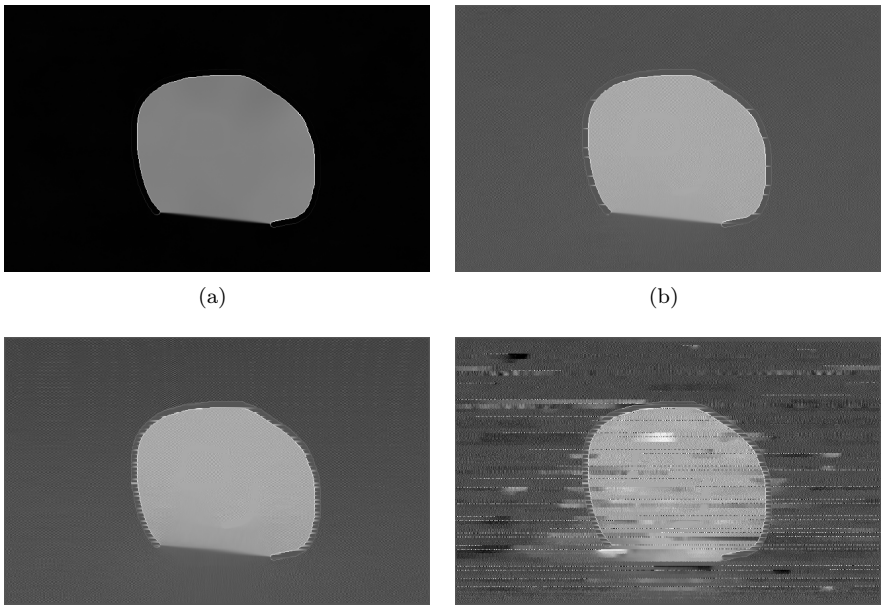


Fig. 3.8: The degradation rate of the segmentation without synchronization. During the 30000 iterations we have performed 100% (a), 66% (b), 33% (c) and none (d) of the synchronizations between the threads. Note that the actual fields diverge, so we have cut of the extremal values (above 2 and belov -1).

We can easily estimate the actual overhead of the synchronization of the algorithm. As all the threads know the number of iterations, we can switch off the barriers. This way the threads carry out the same amount of operations and memory access, but the cache coherence and more generally the order of the flow–potential–constraint update will no longer be guaranteed. The performance in these cases can increased up to four times as we can see in the fig. 3.9.

The degraded segmentation results are collected on fig. 3.8. We have not made detailed considerations, but it is worth to note, that at least visually the results are close enough to the precise results. It is also important to note, that these artefacts cannot be reproduced with GPU-s. These results indicate that the overhead can be eliminated on the hardware level, if the architecture is designed with fine-grained parallelism in mind.

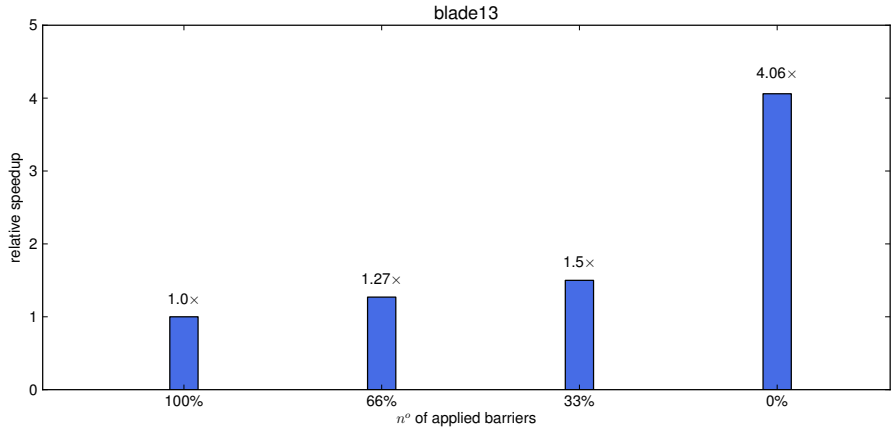


Fig. 3.9: The synchronization penalty of maxflow.

3.3.1.5 Estimated Computation Time

In parallelization of maxflow there are two contradictory forces. If we have small images, the synchronization will constitute a considerable overhead on the computation time. On the other hand if the size of the image increases, the cache will not be able to counterballance the slow memory, and so the performance will be limited by the memory bandwidth. Nonetheless, we will demonstrate in the next section, that the we can achieve performance increase up to the increase in the memory bandwidth and the synchronization penalty.

Let us assume a memory bandwidth v and a single synchronization time t_0 for a segmentation of n iterations. For the segmentation we need approximately M bytes of memory access as estimated in section 3.1.2. The time t needed for the iteration would therefore be higher than

$$t \geq \frac{M}{v} + nt_0 \tag{3.3.1}$$

that is to say the time needed to supply the memory to the processors plus the time needed for the synchronization. If we substitute (3.3.1) into the measured values we get the comparative table 3.10. In the case of large enough images we get estimated computation times, which are comparable with the real measured computation times. This means that in the case of big images, the processors are indeed starving for data and that the computation is approaching it's limits. In the case of small images the apparent memory bandwidth is much higher, so (3.3.1) is no longer a close estimation.

		estimated computation time							measured
	nbt	**1**	**2**	**4**	**8**	**16**	**24**	**32**	**simdflow**
2D	**pc4356c**	104.3s	47.11s	21.89s	17.58s				42.19s
	blade13	88.22s	32.02s	17.57s	13.1s	10.42s	11.37s	43.92s	49.16s
3D	**pc4356c**		220.49s	206.93s	162.66s				244.47s
	blade13			133.68s	121.64s	117.79s		117.09s	126.87s

Fig. 3.10: Estimated iteration time based on apparent memory bandwidth and the synchronization penalty. If substitute the measured apparent memory bandwidth and the synchronization time in (3.3.1), we get the *estimated computation time*. Note that this is a lower estimate. We can compate these times with the measured fastest segmentations using the `simdflow` function from [29]. We can note that the fastest 3D segmentation is close to the lower estimate (117.09s vs. 126.87s). Also the estimate is no longer sharp for small images (10.42s vs. 49.16s). Note that estimated time increases as the number of threads increase as it takes longer to synchronize many threads.

3.3.2 Streaming Algorithms on x86

Most of the contemporary x86 processors possess a SIMD[16] module. This instruction set has been developed to assist the growing need of real-time gaming experience and video transcoding [31]. As we will demonstrate it is also useful for general streaming algorithms and more particularly for maxflow. It operates on relatively short vectors of 128 bits. At each computational step the SIMD module carries out the same operations on every part of the vector which, generally, consists of four integer or four float values. As the SIMD operation takes the same time as a general operation, it can speed up an implementation up to four times compared to the floating point module. Unfortunately, this speedup is mostly theoretical, as the data has to be aligned and set up in the floating point registers. In addition to the alignment the SIMD module will still be bounded by the memory bandwidth. However, if it's possible, it is still important to vectorize an algorithm, as the processor will be less busy and it can use the idle time to prefetch the data from the memory or to maintain the coherence of the cache.

In the maxflow algorithm we carry out the same operation on every pixel. This makes the maxflow a good candidate for streaming. We can put 4 float values in a SIMD vector. Therefore, we first normalize the size of the image, so each of its dimensions would be divisible by four. In theory it would be sufficient to convert ensure that the x dimension divisible, but we can eliminate every side condition that could pose a special case, and the resizing has little

[16] **S**ingle-**I**nstruction **M**ultiple-**D**ata

data overhead. The SIMD instruction set also possess floating point conditions. With instructions `_mm_cmpn**_ps` we can compare two SIMD vectors of four floats and the results of the appropriate comparisons will be stored in a vector of floats. The values can be set to 0 and 1. Moreover sse4.2 implements an instruction `_mm_blendv_ps` which assigns one of its operands according to a conditional map.

The detailed discussion of how to *streamline* the maxflow algorithms is given in section 3.4.2. In this section we are going to discuss the treatment of special cases which are implied by vector processing. All the sizes of the image are divisible by four. This means that all but one of the neighbor pixels are four-divisible-distance away from each other. In the memory representation defined in chapter 3.2. The only difference is, that in SIMD we have an array of SIMD vectors[17] instead of a single floating point value. A given point in the image can be found as

$$I[n] = I\left[[p_0, \ldots, p_{d-1}]\right] = I\left[p_0 + p_1 \cdot \frac{\prod_{d-1} d_q}{\prod_{q=1} d_q} + \cdots + p_{d-1} \cdot \frac{\prod_{d-1} d_q}{\prod_{q=d-1} d_q}\right] \tag{3.3.2}$$

$$= I\left[p_i \cdot \frac{\prod_{d-1} d_q}{\prod_{q=i} d_q}\right] \tag{3.3.3}$$

in (3.3.2) the double brackets represent the native coordinate access, where the simple bracket represent the linear memory access[18]. In (3.3.3) the i is summation index. As the points are paired by 4 the point $I[n]$ will be represented as $SI[n \text{ div } 4].f[n \mod 4]$ here SI is an array of SIMD vectors.

A neighbor point can be found as

$$I[[p_0, \ldots, p_i + 1, \ldots, p_d]] = I\left[n + \frac{\prod_{d-1} d_q}{\prod_{q=i} d_q}\right] \tag{3.3.4}$$

$D = \frac{\prod_{d-1} d_q}{\prod_{q=i} d_q}$ is divisible by 4 for $i > 0$ as d_0 is present in all but the first difference. The neighbor of a point in the SIMD vector will be $SI[n \text{ div } 4 + \frac{D}{4}].f[n \mod 4]$. In the previous expression $\frac{D}{4}$ is an integer value, as D is divisible by 4 and $n \mod 4$ is unchanged. This means that expressions of

[17] four floats

[18] note: the both accesses are implemented in Pink

type $f = f - \tau \cdot (p^{+1} - p)$ can be calculated directly on SIMD vectors without any shift or shuffle.

In the x direction this equality does not apply. The neighbor point of p in direction x can be in the same vector as p. In the x direction, therefore we have to shift the values to get the proper vectors. This shuffle becomes less and less significant with the dimension increase, as in 3D the only one third of the operations are shifted, and in 4D only 20%. After the vectors have been properly loaded the necessary operations can be carried out in order as in section 3.4.2. It is important to note, that no new branches are introduced as direction x can be updated with a different function. Also because of the size normalization no special cases were introduced during the vectorization. The same implementation would also be reusable with minor modifications for longer vectors as for example the Itanium architecture.

3.3.3 The Benefit of the Cache in Accelerating the Performance

Historically memory throughput has increased much less in proportion than the theoretical performance of the processors [32]. This is true for every architecture, x86 is not specific. In the current memory and processor design, the frequency represents a major constraint to the performance [33]. Furthermore due to architectural details random memory access latency increases with the memory size. To avoid this, processors incorporate a local fast and small memory, the *cache*. This memory is closer to the processor than the RAM, and is significantly smaller, causing be much faster and to have much lower latency than the RAM.

A segment of the memory is loaded into the cache of the processor at each memory read. This segment stays in the cache and the system will be conscious of its existence and the place in the RAM through cache coherence. If the processor only accesses the memory which is in the cache, then the memory read can be omitted, considerably augmenting the performance. The cache coherence is automatic and implemented in a proprietary module in the computer, so its functioning cannot be accurately predicted, but its functioning was demonstrated with a simple experiment measuring the apparent memory bandwidth in section 3.3.1.2.

In our research, we have found that the maxflow algorithm gives the best performance, if the image is statically distributed between the processor cores. This way if the image is small enough to fit in the cumulated cache of the processor cores, the memory can be kept isolated except for the parts which need to be synchronized. The effects of avoiding the synchronization will be discussed in section 3.3.1.4. In the SIMD version of the maxflow algorithm the given amount of hardware threads is created in the beginning of the iteration. These threads are then distributed equally on the memory nodes

and are then tied to a specific core on they appropriate nodes. This way a double benefit is attained. Firstly each thread will work on the same part of the image (potential and flow), so we maximize the likelihood that the image will still be in the core's cache. Also if the memory is not in the cache and it has to be fetched, the memory will be distributed equally between the nodes, so the RAM throughput can also be maximized. Eventually as the size of the image grows, the computer will have to rely on the RAM bandwidth, but even in this case the bandwidth will optimal compared to the given computer.

Also because a heavy use of the SIMD instructions will lower the load on the CPU cores, the processor will have time to prefetch the memory needed for the next calculation.

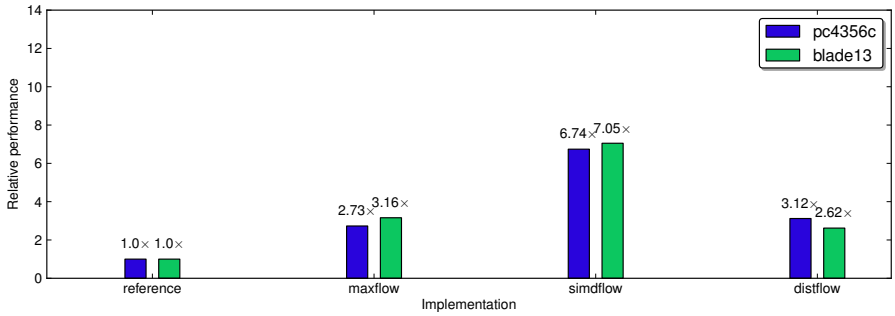


Fig. 3.11: Relative performances of the maxflow implementations in 2D.

3.3.4 Benchmarks

The performance have been measured and compared to the reference implementation. The results can be found for 2D images on fig 3.12 and for 3D images on 3.11. In the case of 3D images the performance improvement is greater and we also know that we approach the physical limits of the given systems. In the case of 2D images the relative improvement is lower as the cache already accelerates the segmentation even on a single thread.

3.4 Maxflow in the OpenCL Framework

From the developer's point-of-view GPGPUs represent a choice, which renders the developer considerably dependent on the hardware vendor. The in-

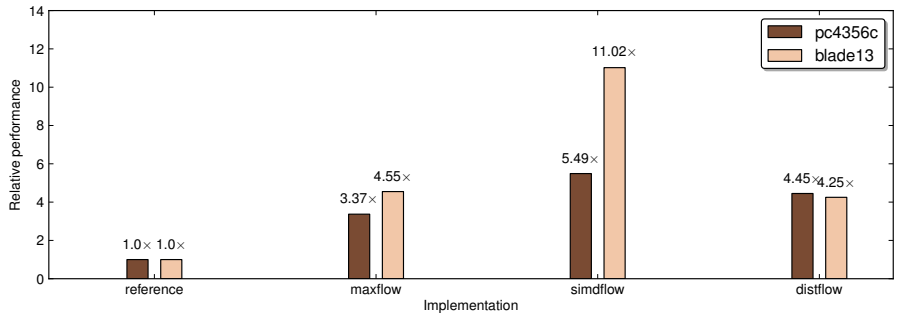


Fig. 3.12: Relative performances of the maxflow implementations in 3D.

struction set of the contemporary GPU is highly proprietary and different across vendors. Indeed there are no low level API's or instruction sets that are shared among vendors. This setup has several drawbacks. Firstly any software development will be forever dependent on the GPU vendor. As vendors have a vested interest in maintaining a continuous hardware supply and driver support, any strategic change of direction for them could turn into a loss of investment for developers. Secondly the developed software has to be anticipated in some sense by the vendor. Indeed, some APIs[19] are highly specialized and are difficult to use for general programming.

To address the question of durability and compatibility The Kronos Group maintains the OpenCL API standard. It is a general purpose programming API and framework optimized for GPU architectures. It allows general programming on the somewhat specialized SPEs[20] using a reduced feature (instruction) set.

This standard is useful in two ways. Firstly if adopted by the majority of the vendors, then it can provide a common framework for GPGPU software development. Secondly as it is a subset of the C language, it can be easily ported to any platform (multi-core x86 or CBEA[21]). This way, even if eventually OpenCL will ceased to be supported by some hardware vendors it could be ported to other architectures[22].

The OpenCL framework is designed to program massive multicore hardware. The programs are divided into *fibers*. The fibers have some key differences compared to traditional threads. Fibers run *kernels* instead of processes and the order of the kernel execution is undefined. This means that the system does not guarantee the order of the operations. The onus is therefore in the implementation to create kernels which can be executed independently.

[19] Like OpenGL or Physix

[20] Synergic Processing Elements

[21] Cell Broadband Engine Architecture

[22] even if it may not necessarily provide optimal performance on every platform

Listing 3.2: An example loop in C++

```
for ( index_t q = 0; q < size; q++ )
{
  c[q] = a[q] + b[q];
}
```

The kernels are restricted to a substantially reduced feature set compared to the threads. In most cases, however, the fiber execution is self-synchronized. This means, that if two cores start to execute two kernels on the same clock signal, then the kernels will finish the execution on the same clock signal[23] This phenomenon is not guaranteed by OpenCL, but it is a common property of the GPU architecture. The corollary of this is that while the synchronization represents considerable trouble and overhead on x86, it is a virtually non-existing problem on GPUs.

