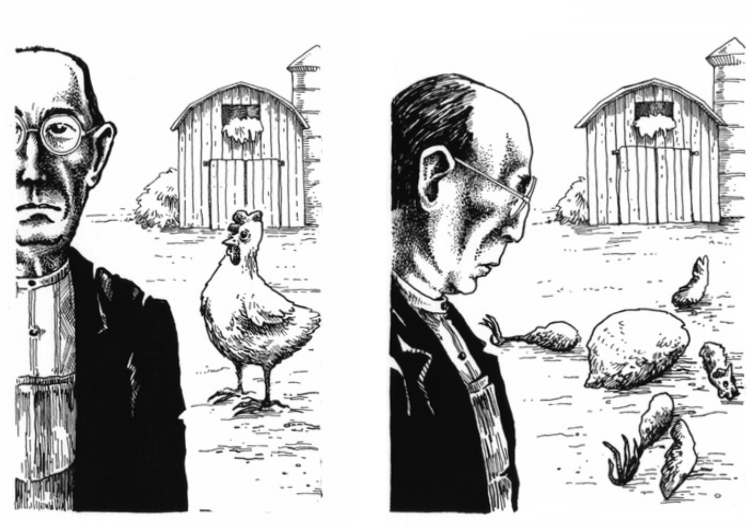
Laszlo Marak

supervisor:
Hugues Talbot

Segmentation with minimum surfaces



à mes parents

We naturally see objects as being composed out of parts. Instead of perceiving indivisible objects, we perceive objects in terms of their labelled or categorized parts. We break objects into parts that we have learned are relevant or important.

Table of Contents

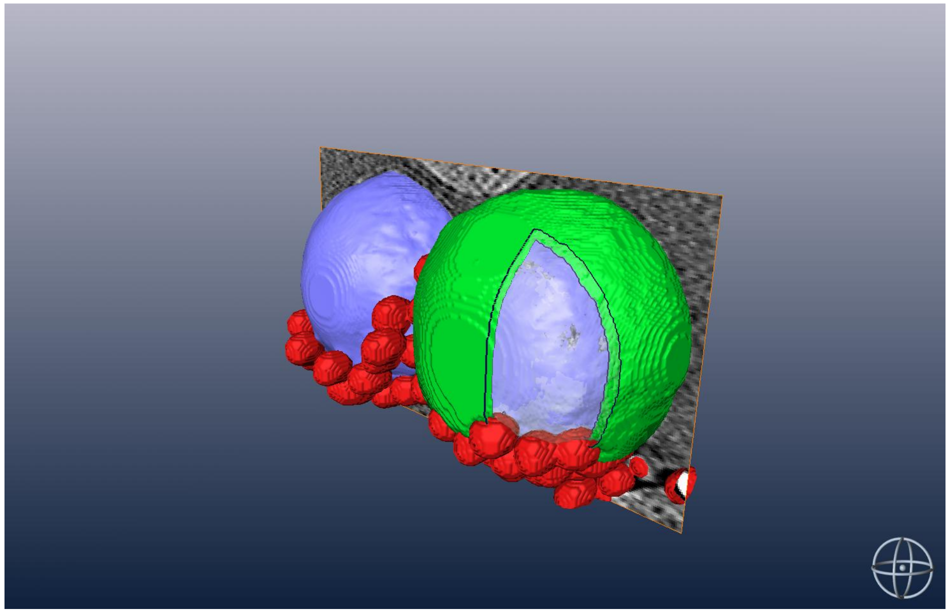


Figure 1: An example segmentation.

Default view

View 1

View 2

About this document

In the past few years more and more documents are published electronically. Usually the scientific publications tend to be conservative but there is more and more special content, that is difficult to describe with text or 2D images. Sometimes lot of information can be extracted from multi-dimensional images just by looking at them.

The references also pose a particular problem. The more illustrations we want to include to a document, the more difficult is to find them hence they can be several pages away from the reference in the text.

Modern standards try to target all of these issues, but unfortunately there is still only a little use of them for professional publishing. Recently [BF07] addressed the issue showing how can 3D data be incorporated in Adobe PDF documents for scientific use. The first scientific article using advanced PDF features was [FB08].

We have prepared this document using as much advanced PDF features as possible to make the reading more convenient. If you have troubles reading the article, please consult the How to use this document section.

Introduction

Image segmentation is the task of finding the boundaries of regions of interest in images. While humans are very good at pattern recognition tasks[1], image delineation (i.e. the task of marking precise contours locations) is usually difficult and time-consuming for people [MFTM01]. It is even more difficult with 3D data, as topological constraints are difficult to enforce when working slice-by-slice [KEK04]. Alternatively, specifying surface elements is a very challenging interactive task.

Nonetheless sufficiently precise contours are necessary for accurate measurement derived from image data. This task can often best be performed using automated segmentation methods using human feedback to correct for contour placement mistakes [Gra06].

In addition, automated image segmentation, particularly using optimization frameworks, can provide relatively unbiased contour placements that are not overly influenced by human interpretation, interpolation and expert opinions.