The OpenCL framework consists of two major parts. The kernels can be programmed in a reduced C-like language, while the rest of the API manages the kernel compilation, execution and the memory transfer between the device and the computer. All the SPEs access the same global memory. The calculation is divided into tasks, where the tasks are either executed in order or out of order. In practice the in-order execution for the maxflow algorithm has little overhead.

OpenCL works as follows. The developer creates the so called kernels. Each kernel is *de facto* the statement scope of a C/C++ like for loop of one counter variable. For example the loop 3.2. could be translated into the kernel 3.3.

The kernels are then called as tasks with the limits specified. The task's execution is the divided into chunks by the API and the statements are executed on the SPs. The counter value is loaded as a quasi-parameter. There is no guarantee of the order within the loop, so the statement scope must be independent. The execution can be synchronized only between the tasks.

The OpenCL kernel performs best in the case when all the kernels execute the same binary path. There are therefore two major challenges in the OpenCL implementation: 1. creating independent kernels 2. eliminating the branches from the execution . In the next section we will discuss the necessary adjustments of the maxflow algorithm for the OpenCL kernels.

[23] That is assuming that the two kernels have the same branching during the execution.

Listing 3.3: A simple kernel in OpenCL

```
__kernel void forloop(
  __global float * a,
  __global float * b,
  __global float * c,
  __const  int length )
{
  size_t id  = get_global_id(0);
  if (id >= length) return;
  c[id] = a[id] + b[id];
}
```

3.4.1 Maxflow on GPGPU

The GPU[24] is still an emerging architecture as of 2011. Consumer level dedicated graphic processors emerged in the 1990-s. At that time, the algorithms used for visualization and 3D rendering, particularly raster graphics, have been significantly different from common x86 software algorithms. As demand for improved graphical output increased, the x86 architecture proved increasingly poorly adapted for visualisation. The x86 architecture has therefore been amended with SIMD instructions to better accommodate the vector calculation typically abundant in 3D raster visualization. Eventually the x86 architecture has been outperformed by dedicated GPUs. As the sophistication of graphic procedures improved, GPUs became increasingly flexible and powerful, and by the early 2000s even general-purpose calculations became possible.

One of the advantages of the fiber approach is, that the kernels can be partitioned along the memory. Indeed, the GPUs have a very fast, uniform and almost linear scaling of the memory access. This means, that unlike in the x86 architectures, the memory bandwidth remains constant and high even if the memory is fully allocated. This property is demonstrated on fig. 3.13. We can see, that if we double the number of the pixels in the image the computation time doubles as well. The linear increase continues up until the size of the available memory on the card. This will also improve the proportional performance increase between the 2D and 3D images compared to the x86 architecture.

The main deficiency of the SPA[25] from the point of view of the maxflow is, that it does not support branches. On the NVidia Fermi architecture for

[24] Graphical Processing Unit

[25] streaming processor architecture

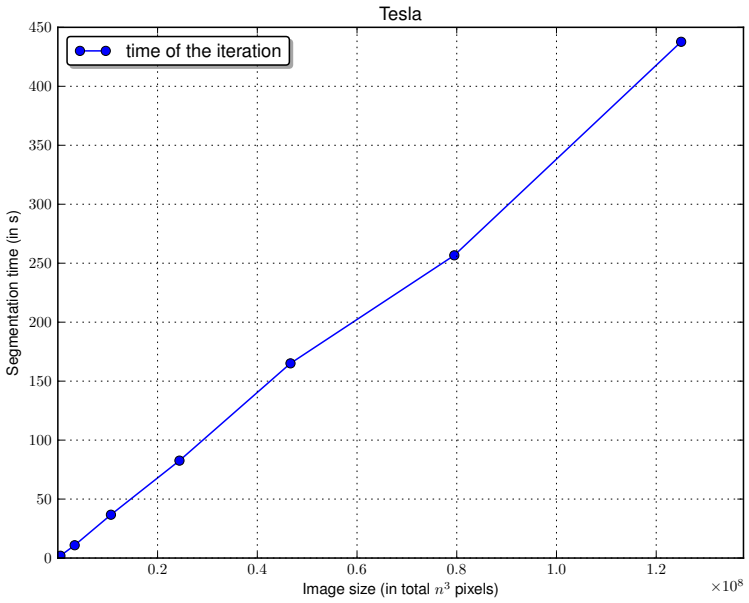


Fig. 3.13: Tesla segmentation of the fibers. 3000 iterations and diminishing image size.

example, for each 16 streaming processors there is one APU[26]. If, for example, an *if* statement is present in the code, then the SPEs have to reschedule the work to the APU. They re-assume they work only after the binary branch has ended. The main challenge therefore is to implement algorithms without branches, only using conditional operator[27] where possible instead.

The conditional operator is a mathematical branchless operator which attributes values according to logical operators. The classical if (cond) statement is implemented by the compiler as a jump in the object code. If the statement is true, an appropriate object code has to be loaded into the processor (SPE). The condition can be interpreted as a function which returns 1 if the statement is true and 0 if it isn't. An example is the comparison $\mathrm{lt}(a,b)$ and $\mathrm{notlt}(a,b)$ are conditional functions. From these functions, the conditional operator can be implemented as:

$$(a < b)?v_1 : v_2 \Leftrightarrow \mathrm{lt}(a,b) \cdot v_1 + \mathrm{notlt}(a,b) \cdot v_2 \tag{3.4.1}$$

[26] Advanced Processing Unit

[27] conditional operator or inline if (iif)

The key difference is, that in this case each SPE can execute the same binary code regardless of the result of the comparison.

3.4.2 Eliminating the Branches From Maxflow

As we saw in section 3.4. for an efficient implementation of the maxflow algorithm in OpenCL we need to eliminate the program branches from the algorithm. In this section we will discuss step by step the reformulation of the algorithm to this end.

Potential For the seek of memory bandwidth optimization, the update of the potential is merged with the updating of the boundary conditions. The first kernel of the flow looks as follows

1. if p is in the source then set p=1
2. if p is in the sink then set p=0
3. in each direction $p = p - \tau \cdot (f_o - f_i)$

The source points and the sink points are kept in a float array SS, which is set to -1 over the sink points, 1 over the source points and 0 everywhere else. This way the first two steps of updating the flow can be replaced by conditional operators:

$$p = (|SS| > 0)?0 : p \tag{3.4.2}$$

$$p = (SS > 0)?1 : p \tag{3.4.3}$$

whereas the third step is a simple floating point operation.

Flow The kernel for the flow is a conditionless update. It only states

$$f = f - \tau \cdot (p_{+1} - p) \tag{3.4.4}$$

Constraint The kernel for the constraint is divided in several steps:

1. We create a vector $\overline{\mathbf{F}}$ with the highest outward flow. The highest flow is the comparison between the inward and outward flows in each direction. The value can be determined in each direction with a conditional operator as:

$$\overline{\mathbf{F}}_i = (-f_{\text{in}} > f_{\text{out}})? - f_{\text{in}} : f_{\text{out}} \tag{3.4.5}$$

2. Calculate the absolute value of the outward flow. As the highest outward flow can be negative, we need to eliminate the values which are inferior to 0. We have managed that as follows

$$v = (\overline{\mathbf{F}}_0 > 0) \cdot F_0^2 + (\overline{\mathbf{F}}_1 > 0) \cdot F_1^2 + \cdots + (\overline{\mathbf{F}}_{d-1} > 0) \cdot F_{d-1}^2 \tag{3.4.6}$$

As the condition function returns 0 if the condition is false, the squares of negative values will be eliminated.

3. If the flow is higher than the constraint then we proportionally lower the flow value in each direction. To avoid the branch, create a proportion multiplier, which we set to 1 if the flow is smaller than the constraint and the proper quotient if the flow depasses. The quotient is $\frac{g}{\sqrt{v}}$. The condition is $v > g^2$. The two statements combined give us

$$q = (v > g^2)?1 : \frac{g}{\sqrt{v}} \tag{3.4.7}$$

- Finally if the directional flow is greater than the corrected outward velocity, than the flow is updated to the corrected velocity. This can also be corrected as

$$f_{\text{in}} = (f_{\text{in}} < -\overline{\mathbf{F}}_0)?f_{\text{in}} : -\overline{\mathbf{F}}_0 \tag{3.4.8}$$

$$f_{\text{out}} = (F_0 < f_{\text{out}})?\overline{\mathbf{F}}_0 : f_{\text{out}} \tag{3.4.9}$$

As we can see all the steps are transformable into stream operations, and so the kernel can be optimally compiled by OpenCL. The corresponding source code can be found in Pink under the function named `clflow`. In the following chapter we are going to extend the OpenCL implementation to other architectures.

3.4.3 The Cell Broadband Engine Architecture

The CBEA is one of the first truly heterogenous architectures in mass production. The chip consists of two IBM Power Processing cores (PPE) and 8 Synergic Processing Elements (SPE). The architectural construction of the PPE resembles the classical CPUs (x86, arm), while the SPE resemble the GPUs (GeForce, Radeon). One of the interesting aspects from the point of view of OpenCL is that IBM's OpenCL implementation treats the CELL processor as two devices. One for the PPEs and one for the SPEs. These have different performances for the same kernels, so the work has to be manually separated between these devices. This sharing does not involve copy between the devices as both PPEs and SPEs share the same memory.

The kernels developed for the chapter 3.4.1 are also usable on the CBEA. The limits of the iterations are distributed between the two devices. In the beginning each task is divided equally among the devices. After each task, the time usage is measured, and the work is redistributed among the devices using binary convergence. In our tests the optimal work ratio was 0.37 for the PPEs and 0.63 for the SPEs. As the PPEs are fully featured architectures, they are capable of executing a multi-tasking operating system. We can there-

fore compare the `maxflow` function of Pink with a slightly modified `clflow` function[28]. The standard `maxflow` function on the CBEA runs 2.56× slower than the `clflow` function. This is important because the `clflow` function is only a minor modification compared to the mainstream version. This experiment demonstrates that the OpenCL implementation can function on heterogenous or even complex multi-core architectures with considerable efficiency. Furthermore the `clflow` function on the CBEA slightly outperforms the reference implementation, whereas its design is at least five years older than the reference x86 systems.

3.5 Benchmarks

The implementations have been compared on four computers of three different architectures. The results are collected on fig. 3.14. We can see that the parallel version of the x86 implementation is an order of magnitude faster than the reference implementation, whereas the performance of the same algorithm can increase by two orders of magnitude on a GPGPU architecture. Both are compared with a highly optimized sequential version.

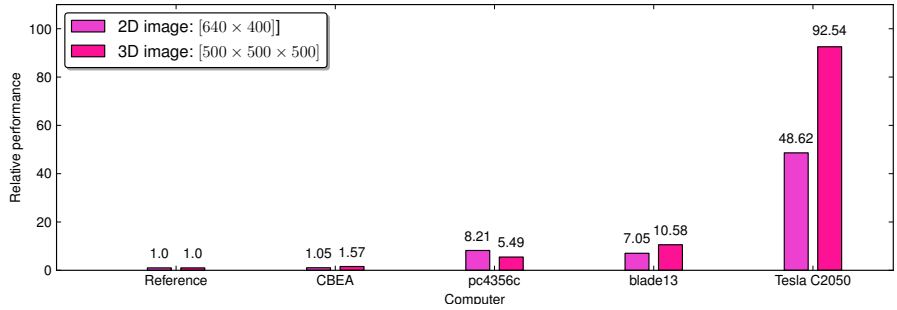


Fig. 3.14: Relative performances of computers using the maxflow algorithm on different architectures.

It is also important to note the relative performances compared to the price of the equipment. If we want to compare the segmentation time with the prices of the equipments, let t be the time of the segmentation on a given hardware and let p be the price of that hardware. Than the value eff_p can be defined as (3.5.1).

$$\text{eff}_p := tp \tag{3.5.1}$$

[28] The modified `clflow` is not part of the mainstream Pink, but it is included in the repository of the thesis.

eff_p indicates an absolute value of the PPP[29] unit of the hardware. The value is in linear rate with the performance and the segmentation time. This means, that longer the segmentation time or higher the price a higher value of eff_p. We are going to evaluate the eff_p for each machine. The smallest value is set to one and only the relative differences are presented. It can be seen on fig. 3.15, that there is an even more radical differences for this quotient. A GPGPU can be up to 500-600 times more economical than the x86 hardware. It is therefore important to investigate which algorithms are implementable efficiently on this hardware.

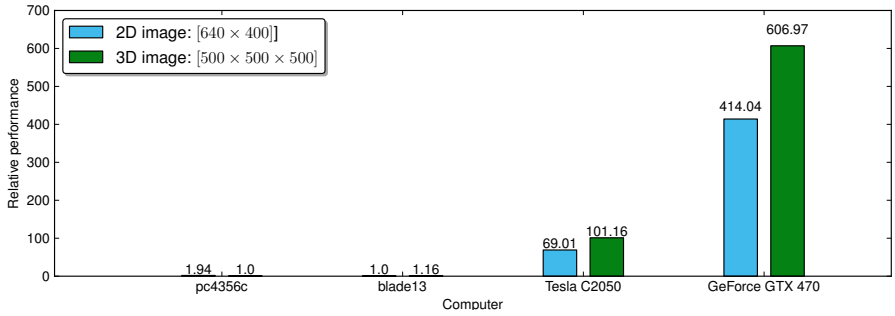


Fig. 3.15: Relative performance according to the price of the equipment. The ratio is estimated according to the hardware's best performance and its approximative price when purchased as new. The smallest eff_p is set to one.

3.6 Summary

In this chapter we went through the implementation of the continuous maximum flow algorithm. We have seen that there is an ongoing change in the processor architecture design paradigm, namely the increasing number of processor cores. As the parallelization has to be done manually and by algorithm we have isolated the major properties of the continuous maximum flow algorithm. We have reviewed the most common architectures: 1. x86 2. GPGPU 3. CBEA .

By estimating the necessary memory bandwidth and synchronization time we have proposed a model for the estimate of the algorithm's best possible performance on x86. After proposing measurements for the memory bandwidth and the synchronization time, we could demonstrate that our model was accurate for 3D images.

[29] **P**erformance **P**er **P**rice

We reformulated the algorithm by eliminating the branches, thus making and efficient implementation possible on the GPGPU. We have developed such implementation using the OpenCL framework.

After the comparison of all the available architectures we have concluded that the GPGPU architecture is the most suited for the continuous maximum flow algorithm, with the performance increase up to two orders of magnitude. That is unless we reach the memory limitations of the architecture, typically in the order of gigabyte images.

With the consideration of the CBEA we could demonstrate that the OpenCL implementation used on the GPGPU will remain reusable on heterogenous architectures.

In the end of the chapter we have provided detailed benchmarks on all the architectures. In the next chapter we look into the applications of the continuous maximum flow algorithm.

Chapter 4
Applications of the Maximum Flow Algorithm

One of the main advantages of the maxflow segmentation method is, that it is directly applicable to 3D images or more. With the modern means of image acquisition the quantity and availability of image has increased sharply. Also with the modern equipment and methods such as electron tomography, magnetic resonance imaging, tomography and synchrotrons, the size of acquired images increases as well. Efficient algorithm implementation is essential to carry out segmentations in a timely manner.

We have managed to apply the maxflow algorithm on several different image processing problems. The segmentation techniques that we have developed were used in silicate bead segmentation, liposomic membrane segmentation as well as medical image segmentation of the cardiac left ventrice.

A typical segmentation consists of several steps. The usual schema is a sequence of pre-processing–segmentation–post-processing. During the preprocessing we apply operators from mathematical morphology, or topology. Morphology is particularly useful for extracting the sources, which can then be used for accurate border detection. To facilitate creation of image processing flow we have developed an image processing framework [29]. All the described techniques are scriptable in the framework, and all the software used for the segmentation is publicly available.

The detailed description of the segmentation, the image types and the acquisition can be found in the appended article in section 4. and also in [34]. Some details of the framework is presented in section 5.