In this document we discuss the some special problems of segmentation and the segmentation strategies, we have developed to solve them. In part III. we try to formulate the problem as well as describe the theoretical background. In part IV. we give a view to the software implementation with a particular stress on the question of optimization. In the third big part, part V. we present segmentation strategies, developed for segmenting three dimensional nanotomographic images.

Publications related to this report

[BBM$^+$08] Olivier Le Bihan, Pierre Bonnafous, Laszlo Marak, Sylvain Trépout, Stéphane Mornet, Hugues Talbot, Jean-Christophe Taveau, and Olivier Lambert. Nanoparticle transport across phospholipid membrane. *submitted*, 2008.

[MTLT08] Laszlo Marak, Hugues Talbot, Olivier Lambert, and Jean-Christophe Taveau. Segmentation techniques for the analysis of electron nano-tomography images. In *3D-IMS*, Carcans-Maubuisson, September 2008. Accepted.

[TMTD07] 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.

[1] such as recognizing features in images

Statement of the problem and the proposed solution

In our case we are trying to delineate objects on the images. These images are 2D or 3D but there is also a possible interpretation for 4D images. The key observation in our case is that the objects are limited by contours. A contour is the physical limit of the object (a curve in 2D and a surface in 3D). As the object is different from its environment the contours are the places where the characteristics of the image, the texture is changing.

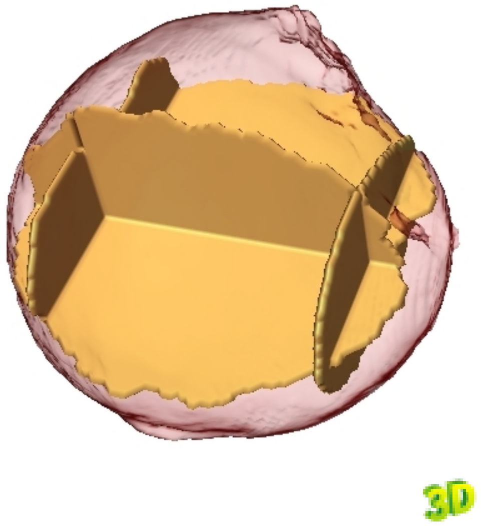


Figure 2: An example of the source. The source is composed from the results of several 2D segmentations.

In the ideal case, on a noiseless image (I_{opt}) we can find the contours with the gradient operator. In places where the texture is changing (high frequency places) the gradient is also high while if the texture is constant, the gradient will be low.

The real images (I_{real}) however, are often affected with different type of noise. The noise can be pictured as the ideal picture would be altered by a probability variable X:

$$I_{\text{real}}(P) = I_{\text{opt}}(P) + X \tag{1}$$

If X is indeterministic enough[2], then there are some patterns that never occur. For example let X be a U(0,1)[3]. If we look on any closed curve in the gradient of the image, the probability that on that curve most of the values are high is very low. For this reason if we notice that on a curve there is a high measure of high frequency values, it is most likely not caused by the noise, but the curve is on a contour. This means that if we find the curve with the **highest measure of high gradient values** it will most probably be the contour of the object. The way how we find these contours and the proper formulation follows in the next section.

1. Geodesic active contours and surfaces

In this section we formulate the problem described above in terms of minimal surfaces[4]. We assume sufficient regularity for all func-

[2]in the other case, when X is deterministic, it can be filtered
[3]a probability variable with uniform density function in the $[0 \ldots 1]$ interval
[4]The use of term **minimal** here is arbitrary, as for any measure, if we take the inverse, the minimal surfaces become maximal.

tions, which are practice always met in physical systems.

The idea is the following: let G be a Riemannian metric on Ω (a compact subset of $\mathbb{R}^n$) as well as S (referred as the *source*) and P (referred as the *sink*) to be two disjoint subsets of Ω. Let all closed hyper-surfaces s (not necessarily connected) be those that contain the source and do not contain the sink. We can define the following functional:

$$E(s) = \oint_s \mathrm{d}G, \tag{2}$$

called the weight or cost of s. As $E(s)$ is smaller than $E(\partial S) < \text{area}(\partial S) \cdot \max(g)$ (that is to say finite for bounded g-s) there exist at least one hyper surface M with minimum weight [Str83]. In our case we mostly use the following metrics:

$$\mathrm{d}G = \frac{\mathrm{d}x}{1 + \nabla I} \text{ and} \tag{3}$$

$$\mathrm{d}G = \frac{\mathrm{d}x}{1 + \triangle I} \tag{4}$$

These are the simplified versions of those proposed in [CKS97]. As imposed in part III the contour of the object is a closed hyper surface[5] which "uses" the most high frequency places from Ω. This problem has proven to be difficult to calculate directly, there exists an algorithm based on simulating differential equations, that can find the optimum surfaces[6]. This algorithm will be presented in the next section.

2. Maximal flows

In this section we describe the method we used for the segmentation. In a discrete domain endowed with an L_1 metric, with exactly one pointwise source and sink[7]. This minimum surface can be computed using the well-known Ford and Fulkerson maximum flow graph algorithm [FF62], which were recently improved in the context of images [BK03].