Journal of Microscopy, 2011
Received 2010-10-08; accepted 2011-03-31

Laszlo Marak[1], Olena Tankyevych and Hugues Talbot

Continuous maximum flow segmentation method

for

nano-particle interaction analysis

Journal of Microscopy RMS

2011-05-04

Université Paris-Est, Département d'Informatique Gaspard-Monge,
Équipe A3SI, ESIEE Paris, Cité Descartes, BP 99, F-93162 Noisy-le-Grand Cedex, France
{ujoimro, tankyevych}@gmail.com, talboth@esiee.fr

[1] PGP: A812 3708 B2E1 3B42 500E D50C 49A0 88FA C060 3F65

Abstract

In recent years, tomographic 3D reconstruction approaches using Electrons rather than X-Rays have become popular. Such images produced with a Transmission Electron Microscope (TEM) make it possible to image nanometer–scale materials in 3D. However, they are also noisy, limited in contrast, and most often have a very poor resolution along the axis of the electron beam. The analysis of images stemming from such modalities, whether fully or semi automated, is therefore more complicated. In particular, segmentation of objects is difficult. In this article, we propose to use the continuous maximum flow segmentation method based on a globally optimal minimal surface model. The use of this fully automated segmentation and filtering procedure is illustrated on two different nano-particle samples and provide comparisons with other classical segmentation methods. The main objectives are the measurement of the attraction rate of polystyrene beads to silica nano-particle (for the first sample) and interaction of silica nano-particles with large unilamellar liposomes (for the second sample). We also illustrate how precise measurements such as contact angles can be performed.

Keywords: electron tomography; image analysis; continuous optimization; Hough circles; transmission electron microscopy.

4.2 Introduction

In this paper, we study the application of image analysis to nano-tomography images and we present an image segmentation technique for this purpose. Image segmentation is the task of decomposing an image into a set of disjoint components that are each semantically consistent within themselves (e.g. finding the red blood cells in histology samples, or people in a photograph). It is an essential task for further image-based studies since it enables measurements to be made on objects, which otherwise would be indistinguishable collections of pixels. Segmentation is one of the fundamental tasks of computer vision, and there exist no generic method to achieve it. In the rest of the paper, we will perform segmentation by finding good-quality contours (in 2D) or surfaces (in 3D) around objects of interests.

Here, we focus on the automated segmentation and interaction measurements of nano-particles in electron tomography. While there exists a number of papers that have already applied image analysis to nano-particle studies [35, 36], segmentation of such particles can be especially difficult, for reasons outlined below.

Motivation

Nanometer-scale particles possess singular physical and chemical properties due to their dimensions, which have motivated a rapidly growing interest in recent years [37, 38]. These new-found properties have led to the novel applications of these nanomaterials to many areas, such as catalysts, semiconductors, sensors, drug carriers, and personal care products [39, 40, 41].

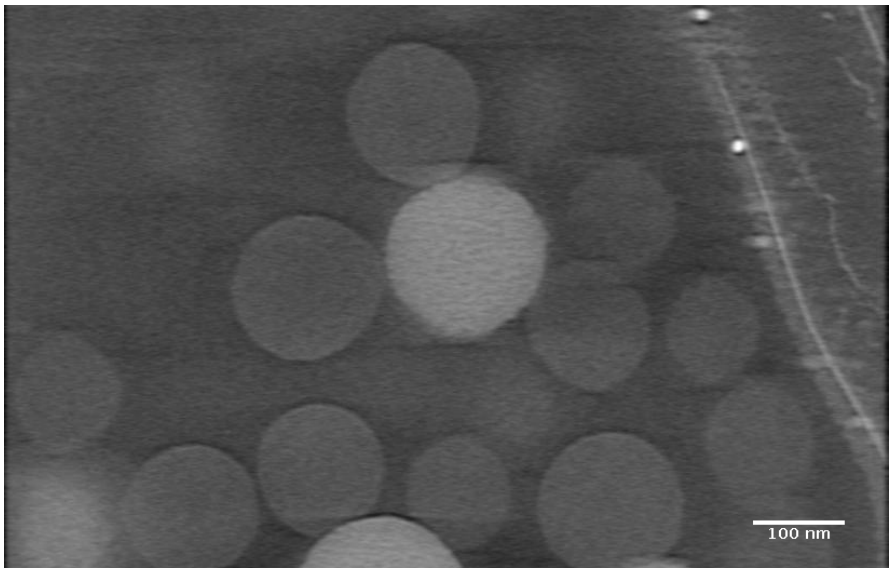


Fig. 4.1: A tomogram slice on the polystyrene beads nucleated around a silica one.

Commercial products including engineered nano-materials in their make-up are expected to become more frequent in the near future. As a consequence, both consumer and professional exposures to these materials are likely to increase in proportion to their use in the society. However, these nanomaterials can be potentially harmful to human and environment due to the large percentage of atoms lying on their surface and unusually high reactivity [42].

It is therefore important to study nano-materials both at the chemical and physical level. For this, Transmission Electron Microscopy (TEM) is the method of choice for nano-particle samples [43]. However, while standard TEM can provide sufficient two-dimensional resolution, it has insufficient depth sensitivity to detect internal three-dimensional structure. The main limitation is that it is a 2D projection of a 3D object. The technique does detect internal 3D structures, but as a 2D projection. To palliate this problem, electron tomography, initially proposed by W. Hoppe in the 1970's [44], has become increasingly popular [45] in order to obtain 3D views of nano-

scale materials. Electron tomography works broadly on the same principles as X-ray tomography, but uses electrons instead. Also, instead of a dedicated instrument, standard TEM equipment with relatively minor add-on can be used to acquire the data.

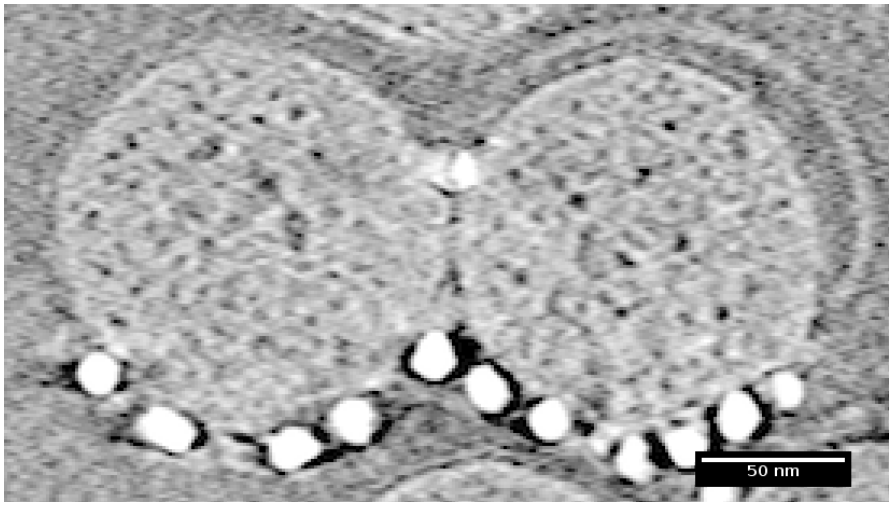


Fig. 4.2: A tomogram slice of silica nano particles with large unilamellar liposomes.

The sample under study is placed in an automated stage control, which tilts at regularly spaced angles and the microscope digitally acquires a projected image. Tomography software is then used to perform a 3D reconstruction from these projections. Due to limitations in the achievable tilt angle, the resolution of the images that are obtained is usually very poor along the electron beam axis, and can also be noisy and feature low contrast. We recommend [46] for further reading.

In this work, we are interested in assessing two different nano-article interaction studies with image analysis techniques. For the first one, the objective is to get an insight into the surface contact of polystyrene beads with silica nano-particles extracted from tomographic reconstructions. One slice of a sample can be seen in Fig. 4.1 [2]. The interest of the segmentation lies in the measurement of the attraction rate of the beads to the nano-particle. The attraction is measured by the contact angle.

For the other sample, we focus on interaction of silica nano-particles with large unilamellar liposomes (see Fig. 4.2 for an image slice example) [3].

[2] The material and the problem is described in more detail in [47].

[3] More details on the material for this application can be found in [43].

Automated Image Analysis

Once the image is obtained, the next step is often to analyze the content through for instance particle counting, size measurements, composition and interaction studies. While a manual analysis of nano-tomography data is of course possible [48], it is often not desirable. One reason is that the data is inherently 3D, which is difficult to represent on paper or on computer screen, and difficult to interact with. Although 3D viewing software packages have made enormous progress, interactive segmentation (i.e. finding the contour or the enclosing surface) of objects can still take months. Typically, practitioners endeavor to detour objects in 2D interactively slice by slice, or they try to set a global threshold in order to find a suitable grey-level iso-surface. Specifying surface elements manually is generally infeasible. Specifying contours slice-by-slice is technically possible when contours are easily visible, but even then can lead to inconsistent topology [49]. It is also very difficult when contour information is not reliable, as is the case in nano-tomography. When noise is present in high levels, finding a suitable iso-surface by thresholding can prove challenging and time-consuming. There is also a potential for human error due to fatigue, perception bias and operator variations. These effects have been well-documented in 2D studies and also in 3D in the context of medical imaging, which is similar in many respects [50] Therefore, there is a need for methods that are able to find the contour of objects reliably, even in the presence of high levels of noise.

On the other hand, while human operators find it typically difficult to detour objects in 2D or 3D, they are usually able to identify objects reliably and so are able to guide segmentation through interaction. It is therefore important to provide a segmentation method which allows operators to interact easily with the results obtained [51].

The Need for a Specific Segmentation Method

Often sufficiently precise contours are necessary for accurate segmentation of image data. For this, we need the following conditions to be satisfied by a segmentation method: 1) have objective optimization criteria; 2) feature few arbitrary parameters; 3) be little sensitive to noise; 4) be able to optionally interpolate missing data due to the missing wedge effect; 5) allow interactivity; and 6) feature as few inherent artifacts as possible.

Since the late 1980s, optimization methods have been used to address a wide variety of problems in computer vision, including segmentation. Early optimization approaches were formulated in terms of active contours and surfaces [52] and then later level sets [53]. These formulations were used to optimize energies of varied sophistication (e.g., using regional, texture, motion or contour terms [54]) but generally converged to a local minimum, generating

results that were sensitive to initial conditions and noise levels. Consequently, more recent focus has been on energy formulations (and optimization algorithms) for which a global optimum can be found.

The max-flow/min-cut problem on a graph is a classical problem in graph theory, for which the earliest solution algorithm goes back to Ford and Fulkerson [55]. Initial methods for global optimization of the boundary length of a region formulated the energy on a graph and relied on max-flow/min-cut methods for solution [56, 57]. It was soon realized that these methods introduced a so-called grid bias (also called metrication error) for which various solutions were proposed. One solution involved the use of a highly connected lattice with a specialty edge weighting [58], but the large number of graph edges required to implement this solution could cause memory concerns when implemented for large 2D or 3D images.

To avoid the gridding bias without increasing memory usage, one trend in recent years has been to pursue spatially *continuous* formulations of the segmentation problem [25, 10, 59]. Historically, a continuous max-flow (and dual min-cut problem) was formulated by Strang [15]. Strang's continuous formulation provided an example of a *continuization* (as opposed to *discretization*) of a classically discrete problem, but was not associated to any algorithm. Work by Appleton and Talbot [10] provided the first PDE-based[4] algorithm to find Strang's continuous max-flows and therefore optimal min-cuts.

This method, named in the remainder Continuous Maximum Flows (CMF) is essentially a convex reformulation of the classical Geodesic Active Contour (GAC) framework [60], which is widely used in 3D image segmentation. Being convex means, that the PDE can be solved by the broken-line algorithm and it provides a globally optimal solution with no metrication artifact. It is not a limitation on the form of the object which can be non-convex or non smooth. In addition, the method is efficient in 3D, and is therefore a good candidate for our purpose.

In the present work, we are applying the CMF method and both pre-, and post-processing image analysis methods to two different nano-material problems.

4.3 Electron Nano-Tomography

Electron nano-tomography originally proposed in the 1970s that uses a Transmission Electron Microscope (TEM) as an illuminating source for 3D object tomography, but has lately become more popular due to the increased availability of effective reconstruction software.

[4] PDE–Partial Differential Equation, are a type of differential equation, i.e., a relation involving an unknown function of several independent variables and their partial derivatives. Partial differential equations are used to formulate, and thus aid the solution of, problems involving functions of several variables.

Principles

While more standard tomography techniques use X-Rays as source, nano-tomography uses electrons instead. The main benefit of using electrons is the significant increase in resolution compared to X-Rays. As electrons in TEM behave in some ways as waves with a very high frequency, they allow for nanometer scale resolution, while X-Ray sources are typically limited to about micrometer scales resolution.

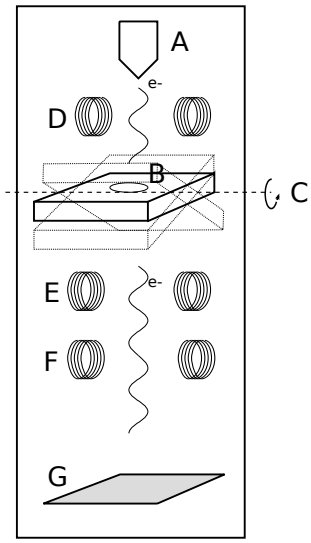


Fig. 4.3: Simplified schematics of a transmission electron microscope for nano-tomography. A: electron source, B: Sample, C: Tiltable stage, D: Condenser magnetic lens, E: Diffraction lens, F: Projection lens, G: Sensor, .

The general principles of electron nano-tomography (ET) are broadly similar to standard X-ray tomography [61], in the sense that projection images around the volume of interest are used, and the reconstruction of a 3D image of this volume is effected through the use of inverse tomography algorithms (for instance filtered back-projections, iterative methods, etc). Electrons traverse the material to be imaged, and are either left untouched, absorbed, diffracted or deflected as a result. The projection of the intensities as recorded by the TEM's imaging device is, under suitable conditions, comparable to an attenuation image, although many artifacts are typically present.

The sample, rather than the device, is rotated along one (or sometimes two axes), and projection images are recorded at regular angles. See Fig. 4.3 for simplified schematics. Similarly with CT scan, the angle(s) of tilt can be automatically associated with all recorded projection.

Features and Limitations of the Technique

One feature of TEM is that since electrons are negatively charged fermions, they readily interact with matter and are easily scattered by positively charged atom nuclei. As a result, unlike X-Rays, electrons cannot penetrate much into matter, and are affected by sample chemical contents: heavy atoms deviate electrons more than lighter ones. As a result, this so-called chemical (or Z-number) contrast is present in addition to absorption contrast. This effect can be used productively in Z-number contrast tomography [46]. The loss of energy in exiting electrons resulting from this interaction can also be used to derive chemical content in other modalities such as EFTEM [62].

However, due to this interaction with matter, preparation for TEM imaging implies thinning the sample under study as much as possible, so as to make it mostly transparent to electrons, using physical processes such as ion mills for instance. For some materials and high resolution needs, the final sample may be only a few hundreds of nanometer thick at its thinnest point. Because the final sample is then extremely fragile, it is currently impossible to obtain such thinness over a large area, and much less in such a way that the final sample is thin in two directions at once (like a thread or a needle). In other words, the final sample is most often like a thin layer in a relatively deep well, as illustrated in Fig. 4.4. In these planar objects the effective thickness T corresponds to their height H divided by the cosine of the tilt angle α. Therefore when $\alpha \to 90$, $H > 0$; $T = \frac{H}{\cos(\alpha)} \to \infty$. Even in naturally flat, thin samples, such as nanoparticles dispersed on a carbon grid, the grid holding the sample induces shadowing at high angles.

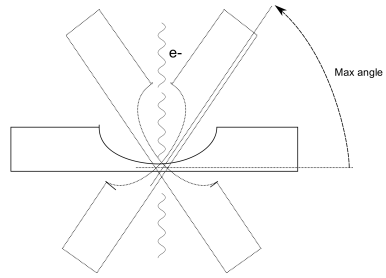


Fig. 4.4: As the sample is thin in one direction only, it is usually not practical to rotate it fully around the electron beam. The maximum tilt angle is around 70° or so.

As a direct consequence, the sample cannot be illuminated in all directions around it, unlike a patient in a CT scanner. This means that the electron beam cannot illuminate the sample in directions that are too far away from the normal to the thin layer. Finally, in a typical ET scanner the specimen holder sits inside the objective lens pole piece so there is very limited space for tilting, even if the sample would be string-like. Typically, illumination

much beyond 70° from the normal is difficult or impossible, and a complete set of projections cannot be obtained, from which to derive a complete tomographic reconstruction. To ameliorate the situation somewhat; tilt sequences can be recorded along several angles by rotating the grid on the horizontal plane. However *in fine* there is most often a so-called "missing wedge" in the projection space.

In practice, this translates into 3D images that exhibit noise and weak or elongated edges in the direction of the normal to the surface of the sample (i.e. the direction of the beam when the sample is untitled), but relatively strong features in the perpendicular directions to this normal. Fig. 4.5 shows some of these effects. This is an image of a polystyrene ball. Along the "equator" of the ball, edges are strong, but at the "poles", edges are weak.

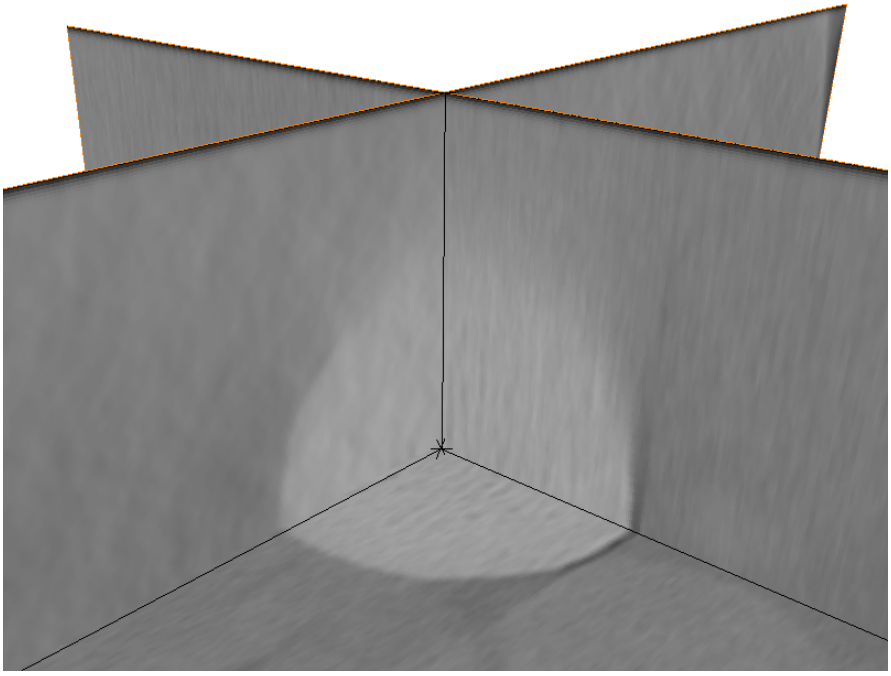


Fig. 4.5: This image represents some artifacts of the missing wedge effect commonly observed in electron nano-tomography. Because it is not possible to rotate the sample fully around the beam, edges perpendicular to the main beam direction are weak and elongated due to missing wedge effect, and the image is very noisy.

Image Analysis Challenges

Segmentation of nano-tomography images is difficult because of these characteristics, as we illustrate on Fig. 4.6. In particular, because of the missing wedge and strong noise, thresholding is unreliable (see Fig. 4.6(a)). For the same reasons, watershed [63, 64] is prone to leaking (see Fig. 4.6(b)). More recent methods, such as graph cuts [56, 65] are more successful but due to their anisotropic formulation, they tend to find edges that are aligned with the principal directions of the image sampling grid (see Fig. 4.6(c)), leading to clipped results. Here, the need for a segmentation solution which is little sensitive to noise, globally optimal and isotropic is essential. In the following section, we describe an improved solution.

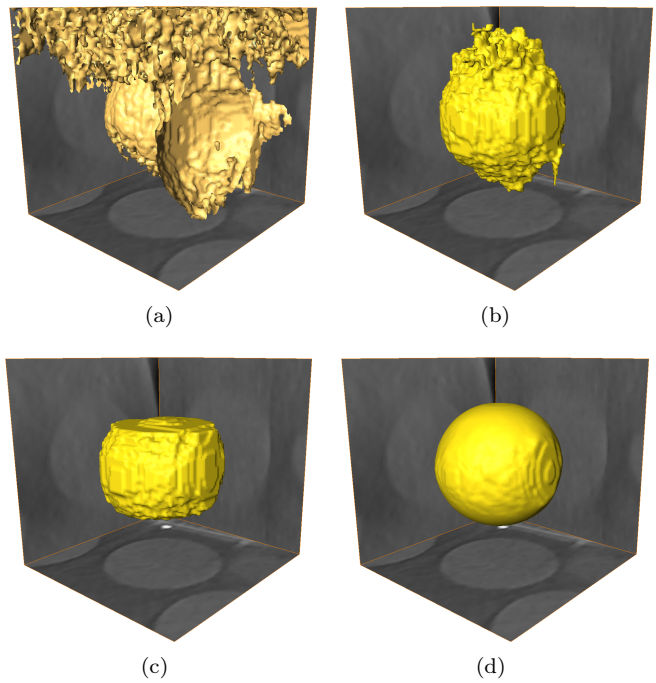


Fig. 4.6: Application of various 3D segmentation methods on nano-tomography images. (a) Optimal threshold ; (b) Watershed ; (c) Graph cuts ; (d) Continuous maximum flows. Only this last method provides a good result in our case.

4.4 Continuous Maximum Flows and Minimal Surfaces

In this section, we describe the continuous maximum flows model used for the segmentation of particles of nano-tomography images. We assume sufficient regularity for all functionals whose practice is usually met in physical systems.

As a fundamental idea, let g be a function on $\Omega \subset \mathbb{C}(\mathbb{R}^n)$ that defines a local metric cost, i.e. the value of g at point x is the cost of traversing point x. We will assume that g is scalar and provides values in $\mathbb{R}$, and we will call this function our *cost function*. Furthermore, let S (*source*) and P (*sink*) be two disjoint subsets of Ω. Let all closed simple hyper-surfaces[5] s with finite area (not necessarily connected) be those that do contain the source and do not contain the sink. In some sense, we can define the source of the segmentation as a marker for the interior of the object to be segmented, and the sink as a marker for its exterior.
Then we can define the following functional:

$$E(s) = \oint_s g ds, \tag{4.4.1}$$

called the total weight or total cost of s. As $E(s) \leq \text{area}(s) \cdot \max(g)$ is finite for bounded g. Furthermore, there exists at least one hyper-surface M exhibiting minimum weight [15]. In a discrete domain this minimum surface can be computed using the Ford and Fulkerson maximum flow graph algorithm [66], which was improved in the digital image context by [65].

In the continuous domain, M can be computed directly for every cost function g and sets P, T using for example active contours or surfaces methods [67], or level-sets methods [68, 69]. However, these methods compute surfaces iteratively via gradient descent schemes, and thus the solution is only locally optimal. Hence, the solution depends on initialization and noise levels. On the other hand, the algorithm presented by [10] provides a globally optimal solution to this problem. The solution is obtained in form of a smooth quasi-binary function[6] monotonically decreasing from source to sink. The iso-surfaces of this function will represent the minimum surface. As with the continuous nature of the algorithm, in some cases, it provides a sub-pixel accurate positioning of the minimal surface.

The differential geometric approach, that is to say, segmentation by optimization (Eq.4.4.1) is advantageous in cases where only parts of the contour are known. These are parts of the image where g is close to a constant. In

[5] A simple hyper-surface is a curve in 2D or a surface in 3D that does not intersect itself.
[6] One for which most values are either 0 or 1.

this case, this approach interpolates the known area with patches of minimum surfaces (in the geometric sense).

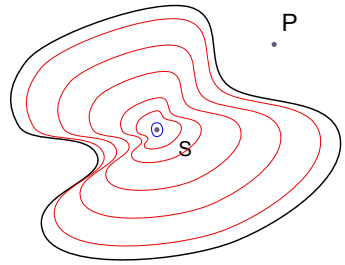


Fig. 4.7: We assume that our image is constant 1 everywhere except on the black curve, where its value ε is small. Now if we consider the integral on the black curve, than this integral will also be small. We specify pointwise source and sink S and P respectively. We would like to find the optimum curve, however as we take curves towards the source (the red curves), the integral tends to zero. A small curve around the source (blue) will have smaller integral than the black curve.

The solution with the maximum flows method depends on choices of: 1) a relevant cost-function g to our problem; 2) sources and 3) sinks. Examples of synthetic objects together with its robustness against noise can be found on Fig. 4.21. Here we consider the three basic shapes: Circle interpolation, convex and concave objects. In Fig. 4.21 (j)-(m) we use absolute-gaussian noise [70] generated by a quantum-random source [71] at rising levels. We can see, that the segmentation remains coherent even in relatively high noise situations.

4.4.1 The Choice of the Image Function

one that has the optimal integral) around the source of an object. However, on point-wise sources $E(s) = \oint_s \mathrm{d}G = \oint_s g\mathrm{d}x \leq \mathrm{surf}(s) \cdot \max_s g \to 0$ if we take smaller and smaller surfaces around S. So, that if we take a small surface around S, it will have small cost, but not on the contour of the object, like in Fig. 4.7. This can be overcome in two ways. The first possibility is that impose S to have some minimum length or surface. The boundary ∂S will then have non-zero cost, so that we can find the contours if they are smaller than the cost of the boundary. The second possibility is to modify our cost function with a particular weighting. For example, in a constant image L, lets define g as:

$$g_I(P) := \frac{1}{\mathrm{d}(P,S) \cdot (1 + \mathrm{d}L)} \tag{4.4.2}$$

Here, $d(P,S)$ denotes the distance of P and S.
Now let us look at the circles with ray r around the point-wise S:

$$\oint_{|\overline{\mathbf{x}}|=r} g_I(\overline{\mathbf{x}})\ \mathrm{d}\overline{\mathbf{x}} = \oint_0^{2\pi} \frac{r}{\underbrace{\mathrm{d}\left(\begin{bmatrix} r\sin(t) \\ r\cos(t) \end{bmatrix}, S\right)}_{r} \cdot (1+\mathrm{d}L)}\ \mathrm{d}t = \oint_0^{2\pi} \frac{\mathrm{d}t}{1+\mathrm{d}L} \tag{4.4.3}$$

If L is constant then Eq.4.4.3 gives 2π for all the circles around S. It can also be proven that these circles are the minimum cost curves of this measure. If $\mathrm{d}I$ is different from 0 at some point A (For example there is one point in I), then the "cheapest" curve will be the circle that contains A.

The following, complementary way to look at the problem is also true: if an image I is constant between the parts of the contour, then on these parts the solution will be interpolated with a circle. This approach is used in section 4.5.2.

We can extend this idea and define any set of curves D which do not intersect. It can be proven that there exists a weighting W, where the minimum cost curves are those of D.

This approach will tend to put the surface near the high frequency places before the curves of D. So it has a limited usage if the noise is at the level of the contours, like in Section 4.5.2, Fig. 4.9. Also there are cases where the set of curves is more difficult to define, like in section 4.5.3, Fig. 4.8.

The remaining challenges are how to choose sources and sinks. In the next few sections we present the ways we have solved this in the cases of weak gradient or/and high noise.

4.4.1.1 Calculation of the Cost Function

A relatively easy way to attract a minimal surface near object contours is to consider the following cost function, given by $g = \frac{1}{1+\|\nabla I\|}$. This cost function is high in relatively constant parts of the image, but drops to zero near edges, which are areas of high gradient. However, in regions where g is near-constant (in our images, this will occur near the extremities of objects under study that are along the electron beam, i.e. near the “poles” of objects), the minimum surface is a portion of plane. This means that the minimum surfaces will be attracted by the global noise.

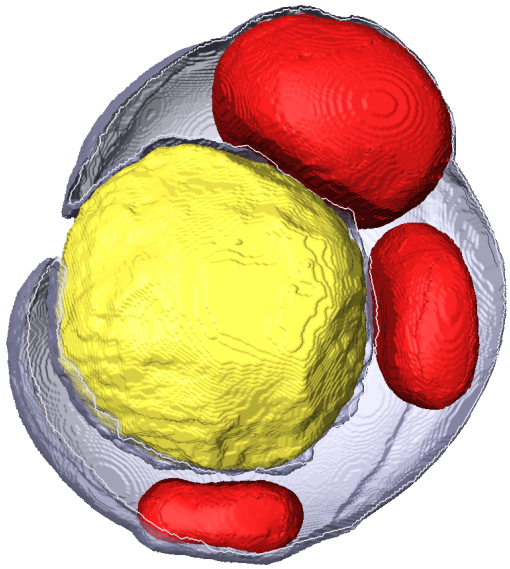


Fig. 4.8: Nanoparticle entering into the cell. The membrane has been cut in half for better visibility.

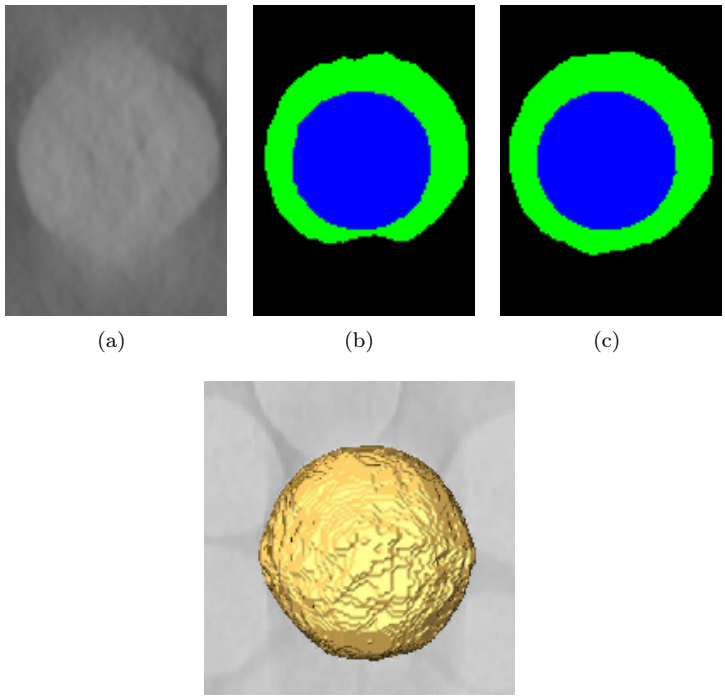
In this case the key observation is that the reconstruction-created noise shows parallelism with the axes. The result of this initial segmentation is close to a truncated sphere. To correct this defect, we used a *modified* g. More precisely, we used separable spline-interpolated gradient (available in [72]) in direction from the source to the point and the sphere weighting introduced in Section 4.4.1.

The formal definition follows: For a one-dimensional vector $\overline{\mathbf{x}}$, its continuous spline interpolation is denoted with $f_{\overline{\mathbf{x}}}$. For a 3D image I, the three axial vectors that contain the point P are denoted with $\overline{\mathbf{x}}_P$, $\overline{\mathbf{y}}_P$ and $\overline{\mathbf{z}}_P$, respectively. The gradient of the image I in point P is defined by $\mathrm{d}I_P = \begin{pmatrix} \partial f_{\overline{\mathbf{x}}_P} \\ \partial f_{\overline{\mathbf{y}}_P} \\ \partial f_{\overline{\mathbf{z}}_P} \end{pmatrix}$.

To calculate the directional gradient in point P we use the direction vector $\overline{\mathbf{c}} := \overline{\mathrm{CP}}$. The final cost function used in this case is

$$g(P) = \frac{1}{\underbrace{\mathrm{d}^2(P,S)}_{*} \cdot (1 + \underbrace{\overline{\mathbf{c}} \cdot \mathrm{d}I}_{**})} \tag{4.4.4}$$

Here, (**) is the gradient in direction of $\overline{\mathbf{c}}$ and (*) is the square of the distance from the source. The square exponent is needed because the area of the minimum surface is proportional to the square of the radius of the object. This cost function filters the artifacts in the direction z and the closer we are to the direction, the more we ignore the noise. An example of an image of a polystyrene ball segmented by the CMF method is shown in Fig. 4.9(a). Fig. 4.9(b) shows the ball segmented without the cost function weighting. We can observe an incorrect reconstruction due to weak gradient in this area of the image. In Fig. 4.9(c), 4.9(d) a better segmentation results are shown. It was achieved using the $\frac{1}{r}$ weighting described above.



(a) (b) (c)

(d)

Fig. 4.9: Slice of a segmented ball with the maximum flows method. (a) Original data. (b) Unweighted cost function segmentation result in green (in blue - marker). (c) Weighted cost function result (same color coding as in (b)). (d) 3D segmentation result.

4.4.1.2 Curvature Estimation

Curvature estimations were provided in some key areas of the image, for instance near points of contact between nanoparticles and liposomes as well as in the case of the membrane of the liposome. In the presence of a reliable

surface segmentation and in the continuous domain, the local curvature is well-defined mathematically and can be estimated using local second derivatives. Curvature estimation is also possible from implicit surface representations [73]. However, in our case we found that the precision of these methods was not good enough due to discretization and noise. A scale must be chosen at which to estimate the curvature and appropriate smoothing must be applied with some caution, in particular in order to preserve topology.

Instead, we developed semi-local representations of curvature appropriate to our problem, in particular given our priority regarding topology preservation. We started from the medial axis representation of our segmented result S [74] and found the extremities of this representation. The medial axis is the locus of the centers of maximal disks (2D) or balls (3D) included in S. Maximal balls are such that no ball can strictly contain them and still be included in S. Their center lie at local centers of symmetry for S, and they touch (and are in fact tangent to) the border of S on at least two distinct points. The superset of the medial axis that is connected and topologically equivalent to S is called the skeleton of S, and there exists efficient algorithms for computing the medial axis and the skeleton in 3D, see for instance [75, 76]. Skeleton extremities can be detected using local configurations (e.g. points with a pre-determined number of connected neighbors).

4.4.2 Source and Sink Determination

One of the keys to maximum flow segmentation is the choice of the source. In the model we assume three things: the source is inside the object, the sink is outside of it and we segment only one object at a time. While there exists situations where the choice of the source can be arbitrary placed within the object, in many cases, however, the segmentation can be improved with a good choice of source. While segmenting more objects at a time is theoretically possible, the topology of the final object is not guaranteed. There exist examples where more isolated sources still lead to a single object after the segmentation.

In our case, due to the missing wedge effect, a more precise choice of source and sink can considerably improve the segmentation results. In Section 4.5.2 we have used a preprocessing technique to determine the "centres" of the objects. From these carefully chosen sources we could apply a position-based noise filtering on the cost function g which led to our final segmentation.

In general, sinks are made of 3D bounding boxes around known objects of interest and do not present a strong challenge. In other cases, they might be derived from previous segmentations.

4.4.3 Calculation of the Optimal Surface

The continuous maximum flow system is described by the following system of equations:

$$\frac{\partial P}{\partial \tau} = -\mathrm{d} \cdot \overline{\mathbf{F}} \tag{4.4.5}$$

$$\frac{\partial \overline{\mathbf{F}}}{\partial \tau} = -\mathrm{d}P \tag{4.4.6}$$

$$|\overline{\mathbf{F}}| \leq g \tag{4.4.7}$$

Here P represents a scalar (pressure-like) field and F a flow vector field (a speed-like field). P is forced to 1 on the source and 0 on the sink. The equation can be solved numerically (by simulation). Assuming convergence for a given g, the steady-state solution is:

$$\mathrm{d} \cdot \overline{\mathbf{F}} = 0 \tag{4.4.8}$$

$$\mathrm{d}P = 0 \qquad \text{if} \quad |\overline{\mathbf{F}}| < g \tag{4.4.9}$$

$$\mathrm{d}P = -\lambda \overline{\mathbf{F}} \qquad \text{if} \quad |\overline{\mathbf{F}}| = g \tag{4.4.10}$$

The Eq. 4.4.8 simply restates the conservation of the flow. Eq. 4.4.9 applies if the flow has stabilized during the evolution without the constraint (Eq. 4.4.7). At stability, direction or magnitude of the flow vector field cannot change. From Eq. 4.4.6, 4.4.7 we can deduce that $\mathrm{d}P \cdot \overline{\mathbf{F}} \leq 0$, which means that P is a monotonically decreasing function along the flow lines. If $\overline{\mathbf{F}}$ is dense, as it is divergence-free, these flow lines can only initiate in the source and end at the sink.

Now, we define set $A = \{x | P(x) > p\}$ with $0 < p < 1$. On the iso-surface $Y := \partial A$ the $\mathrm{d}P \neq 0$ by construction, which means, that in these points (Eq. 4.4.7) applies thus:

$$\int_A \mathrm{d} \cdot \overline{F_Y} = \oint_Y \overline{N_Y} \cdot \overline{\mathbf{F}} \mathrm{d}Y = \oint_Y g \mathrm{d}Y = \oint_Y \mathrm{d}G \tag{4.4.11}$$

This implies, that every iso-surface of P is a minimal surface. If there is only one minimum, this also means that the P field can be $0 \leq P \leq 1$ only on a zero measure set.

Computation of minimal surfaces by this flow simulation is reasonably fast. For instance, in the case of section 4.5.2, steady-state convergence of a 116x116x116 pixel image is reached in 2000 iterations in 80 seconds on a dual-core AMD 2.5GHz Opteron CPU. Memory consumption is in the order of 4 times the initial image size in single-precision floating point format.

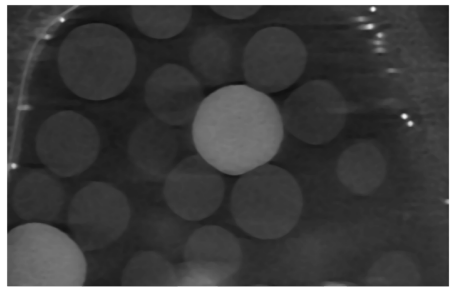
4.5 Segmentation and Interaction Analysis of Nano-Particles

4.5.1 Image acquisition

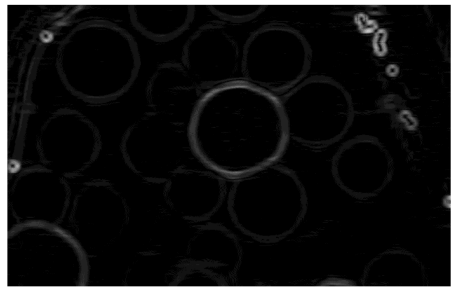
The volume images are reconstructed from a series of projected 2D images. The object is turned around its axe and full 2D attenuation pattern is recorded at each angle. In this application, a 5 μl sample was deposited onto a holey carbon coated copper grid, while the excess was blotted with a filter paper. The grid was plunged into liquid ethane, cooled with liquid nitrogen (Leica EM CPC). Specimens were maintained at a temperature of approximately -170 °C, using a cryo holder (Gatan). For the acquisition, the images were observed with a FEI Tecnai F20 electron microscope operating at 200 kV and at a nominal magnification of 50 000× under low-dose conditions. Images were recorded with a 2000×2000 slow scan CCD camera (Gatan). For cryo-electron tomography, tilt-series were collected automatically on both FEI Tecnai F20 and Tecnai G2 Polara from $-60°$ to $+60°$ with 2° angular increment using the FEI tomography software. Images were recorded on CCD camera at a defocus level between -8 μm and -4 μm. The pixel size at the specimen level varied between 0.5 nm and 0.36 nm. The sample was injected with high intensity particles before the recording, so the exact position of the carbon grid could been determined. For image processing, colloidal gold particles were used as fiducial markers. The 2D projection images were then binned by a factor of two and aligned with the IMOD software [77]. Finally the tomographic reconstructions were calculated by weighted back-projection using Priism/IVE package [78].

Fiducial markers used for image alignment can be seen as white spots in Fig. 4.22. However, as the carbon grid can only be turned around within about 120 degrees (due to the thickness of the grid), there are some parts of the object which are not present in the volume image. The result of the microscopy are three-dimensional images, however the signal-to-noise ratio becomes very low near object's poles due to missing wedge effect. An example can be observed in Fig. 4.5.

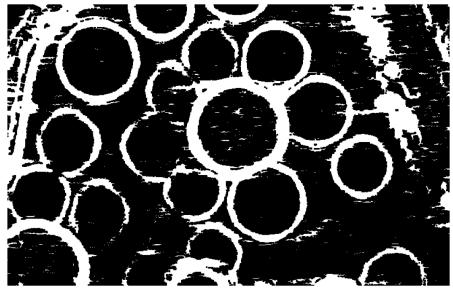
In this situation, it is desirable to present both the reliable segmentation, i.e. the part of the contour that was detected based on strong edge information, and the interpolated ones. In the following, we are detailing the filtering and the segmentation procedure with the described algorithm of the maximum flows and its optimization applied for two types of samples.



(a)



(b)



(c)

Fig. 4.10: The pre-processing steps: (a) Median filter. (b) First derivatives. (c) Morphological opening and closing results.

4.5.2 Polystyrene Beads Nucleated Around Silica Nanoparticles

The material consists of polystyrene nodules and silica bead embedded in a substrate. These roughly spherical objects of size range 100-300 nm are nucleated around an existing silica bead. One slice of a sample can be seen in Fig. 4.1. The work by [47] details all materials and methods.

The aim is to get an insight into the surface contact of polystyrene beads with silica nano-particles extracted from tomographic reconstructions. For this, the particles are automatically segmented and then their contact angles are also automatically measured.

In order to prove the usefulness of the max-flow method, we compare its performance to other widely used segmentation algorithms: simple threshold, watershed algorithm and the combinatorial graph cuts (that is the Ford-Fulkerson maximum flow algorithm). In the test images, we first applied a median filter, then we segmented the same filtered image by all the comparison algorithms and CMF. The final segmentation results are superimposed on the original image. This example shows the side-effects of these segmentation algorithms.

The Ford-Fulkerson, graph-cut and watershed may perform similarly on 2D images, whereas they have radically different performances when the problem becomes 3D on this data. The results of these four different methods can be seen in Fig. 4.6. In this case, the simple threshold of the image cannot produce any usable propositions Fig. 4.6 (d). We can see that the Ford-Fulkerson algorithm will converge on flat limiting planes instead of spherical ones and that watershed cannot find the limits of the objects at all, making thus the object reach the borders of the image. Considering these examples, the continuous maxflow algorithm equipped with the specialized gradient described earlier can be a good alternative over these methods. It can interpolate the missing (or weak) parts of the gradient with a simple form; in this case close to a sphere, but concave objects would also be possible. More complex objects, like facets of a crystal have not yet been tried, but would be an interesting problem to look at.

In order to segment the image with the maximum flows method from section 4.4, it is desirable to filter out the noise. As well, we need to specify a *source* and a *sink*.

As a preprocessing step, in order to reduce the noise in the original data (see Fig. 4.1) we used several filtering steps. 3D median filter was used in order to reduce speckle and salt-and-pepper noise. 2D edge extraction with the first derivative on a large scale in order to smooth out the noise. Afterwards, a 3D connectivity filter was used in order to eliminate smaller connected areas after thresholding [79]. A series of morphological openings and closings were useful in reconnecting and reconstructing the 2D circles. These filtering results can be observed in Fig. 4.10 .

Due to the missing wedge effect, on some slices, insufficiently reconstructed bead poles appear very dim, while well-reconstructed bead slices near the equator appear well separated from the background as in Fig. 4.5.

For this sample, in order to compensate for this drawback, we have used a preprocessing technique to determine the centres of the particles as closely as possible. The complete bead surface is interpolated by segmentation of its circles from the fully reconstructed (horizontal) image slices. The bead surfaces are used as *sources* for the maximum flow method segmentation.

In order to detect circles from 2D slices of the image volume, the Hough circle transform was used. The original Hough transform [80] and its derivatives have been largely applied and recognized as a robust technique [81] even in the presence of heavy noise. The circle Hough method did indeed succeed in localizing circle centres and radii even from incomplete initial circles (see Fig. 4.11).

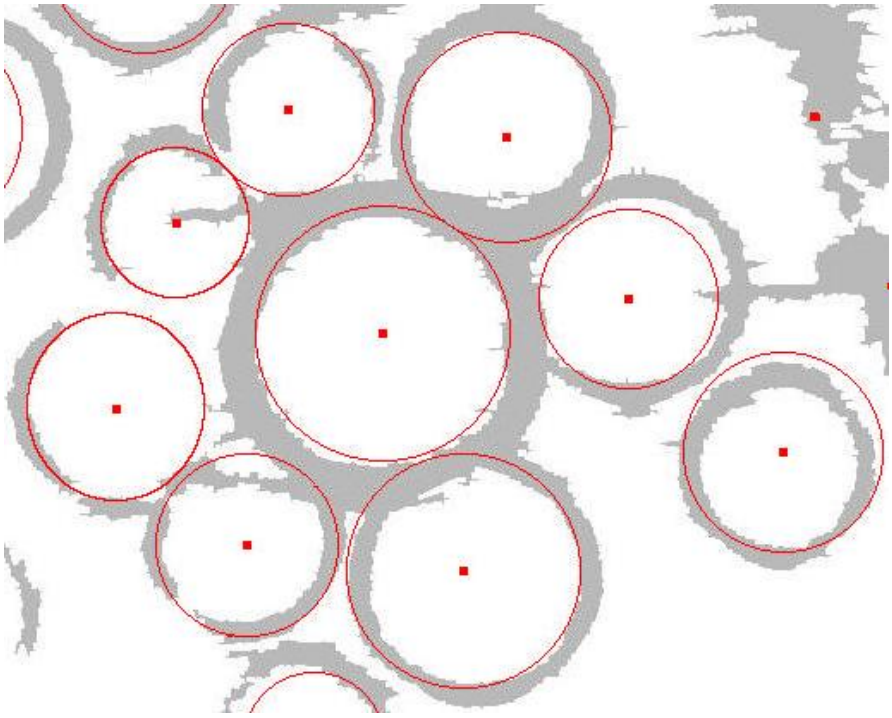


Fig. 4.11: A pre-processed image slice together with the Hough circle transform result.

Once the centres have been determined, these centres were used as sink for the maximum flow method. We have considered one bead at a time. In this case the extent of the beads is predictable. To facilitate our task without the loss of generality we could assume that a bead is fully contained within $0.7R$ to $1.2R$, where R is the estimated radius of the bead[7]. For the segmentation of each bead, as a *source* marker, a sphere with radius $0.7R$ centered in C was used. The complement of a sphere with radius $1.2R$ centered in C was used as the *sink* marker. This step accelerated the speed of the segmentation as the region excluded was omitted during the segmentation.

Due to the gradient's high sensitivity to noise, we used the gradient of a cubic spline [72] with some improvements: First the gradient is calculated

[7] the estimated radius is a byproduct of the Hough transform

on each line in each direction with an approximating spline. A spline is a function defined in our case on a line. At each integer point the value of the spline is the same as the image intensity and its derivatives up to degree 3 can be computed analytically.

Here the key observation was, that the noise is roughly parallel to the axes. We could therefore filter the noise from the gradient by calculating a **directional** gradient from the centre. This is done as follows: for a point A to which we would like to know the value of the directional gradient, we consider the ray[8] $\overrightarrow{\mathbf{OA}}$, where O is the centre of the bead. At point A we calculate the sum of the scalar products of the gradients in each direction $g_d = \overrightarrow{OA} \cdot \overrightarrow{g_x} + \overrightarrow{OA} \cdot \overrightarrow{g_y} + \overrightarrow{OA} \cdot \overrightarrow{g_z}$. The higher degree is used to avoid local fluctuations caused by the noise. Finally, the segmentation was performed using the continuous maximum flows method and the results can be observed in Fig. 4.12.

4.5.2.1 Contact Angle Measurement

The interest of the segmentation lies in the measurement of the attraction rate of the beads to the nanoparticle. The attraction is measured by the contact angle.

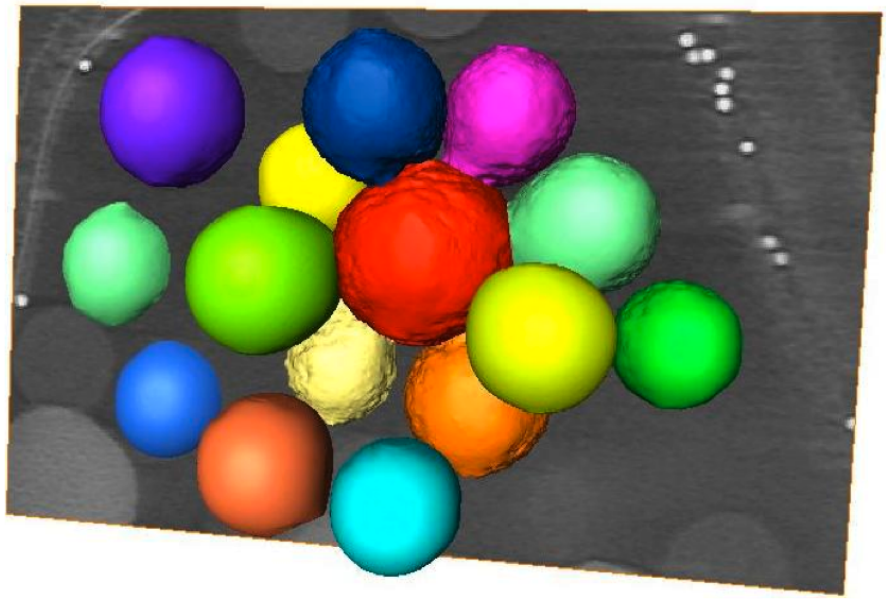


Fig. 4.12: Polystyrene and silica beads embedded in a substrate. The coloration is arbitrary.

[8] half-line

The common way of the angle calculation is Axisymmetric Drop Shape Analysis [82]. In this method a model of the bead is fitted to the image while minimizing the quadratic difference from the image. This way accurate angle measurements can be performed.

However, this method supposes that the physical model of the bead is known and precise. In our case, we do not make any assumptions about the physical properties of the material, so we perform direct angle measurements. In our case, the image plane is projected to the $\overline{\mathbf{x}}$ $\overline{\mathbf{y}}$ plane using an angle-preserving conformal map. Secondly, we find the contact point, and finally, we interpolate the contact arcs with a circle calculating the angle between the upper and lower interpolating circles.

Projection of the image

We consider the two center points of the beads A and B. We want to consider the planes containing the line $\overline{\mathbf{AB}}$. For the rotation, we use $\overline{\mathbf{x}}$ as the reference vector and $\overline{\mathbf{AB}} \times \overline{\mathbf{x}}$ as the third vector of the base. The basis $\underline{\underline{\mathbf{B}}}_1 = \left[\overline{\mathbf{x}}, \overline{\mathbf{x}} \times \overline{\mathbf{AB}}, \overline{\mathbf{AB}}\right]$ will be our reference basis.

As we want to calculate the angle for every possible cut, we rotate the basis B_1 around the axis $\overline{\mathbf{AB}}$. For this, we project the basis into the origin ($\underline{\underline{\mathbf{O}}}$) and then we apply the rotation transformation. Formally:

$$\underline{\underline{\mathbf{R}}} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

will be the rotation transformation and

$$\underline{\underline{\mathbf{O}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

will represent the origin basis. $\underline{\underline{\mathbf{T}}} = \underline{\underline{\mathbf{R}}} \cdot \underline{\underline{\mathbf{O}}}^{-1}$ will project the reference basis to the origo, with $\overline{\mathbf{z}}$ matching $\overline{\mathbf{AB}}$. From this $\underline{\underline{\mathbf{B}}}_2 = \left(\underline{\underline{\mathbf{T}}}^{-1} \cdot \underline{\underline{\mathbf{R}}} \cdot \underline{\underline{\mathbf{T}}}\right) \cdot \underline{\underline{\mathbf{B}}}_1$ will represent the projection basis which is the reference basis $\underline{\underline{\mathbf{B}}}_1$ rotated with angle α. For the angle measurement, we want to project the axe $\overline{\mathbf{AB}}$ to the axe $\overline{\mathbf{y}}$, so we swap the third and the second column of $\underline{\underline{\mathbf{B}}}_2$. We will mark this with $\underline{\underline{\mathbf{B}}}_2^{'}$. In the last step we calculate the projection matrix

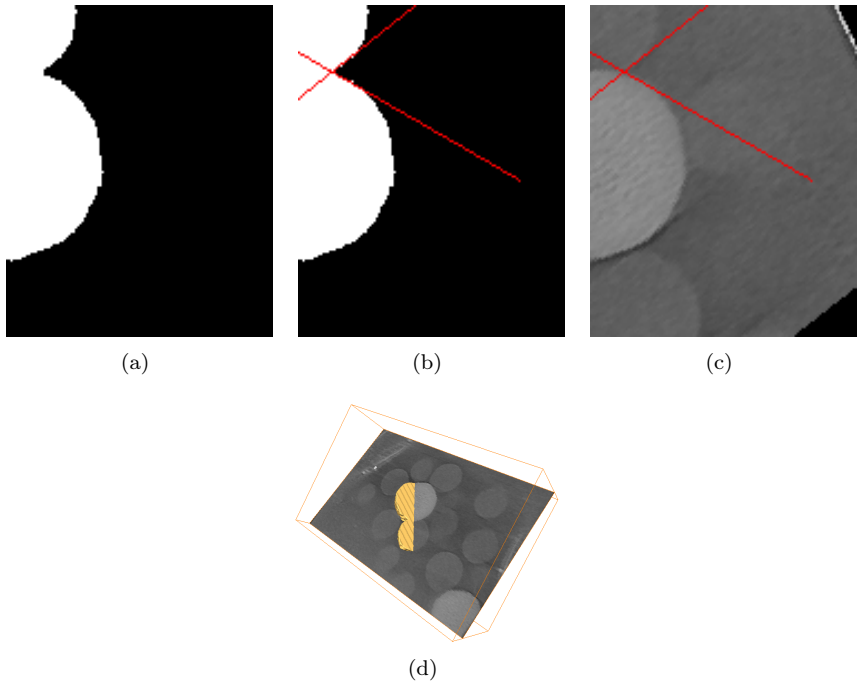
$$\underline{\underline{\underline{\underline{\mathbf{PR}}}}} = \underline{\underline{\mathbf{B}}}_2^{'} \cdot \underline{\underline{\mathbf{O}}}^{-1}$$

The $\underline{\underline{\mathbf{PR}}}$ matrix will project the points of the $\overline{\mathbf{x}}\ \overline{\mathbf{y}}$ plane to the plane around the axe $\overline{\mathbf{AB}}$. From this we calculate the actual intensity by nearest neighbour interpolation.

Isolation and interpolation of the contact arcs

As shown in Fig. 4.13, we can now conformingly map the cutting planes to 2D images. The neck found by a simple pass on the image looking for the closest point to the axis.

For the angle interpolation, we consider the radius r from the bottle neck point and we separate the upper and the lower arcs. The circles, which minimize the square error will represent the interpolation of the derivatives of the images.



(a) (b) (c)

(d)

Fig. 4.13: Contact angle measurement. The segmented object (a), the estimated angle (b), the angle superimposed to the object (c) and the cut plane in the 3D image (d)

Finding the best fitting circle

For the best circle fit we use a modified version of the ellipse fitting algorithm from [83]. The description of the algorithm follows:

Let's assume, that our circle is defined by the equation:

$$a\left(x^2+y^2\right)+bx+cy+d=0, \tag{4.5.1}$$

where $\overline{\mathbf{a}}=\{a,b,c,d\}^T$ and $\overline{\mathbf{x}}=\left\{x^2+y^2,x,y,1\right\}^T$.

To find the fitting circle, we minimize the algebraic distance:

$$\text{dist}(\overline{\mathbf{a}})=\sum_{i=1}^{N}\left(\overline{\mathbf{a}}^T\cdot\overline{\mathbf{x}}\right)^2 \tag{4.5.2}$$

If we reformulate Eq. 4.5.1 to the conventional form:

$$\left(x+\frac{b}{2a}\right)^2+\left(y+\frac{c}{2a}\right)^2+\left(d-\frac{b^2}{4a^2}-\frac{c^2}{4a^2}\right)=0, \tag{4.5.3}$$

from Eq. 4.5.3 we can see that the condition for Eq.4.5.1 being a circle is:

$$d-\frac{b^2}{4a^2}-\frac{c^2}{4a^2}<0 \tag{4.5.4}$$

$$0<4a^2d-b^2-c^2 \tag{4.5.5}$$

As the circle equation is overdetermined (namely the a constant), we can impose Eq. 4.5.4 as constraint $4a^2d-b^2-c^2=1$. With these considerations we can reformulate the problem as a Lagrange minimization:

$$\min_{\overline{\mathbf{a}}}||\underline{\underline{\mathbf{D}}}\cdot\overline{\mathbf{a}}||^2 \qquad \text{s.t.} \qquad \overline{\mathbf{a}}^T\cdot\underline{\underline{\mathbf{C}}}\cdot\overline{\mathbf{a}}=1 \tag{4.5.6}$$

Here, $\underline{\underline{\mathbf{D}}}$ denotes the *design matrix* of size $N\times 4$:

$$\underline{\underline{\mathbf{D}}}=\begin{bmatrix} x_1^2+y_1^2 & x_1 & y_1 & 1\\ x_2^2+y_2^2 & x_2 & y_2 & 1\\ \vdots & \vdots & \vdots & \vdots\\ x_n^2+y_n^2 & x_n & y_n & 1\end{bmatrix}$$

and $\underline{\underline{\mathbf{C}}}$ denotes the *constraint matrix*:

$$\underline{\underline{\mathbf{C}}}=\begin{bmatrix} 0 & 0 & -2 & 0\\ 0 & 1 & 0 & 0\\ -2 & 0 & 1 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$$

Following the argumentation of [83], the Lagrange system can be rewritten as

$$\underline{\underline{\mathbf{S}}} \cdot \overline{\mathbf{a}} = \lambda \underline{\underline{\mathbf{C}}} \cdot \overline{\mathbf{a}} \tag{4.5.7}$$

$$\overline{\mathbf{a}}^T \cdot \underline{\underline{\mathbf{C}}} \cdot \overline{\mathbf{a}} = 1 \tag{4.5.8}$$

where $\underline{\underline{\mathbf{S}}}$ is the *scatter matrix*, $S = \underline{\underline{\mathbf{D}}}^T \cdot \underline{\underline{\mathbf{D}}}$. This system is readily solved by considering the generalized eigenvectors of (4.5.7). If $(\lambda_i, \overline{\mathbf{u}}_i)$ solves (4.5.7), then so does $(\lambda_i, \mu\overline{\mathbf{u}}_i)$. Giving

$$\mu_i = \sqrt{\frac{1}{\overline{\mathbf{u}}_i^T \cdot \underline{\underline{\mathbf{C}}} \cdot \overline{\mathbf{u}}_i}} = \sqrt{\frac{1}{\overline{\mathbf{u}}_i^T \cdot \underline{\underline{\mathbf{S}}} \cdot \overline{\mathbf{u}}_i}} \tag{4.5.9}$$

and setting $\overline{\mathbf{a}}_i = \mu_i \overline{\mathbf{u}}_i$ solves Eq.4.5.8.

The solution of the eigensystem Eq. 4.5.7 gives four results. These four results are all local minima of the equation, so selecting the vector which minimizes Eq. 4.5.2 yields be the optimal vector.

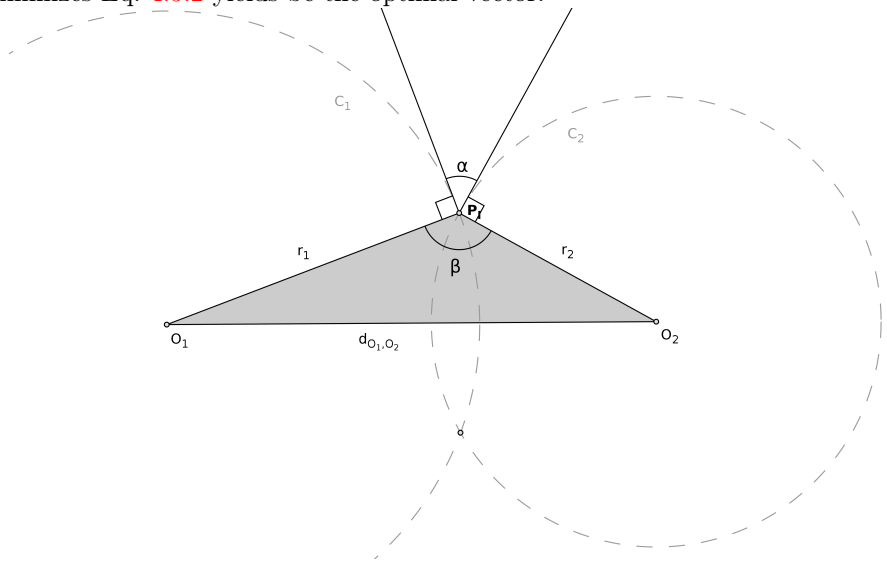


Fig. 4.14: Contact angle measurement of two artificial spheres C_1 and C_2. In this image, the desired angle is α at the contact point P_1. r_1 and r_2 represent the radiuses of the beads and d_{O_1,O_2} represents the distance of the centers of the beads. As in this example we model the beads by spheres, the volumic (mass) and the geometric centers coincide.

Verification of the method

We will demonstrate the correctness of the algorithm by statistical measurements. We model the beads by two spheres which intersect each other. The contact angle will be the angle measured at the contact ring. This angle can be calculated from geometrical considerations as seen in Fig. 4.14. We can see that angle α is the complementer angle of β, whereas β can be calculated from the area of the triangle $O_1O_2P_I$:

$$S = \frac{1}{2} r_1 r_2 \sin\beta \tag{4.5.10}$$

$$S = \frac{1}{4}\sqrt{\left(r_1^2 + r_2^2 + d_{O_1,O_2}^2\right)^2 - 2\left(r_1^4 + r_2^4 + d_{O_1,O_2}^4\right)} \tag{4.5.11}$$

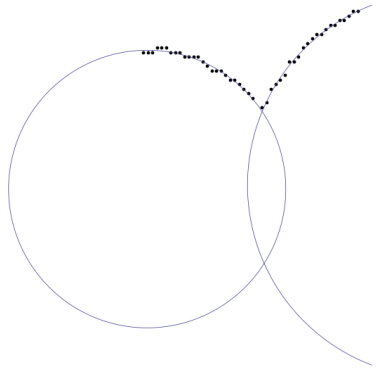
Here, Eq. 4.5.10 is the SAS[9] theorem, while Eq. 4.5.11 denotes Heron's formula. Final formula:

$$\alpha = \pi - \operatorname{asin}\left(\frac{\sqrt{\left(r_1^2 + r_2^2 + d_{O_1,O_2}^2\right)^2 - 2\left(r_1^4 + r_2^4 + d_{O_1,O_2}^4\right)}}{2r_1 r_2}\right) \tag{4.5.12}$$

gives us the reference angle. In the test we generate two randomly posed intersecting spheres and follow the procedure described above. The measured angles are then averaged and compared with the artificial angle estimation.

Results

During the tests we have measured more than 600 pairs of random circles and measured a mean absolute difference of 3.3% compared to the artificial estimations. An example of the interpolated image can be seen in Fig. 4.15.



[9] side-angle-side

Fig. 4.15: Circe fitting on two arcs

The contact angle provides an important information concerning the rate of attraction of the particles. While the contact angle can be measured by hand, the automated measurement can provide more consistent and objective information about the chemical forces inside the substrate. An example of the superposed angles can be seen on Fig. 4.13. In this image, the angle is calculated at several different angles. From these we calculate the average angle. Higher angles correspond to higher attraction forces.

4.5.3 Nanoparticle Transport Across Phospholipid Membrane

For this application, we focus on the interaction of silica nano-particles with large unilamellar liposomes (see Fig. 4.2 for an image slice example).

While many past studies focused on measuring the end-point nanomaterials and the distribution of their particles, relatively few studies have been dedicated to the understanding of molecular interactions between nanomaterials and cell membrane, which may provide the necessary information to understand how nanomaterials bind and enter cells [84].

Nano-particle transport across cell membrane is important in the development of drug delivery systems, as well as in the question of nano-particle poisoning. We know that hydrophilic nano-particles interact with the lipid membranes. However, if they succeed to enter into the cell and to which extent, we do not know. Several models have been proposed based from the membrane curvature to even the complete form of the particle.

It was generally believed, that the particles did not enter into mammalian cell by endocytosis[10]. As evidence [85] and [86] argumented with the entry of ultra fine particles into the red blood cells and cyt-D blocked macrophages[11]. Both of these cells are known for their lack of endocytotic capabilities. However, [84] revealed[12] that in some cases the molecules did not pass through the membranes as expected. This suggests that the nano-particle transport requires an interaction with the membrane. Unlike the nano-particles larger than 30 nm, these 20 nm particles could not “break into” the membrane.

[10] Endocytosis is a process where cells absorb material (such as nanoparticles) from the outside by engulfing (wrapping around) it with their cell membrane.

[11] White blood cells that absorb material foreign to the body (bacteria, etc).

[12] In a study made with gold molecules and a liposomes that mimics the biological membrane.

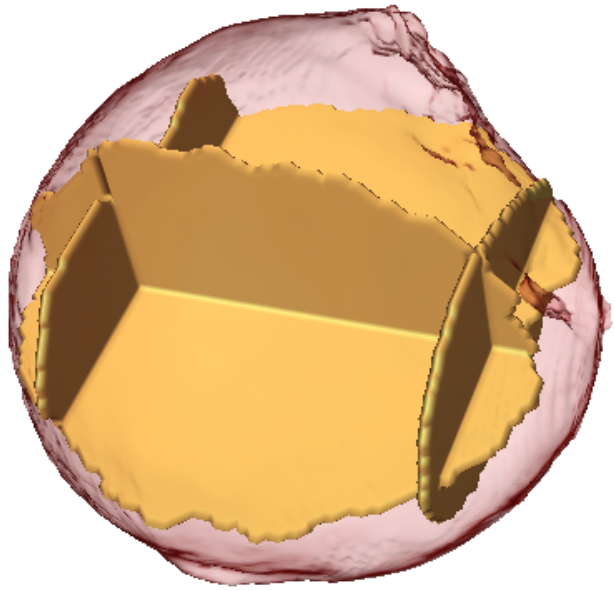


Fig. 4.16: The inner yellow object is the source that have been generated by segmenting the 2D slices with good visibility.

The results of the study (to which this paper has partially contributed) provided in [43], indicate that silica particles, which are bigger than 30 nm can enter into the liposomes composed of phosphocholine lipid, while smaller particles cannot. This is due to the favorable balance between the adhesion strength and membrane curvature. Smaller particles will not be able to enter because of the less favorable balance.

In this segmentation in addition to the above presented techniques, we used a special method for determining the source. Because of the asymmetric shape of the object, a simple constraint bias was not sufficient. We have extracted instead several 2D slices from the image, which we have segmented with the same method, then we have created a complex 3D source from the result. The created source is presented in Fig. 4.16.

Our segmentation results are summarized in Figures 4.8, 4.17, and 4.19. These show a slice of the input image together with the borders of the segmented objects superimposed in white. We used this evidence to visually check the correctness of the segmentation and the estimation of curvature, which we measured in places of interest (Fig. 4.20 and Fig. 4.18).

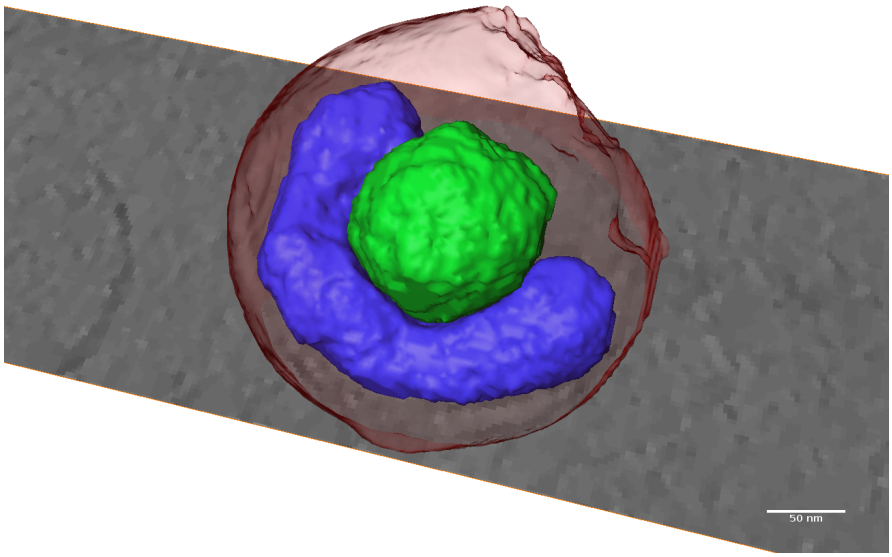


Fig. 4.17: Nanoparticle penetrated into the cell. The inner membrane wraps around the particle.

4.6 Discussion and Conclusion

In this article, we have demonstrated the usefulness of the continuous maximum flow framework for the segmentation of electron nano-tomography images. Despite the presence of noise, lack of contrast and low resolution of images, the method was extended to provide for the interpolation of missing parts of data as well as to cope for the structural noise. Its algorithmic design allows high level of parallelization which makes it suitable for high resolution images. Moreover, a free implementation exists [**?**], which makes it suitable for research and a useful option for inclusion into other image processing frameworks.

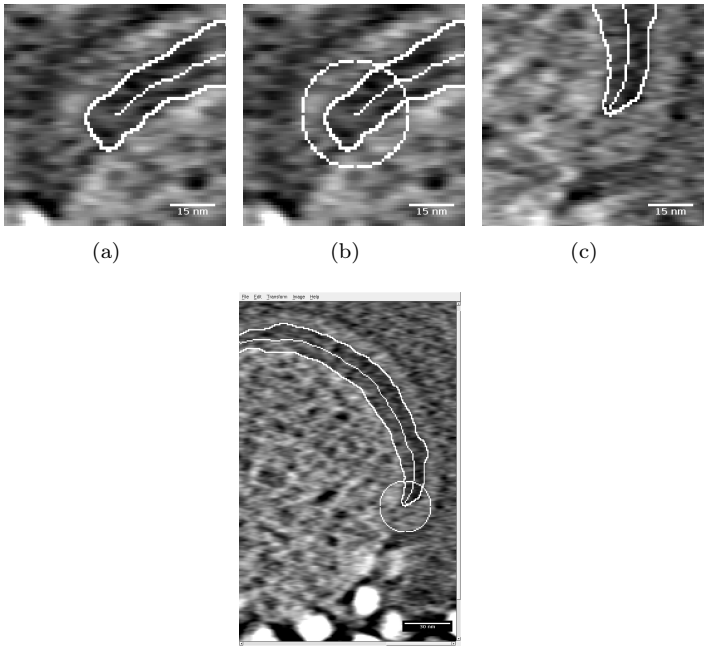
The method performs reasonably fast in all the above applications.

The method was further improved by adding shape constraints, optimizing its performance for the shape-corrupted objects due to the missing wedge effect inherent to the image modality.

The filtering and the segmentation procedure with the maximal flows algorithm and its optimization were applied for two types of nano-material samples.

In the first example, the aim was to find the size of some polystyrene beads and location with respect to a silica bead. The presented method was compared with other classical segmentation methods: thresholding, watershed and graph-cut, and shown to present significantly better performance.

Moreover, we have presented an automatic contact angle measurement algorithm and its statistical evaluation on simulated data. Such automated measurement can provide more consistent and objective information about the chemical forces inside the substrate.



(a) (b) (c)

(d)

Fig. 4.18: The estimated curvatures.

In the second image sample, the max-flow technique was shown to be useful in understanding of molecular interactions between nano-materials and cell membrane, which may provide the necessary information in the understanding of binding and entering of silica nano-particles and large unilamellar liposomes.

4.7 Future Work and Objects of Different Shape

As future work, the presented methods will be tested on larger data sets and validated by specialists. In the case of high volume datasets, or many images to segment, the algorithm is implementable on GPU architecture.

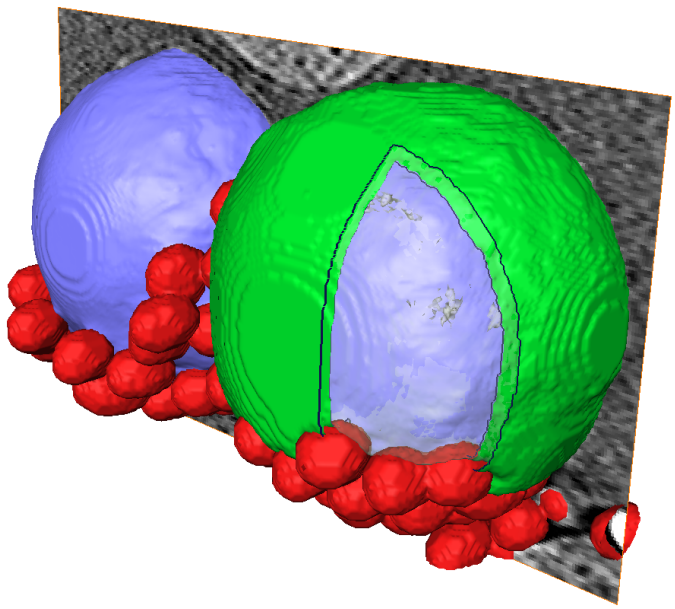


Fig. 4.19: Interaction of the lipidic membrane with the nanoparticle. The red pieces are the gold markers that were used for the reconstruction.

A GPU implementation would improve the segmentation speed by an order of magnitude.

In the current state of the algorithm, as demonstrated, results are good on spherical or elliptical objects both in 2D and in 3D. These types of objects were indeed those which we have been working on. In the case of long fiber-like, tube-like[13] or generally more complex objects, the complete reconstruction might of course be more difficult, as in such problems shape constraints can be difficult to express. There is, however, no theoretical difficulty in applying the method on any type of objects. Furthermore, we would be interested in looking at other problems as well, should we be presented with them.

[13] like nanotubes

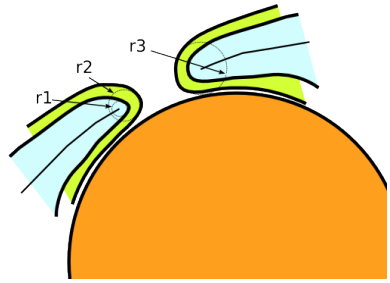
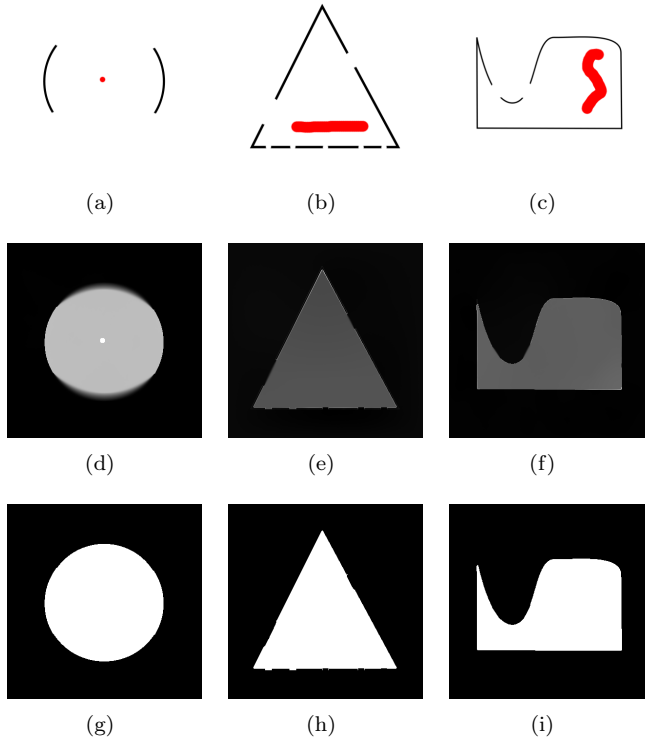
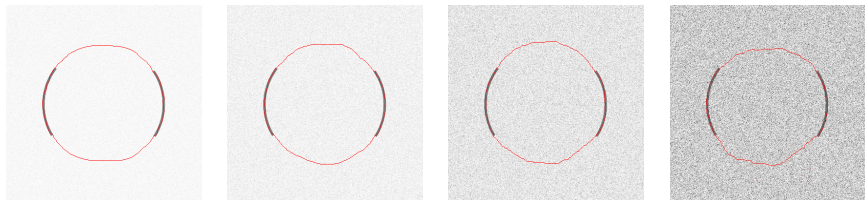


Fig. 4.20: Curvature estimation. R1 is the radius of the inner maximal ball, R1 the radius of the external maximal ball and R3 the radius to the nano-particle. Depending on the configuration either R2 or R3 yield a robust curvature estimation.

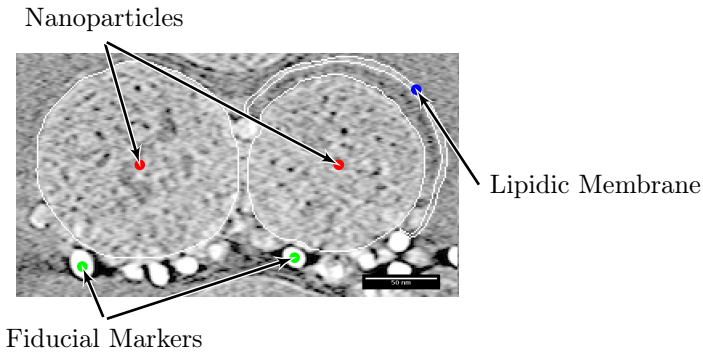
Concerning the improvement of the algorithm, the basic tools have been collected for extending the algorithm to optimize the flow according to an arbitrary convex-set-function. In this work, mostly spherical constraints were imposed. Instead of $|\overline{\mathbf{F}}| \leq g$ we could simply enforce $\overline{\mathbf{F}}(x) \in \Gamma(x)$ where $\Gamma(x)$ is a convex set defined in each point. This would make it possible to optimize the flow for more complex family of curves.



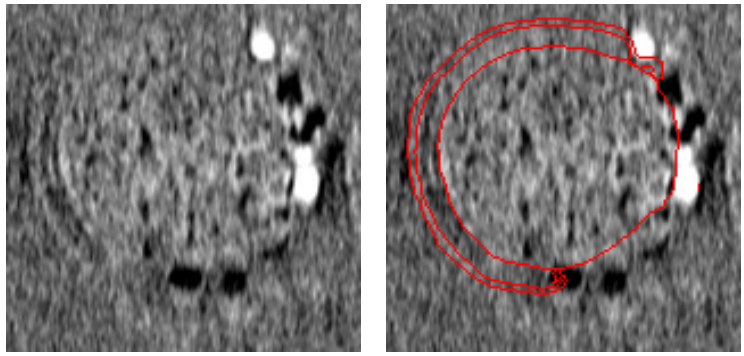


(j) (k) (l) (m)

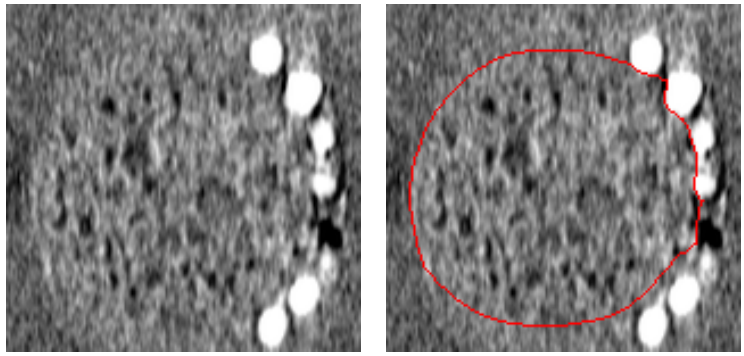
Fig. 4.21: Examples of continuous maximum flow segmentation. The sources are marked in red. The sinks are the border of the images. In the image (a), the constraint-field is affected with a circular bias, whereas the images (b) and (c) are not altered. We can see, that the segmentation succeeds even on incomplete or concave objects. Images (d), (e) and (f) represent the partially converged pressure fields, while images (g), (h) and (i) are the final segmentations. Images (j)-(m) show the CMF's resistance against noise. Noise levels are $\sigma = 8$ (j), $\sigma = 16$ (k), $\sigma = 32$ and (l) $\sigma = 64$ (m) on a 256 grey-level image. Note, that images have not been prefiltered before segmentation.



(a)



(b) (c)



(d) (e)

Fig. 4.22: Summary of segmentation procedure results: segmentation accuracy of nanoparticle. Results superimposed in white over the original image data. Slice (a) is that of the best original image quality. (b)-(c) and (d)-(e) are plane cuts in the perpendicular direction of (a). Here the week parts of the image were interpolated with the CMF algorithm.

4.8 Conclusions and Future Work

During this thesis we have presented our work which have been carried out in the field of image segmentation by continuous optimization. In particular we have analyzed in detail the continuous maximum flow algorithm. The continuous maximum flow algorithm is a segmentation algorithm based on minimum surfaces. It provides the segmentation in form of image partitions, where each partition is wrapped around by a minimum surface. While mainly used for image segmentation, the algorithm can also be used for image denoising.

We have analyzed in detail the implementation of the algorithm with a detailed review of the current available architectures. We have adapted the algorithm to run in parallel on many architectures and reformulated the update scheme to optimize it on some embedded GPGPU architectures.

From the application point of view, we have work on segmentation of several nanotomographic images of materials, nanoparticles and, lipidic membranes as well as some applications in medical images. During the work we have proposed new preprocessing methods, measure calculations as well as new source and sink determinations.

From the developer point of view we have included our work in an image processing frameworks, which is cross platform, actively developed and which provides a fertile ground on which new image processing applications can grow in the future.

Appendix

Chapter 5
The Pink Image Processing Library

Pink is an image processing and image analysis library developed at ESIEE Engineering for research and teaching purposes. It is build upon Boost Python and the Python Imaging Library, and uses plug-ins that use python-vtk, numpy and matplotlib. While the core foundation has been in development for over a decade, its Python front-end is recent work.

Most existing image processing libraries concentrate on pixel-based operators and linear filtering methods like convolutions, FIR and IIR filters, diffusion and so on. Pink complements these with implementations of over 200 algorithms for image segmentation and non-linear filtering. Whereas most operators come from mathematical morphology, discrete geometry and discrete topology, operators from other fields are implemented as well.

The Pink dedicated front end is exposed using Boost Python. The library is designed to behave natively in Python, with the functional programming paradigm. Operators are functions and the images are returned after the operation as results. This way complex image processing pipelines and algorithms can be easily scripted in Python. This enables an easy plugin creation or development of solutions for particular applications (preprocessing—segmentation—post-processing).

The Python front-end makes it suitable to use Pink together with other packages like SAGE, PyLab or Tkinter. This makes it possible to develop image segmentation and processing methods that are using graphical interfaces for parametrization. The operators can be demonstrated using Python's facilities in an interactive manner. This functionality is particularly useful as a learning tool.

All the operators feature research-level, scientific journal-described algorithms with 3D extension where possible. Notable operators include skeletonization and topological thinning, mathematical morphology, watershed, maximum flow, total variation filtering, and more.

From the technical point of view, the images in Pink are exposed in Python as objects. The operators are functions, with images as first parameters. As result, they return a processed image. Pixels, if necessary, can be accessed

with intuitive operators. This means that simple algorithms for experimenting can be implemented directly in Python with speed as the only constraint.

By design new C/C++ operators can be exposed using only a few lines of C++ code, completely omitting third party languages. Indeed, the complete library is implemented using C/C++ and Python. It has been ported to most of the popular Linux distributions and it also runs on OSX and Microsoft Windows.

The software is open-source and freely available at http://pinkhq.com. It is currently soliciting new users and developers.

References

1. E. N. Malamas, E. G. M. Petrakis, M. Zervakis, L. Petit, and J.-D. Legat, "A survey on industrial vision systems, applications and tools," *Image and Vision Computing*, vol. 21, no. 2, pp. 171 – 188, 2003.
2. T. Brosnan and D.-W. Sun, "Improving quality inspection of food products by computer vision–a review," *Journal of Food Engineering*, vol. 61, no. 1, pp. 3 – 16, 2004. Applications of computer vision in the food industry.
3. J. S. Kim, Y. T. Son, H. S. Cho, and K. I. Koh, "A robust visual seam tracking system for robotic arc welding," *Mechatronics*, vol. 6, no. 2, pp. 141 – 163, 1996.
4. M. B. McDonald, "Seed quality assessment," *Seed Science Research*, vol. 8, no. 02, pp. 265–276, 1998.
5. C. J. Vyborny and M. L. Giger, "Computer vision and artificial intelligence in mammography.," *Ajr American Journal Of Roentgenology*, vol. 162, no. 3, pp. 699–708, 1994.
6. L. B. Dorini, R. Minetto, and N. J. Leite, "White blood cell segmentation using morphological operators and scale-space analysis," in *Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing*, (Washington, DC, USA), pp. 294–304, IEEE Computer Society, 2007.
7. M. A. S. Cordeiro, A. C. Lardo, M. S. V. Brito, M. A. Rosario Neto, M. H. A. Siqueira, J. R. Parga, L. F. Avila, J. A. F. Ramires, J. A. C. Lima, and C. E. Rochitte, "Ct angiography in highly calcified arteries: 2d manual vs. modified automated 3d approach to identify coronary stenoses.," *Int J Cardiovasc Imaging*, vol. 22, no. 3-4, pp. 507–16.
8. T. Otto, "Computer-aided direct all-ceramic crowns: preliminary 1-year results of a prospective clinical study.," *Int J Periodontics Restorative Dent*, vol. 24, no. 5, pp. 446–55, 2004.
9. M. Couprie and H. Talbot, "Distance, granulometry, skeleton," in *Mathematical morphology: from theory to applications*, ch. 10, pp. 291–316, Wiley, 2011. To appear.
10. B. Appleton and H. Talbot, "Globally minimal surfaces by continuous maximal flows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 1, pp. 106–118, 2006.
11. D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 207, no. 1167, p. 187, 1980.
12. N. Sochen, R. Kimmel, and R. Malladi, "A general framework for low level vision," *IEEE Trans. Image Processing*, vol. 7, pp. 310–318, 1999.
13. L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
14. D. Kirsanov and S. J. Gortler, "A discrete global minimization algorithm for continuous variational problems. harvard computer science technical report: Tr-14-04," tech. rep., Harvard University, Cambridge, MA, 07/2004 2004.
15. G. Strang, "Maximal flow through a domain," *Mathematical Programming*, vol. 26, pp. 123–143, 1983.
16. R. Nozawa, "Max-flow min-cut theorem in an anisotropic network," *Osaka Journal of Mathematics*, vol. Volume 27, no. Number 4, pp. 805–842., 1990.
17. B. Appleton and H. Talbot, "Globally optimal geodesic active contours," *J. Math. Imaging Vis.*, vol. 23, pp. 67–86, July 2005.
18. D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Communications on Pure and Applied Mathematics*, vol. 42, no. 5, pp. 577–685, 1989.
19. L. Evans and R. Gariepy, *Measure theory and fine properties of functions*. CRC, 1992.
20. L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D*, vol. 60, pp. 259–268, November 1992.
21. J. Douglas and H. H. Rachford, "On the numerical solution of heat conduction problems in two and three space variables," *Trans. Amer. Math. Soc.*, no. 82, pp. 421–439, 1956.

22. Antonin and Chambolle, "An approximation result for special functions with bounded deformation," *Journal de Mathématiques Pures et Appliquées*, vol. 83, no. 7, pp. 929 – 954, 2004.
23. P. L. Combettes and J.-C. Pesquet, "A proximal decomposition method for solving convex variational inverse problems," *Inverse Problems*, vol. 24, no. 6, p. 065014, 2008.
24. P. L. Combettes and J.-C. Pesquet, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, vol. 48, ch. Proximal splitting methods in signal processing, pp. 185–212. New York: Springer, 1 ed., 2011.
25. A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imaging Vis.*, vol. 20, no. 1-2, pp. 89–97, 2004.
26. H. B. Markus Unger, Thomas Pock, "Interactive globally optimal image segmentation," Technical Report ICG–TR–08/02, Institute for Computer Graphics and Vision Graz University of Technology, Austria, July 2008. contact: Markus Unger "unger@icg.tugraz.at".
27. T. F. Chan, G. H. Golub, and P. Mulet, "A nonlinear primal-dual method for total variation-based image restoration," *SIAM Journal on Scientific Computing*, vol. 20, no. 6, pp. 1964–1977, 1999.
28. M. J. Flynn and P. Hung, "Microprocessor design issues: Thoughts on the road ahead," *IEEE Micro*, vol. 25, pp. 16–31, May 2005.
29. Ecole Normale Supérieure, *The Pink Image Processing Library*, (Paris), ESIEE Engineering, 2011-08-27. http://www.pinkhq.com.
30. L. Marak, "uifibres optimization," tech. rep., ESIEE Engineering, december 2007.
31. A. Peleg and U. Weiser, "Mmx technology extension to the intel architecture," *IEEE Micro*, vol. 16, pp. 42–50, August 1996.
32. C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the last line of defense before hitting the memory wall for cmps," in *High Performance Computer Architecture, 2004. HPCA-10. Proceedings. 10th International Symposium on*, pp. 176–185, IEEE, 2004.
33. M. Gomaa, M. Powell, and T. Vijaykumar, "Heat-and-run: leveraging smt and cmp to manage power density through the operating system," in *ACM SIGARCH Computer Architecture News*, vol. 32, pp. 260–270, ACM, 2004.
34. L. Marak, J. Cousty, L. Najman, and H. Talbot, "4d morphological segmentation and the miccai lv-segmentation grand challenge," 07 2009.
35. R. Fisker, J. Carstensen, M. Hansen, F. Bodker, and S. Morup, "Estimation of nanoparticle size distributions by image analysis," *Journal of Nanoparticle Research*, vol. 2, pp. 267–277, September 2000.
36. G. Woehrle, J. Hutchison, S. Ozkar, and R. Finke, "Analysis of Nanoparticle Transmission Electron Microscopy Data Using a Public-Domain Image-Processing Program, Image," *Turkish Journal of Chemistry*, vol. 30, no. 1, p. 1, 2006.
37. J. Fendler, *Nanoparticles and nanostructured films*. Wiley-VCH Weinheim, 1998.
38. G. Schmid, *Nanoparticles: from theory to application*. Wiley-VCH, 2004.
39. M. Valden, X. Lai, and D. W. Goodman, "Onset of Catalytic Activity of Gold Clusters on Titania with the Appearance of Nonmetallic Properties," *Science*, vol. 281, no. 5383, pp. 1647–1650, 1998.
40. N. Rosi and C. Mirkin, "Nanostructures in biodiagnostics," *Chem. Rev*, vol. 105, no. 4, pp. 1547–1562, 2005.
41. W. C. Chan, "Bionanotechnology progress and advances," *Biology of Blood and Marrow Transplantation*, vol. 12, no. 1, Supplement 1, pp. 87 – 91, 2006.
42. A. Nel, T. Xia, L. Madler, and N. Li, "Toxic potential of materials at the nanolevel," *Science*, vol. 311, no. 5761, p. 622, 2006.
43. O. Le Bihan, P. Bonnafous, L. Marak, T. Bickel, S. Trépout, S. Mornet, F. De Haas, H. Talbot, J. Taveau, and O. Lambert, "Cryo-electron tomography of nanoparticle transmigration into liposome," *Journal of structural biology*, 2009.
44. W. Hoppe, "Towards three-dimensional "electron microscopy" at atomic resolution," *Naturwissenschaften*, vol. 61, no. 6, pp. 239–249, 1974.

45. K. H. Downing, H. Sui, and M. Auer, "Electron tomography: A 3d view of the subcellular world," *Analytical Chemistry*, vol. 79, pp. 7949–7957, 11 2007.
46. P. Midgley, E. Ward, A. Hungría, and J. Thomas, "Nanotomography in the chemical, biological and materials sciences.," *Chem Soc Rev*, vol. 36, no. 9, pp. 1477–94, 2007.
47. J. Taveau, D. Nguyen, A. Perro, S. Ravaine, E. Duguet, and O. Lambert, "New insights into the nucleation and growth of PS nodules on silica nanoparticles by 3D cryo-electron tomography," *Soft Matter*, vol. 4, no. 2, pp. 311–315, 2008.
48. S. Nickell, F. Fôrster, A. Linaroudis, W. D. Net, F. Beck, R. Hegerl, W. Baumeister, and J. M. Plitzko, "Tom software toolbox: acquisition and analysis for electron tomography," *Journal of Structural Biology*, vol. 149, no. 3, pp. 227 – 234, 2005.
49. Y. Kang, K. Engelke, and W. A. Kalender, "Interactive 3d editing tools for image segmentation," *Medical Image Analysis*, vol. 8, pp. 35–46, March 2004.
50. S. Senan, J. van Sornsen de Koste, M. Samson, H. Tankink, P. Jansen, P. J. C. M. Nowak, A. D. G. Krol, P. Schmitz, and F. J. Lagerwaard, "Evaluation of a target contouring protocol for 3d conformal radiotherapy in non-small cell lung cancer," *Radiotherapy and Oncology*, vol. 53, no. 3, pp. 247 – 255, 1999.
51. L. Grady, "Minimal surfaces extend shortest path segmentation methods to 3D," *tpami*, vol. 32, pp. 321–334, Feb. 2010.
52. M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *IJCV*, vol. 1, pp. 321–331, January 1988.
53. J. A. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.
54. N. Paragios, *Geodesic Active Regions and Level Set Methods : Contributions and Applications in Artificial Vision*. PhD thesis, INRIA Sophia Antipolis, Jan. 2000.
55. L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.
56. Y. Boykov and M. P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images," *ICCV'01*, vol. 1, pp. 105–112 vol.1, 2001.
57. V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?," *PAMI*, vol. 26, no. 2, pp. 147–159, 2004.
58. Y. Boykov and V. Kolmogorov, "Computing geodesics and minimal surfaces via graph cuts," *ICCV'03*, pp. 26–33, 2003.
59. M. Nikolova, S. Esedoḡlu, and T. F. Chan, "Algorithms for finding global minimizers of image segmentation and denoising models," *SIAM JAM*, vol. 66, no. 5, pp. 1632–1648, 2006.
60. V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International Journal on Computer Vision*, vol. 22, no. 1, pp. 61–79, 1997.
61. A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*. IEEE Press, 1999.
62. G. Möebus and B. Inkson, "Novel nanoscale tomography modes in materials science," *Microscopy and Microanalysis*, vol. 9, no. Suppl. 02, pp. 176–177, 2003.
63. S. Beucher and C. Lantuéjoul, "Use of watersheds in contour detection," in *Int. Workshop on Image Processing*, (Rennes, France), CCETT/IRISA, Sept. 1979.
64. F. Meyer and S. Beucher, "Morphological segmentation," *Journal of Visual Communication and Image Representation*, vol. 1, pp. 21–46, Sept. 1990.
65. Y. Boykov and V. Kolmogorov, "Computing geodesics and minimal surfaces via graph cuts," in *International Conference on Computer Vision*, (Nice, France), pp. 26–33, October 2003.
66. J. L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.
67. M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–331, 1988.
68. J. Sethian, *Level set methods and fast marching methods*. Cambridge University Press, 1999. ISBN 0-521-64204-3.

69. V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International Journal on Computer Vision*, vol. 22, no. 1, pp. 61–79, 1997.
70. A. J. Kinderman and J. F. Monahan, "Computer generation of random variables using the ratio of uniform deviates," *ACM Trans. Math. Softw.*, vol. 3, pp. 257–260, September 1977.
71. R. Stevanovic, G. Topic, K. Skala, M. Stipcevic, and B. Rogina, "Quantum random bit generator service for monte carlo and other stochastic simulations," in *Large-Scale Scientific Computing* (I. Lirkov, S. Margenov, and J. Wasniewski, eds.), vol. 4818 of *Lecture Notes in Computer Science*, pp. 508–515, Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-78827-0_58.
72. F. S. Foundation, "The gnu scientific library (gsl)," 2010.
73. R. Goldman, "Curvature formulas for implicit curves and surfaces," *Comput. Aided Geom. Des.*, vol. 22, no. 7, pp. 632–658, 2005.
74. H. Blum, "An associative machine for dealing with the visual field and some of its biological implications," in *Biological Prototypes and synthetic systems*, vol. 1, pp. 244–260, 2nd Annual Bionics Symposium, Cornell Univ., E. E. Bernard and M. R. Kare eds., Plenum Press, New-York, 1961.
75. S. Lobregt, P. Verbeek, and F. Groen, "Three-dimensional skeletonization: Principle and algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, pp. 75–77, Jan. 1980.
76. R. Zrour and M. Couprie, "Discrete bisector function and euclidean skeleton," in *Discrete Geometry for Computer Imagery: 12th International Conference, DGCI 2005* (E. Andres and P. Damiand G., Lienhardt, eds.), vol. 3429 of *LNCS*, (Poitiers, France), pp. 216–227, Springer, April 2005.
77. D. N. Mastronarde, "Dual-axis tomography: An approach with alignment methods that preserve resolution," *Journal of Structural Biology*, vol. 120, no. 3, pp. 343 – 352, 1997.
78. H. Chen, D. D. Hughes, T.-A. Chan, J. W. Sedat, and D. A. Agard, "Ive (image visualization environment): A software platform for all three-dimensional microscopy applications," *Journal of Structural Biology*, vol. 116, no. 1, pp. 56 – 60, 1996.
79. E. Breen and R. Jones, "Attribute openings, thinnings and granulometries," *Computer Vision, Graphics and Image Processing*, vol. 64, no. 3, pp. 377–389, 1996.
80. P. Hough, "Methods and means for recognizing complex patterns." US Patent 3069654, 1962.
81. J. Illingworth and J. Kittler, "A survey of the hough transform," *Comput. Vision Graph. Image Process.*, vol. 44, no. 1, pp. 87–116, 1988. Important reference.
82. P. Cheng, D. Li, L. Boruvka, Y. Rotenberg, and A. Neumann, "Automation of axisymmetric drop shape analysis for measurements of interfacial tensions and contact angles," *Colloids and Surfaces*, vol. 43, no. 2, pp. 151 – 167, 1990. Selected Papers from a Symposium on Recent Progress in Interfacial Tensiometry, held at the Third Chemical Congress of North America.
83. A. Fitzgibbon, M. Pilu, and R. B. Fisher, "Direct least square fitting of ellipses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 476–480, 1999.
84. S. Banerji and M. Hayes, "Examination of nonendocytotic bulk transport of nanoparticles across phospholipid membranes," *Langmuir*, vol. 23, no. 6, pp. 3305–3313, 2007.
85. M. Geiser, B. Rothen-Rutishauser, N. Kapp, S. Schürch, W. Kreyling, H. Schulz, M. Semmler, V. Im Hof, J. Heyder, Gehr, and P. Environ, "Ultrafine particles cross cellular membranes by non-phagocytic mechanisms in lungs and in cultured cells," *Environ Health Perspect.*, no. 113, pp. 1555–1560, 2005.
86. B. M. Rothen-Rutishauser, S. Schurch, B. Haenni, N. Kapp, and P. Gehr, "Interaction of fine particles and nanoparticles with red blood cells visualized with advanced microscopic techniques," *Environ. Sci. Technol*, vol. 40, no. 14, pp. 4353–4359, 2006.