[5]hyper surface is a set with one less dimension than the field Ω (that is to say a curve in 2D, a surface in 3D and possibly a volume in 4D)

[6]The idea is similar to those of the Euler-Lagrange equations. The basic idea is that you want to minimize a functional. For this you find a corresponding differential equation which you can solve numerically. The solution of the differential equation will be the function that minimizes the functional. If you are more interested in the classical Euler-Lagrange equation you can read more of it in the Annexes.

[7]but not necessarily a single point inside the image

In the continuous domain (with arbitrary Riemannian metric), also for one source and one sink, M can be computed directly from every G and sets P, T using for example active contours or surfaces models [KWT88], or level-sets methods [Set99, CKS97]. However these methods compute surfaces iteratively via gradient descent schemes, and thus the solution is only locally optimal – hence depends on initialization and noise levels. The algorithm we present here was first described in [AT06]. It provides an optimum solution to this problem. Before describing the algorithm, we would like to take a look why Ford and Fulkerson algorithm cannot provide sufficient results for image segmentation.

2.1. The discrete case

Definition 1. *Discrete flow: Let $G(V, E)$ be a graph with edge costs C_E now reinterpreted as capacities. A flow $F : E \to \mathbb{R}$ from the source S to the sink P has the following properties:*

- ***Conservation of flow**: The total (signed) flow in and out of any vertex is zero.*
- ***Capacity constraint**: The flow along any edge is less than or equal to its capacity:*

$$\forall e \in E,\ F(e) \leq C_E(e) \tag{5}$$

An edge along which the flow is equal to the capacity is described as saturated. Ford and Fulkerson [FF62] demonstrated that the maximal s-t flow equals the minimal s-t cut, with the flow saturated uniformly on the cut. We want to interpret the s-t cut with the tools of analysis. For this we need the following definitions:

Definition 2. *Closed discrete surface on edges[8]: Let $G(V, E)$ be a directed graph on a finite set of points V and let $T \subseteq V$.*

$$B := \bigcup_{v \in T,\ p \notin T \text{ and } v \to p \in E} v \to p \tag{6}$$

*is a **closed discrete surface on edges** of G.*

With simple words the closed discrete surface on edges is the minimal set of edges, we must use to get out of T. Note that the opposite is also true: For a set of edges B if there exists a T such as definition 2. applies, B is a discrete surface. Somewhat analogously we can define the surface on points too:

[8]This definition is used only for this document, in scientific literature you might find different definitions.

Definition 3. *Closed discrete surface on points: Let $G(V, E)$ be a directed graph on a finite set of points V and let $T \subseteq V$. We say that $v \in T$ is a **border point** if there exists an edge $v \to p \in E$ where $p \notin T$. For a set T the set of its border points C is a **discrete closed surface on points** of G.*

The same way as before if for a given set of points C there exists a "volume" T, it is an closed surface. We say for two discrete surfaces S_1 and S_2, that $S_1 \in S_2$ if and only if $T_1 \subseteq T_2$ where T_1 and T_2 are the corresponding volumes respectively. To point out the main inconvenience of the discrete graph segmentation we define two metrics, one of which is used in the classical Ford and Fulkerson [FF62] algorithm and other in the continuous approach [AT06].

Definition 4. *G_{ford}: Let $G(V, E)$ be a finite graph, $Z \subseteq E$ and $\mu : \mathcal{P}(E) \to \mathbb{R}$ the following:*

$$\mu(Z) = \sum_{e \in Z} C_E(e) \tag{7}$$

*where $\mathcal{P}(E)$ represents the powers of E and C_E is the cost of the edges. $(E, \mathcal{P}(E), \mu)$ is a **measure space**.*

Definition 5. *G_{cont}: Let $G(V, E)$ be a finite graph, $T \subseteq V$ and $\nu : \mathcal{P}(V) \to \mathbb{R}$ the following:*

$$\nu(T) = \sum_{v \in T} C_V(v) \tag{8}$$

*where $\mathcal{P}(V)$ represents the powers of V and C_V is the cost of the points. $(V, \mathcal{P}(V), \nu)$ is a **measure space**.*

With these metrics we can very well define the cost of a graph cut in the Ford and Fulkerson's case:

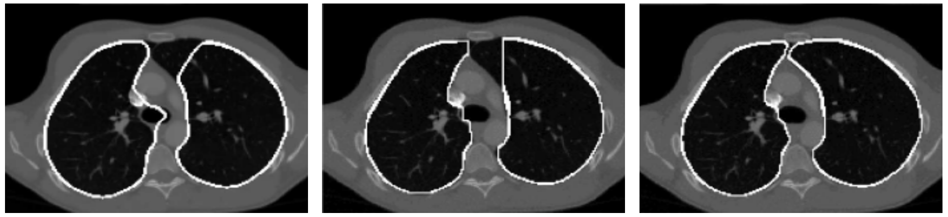
$$E_{ford}(S_{edge}) := \int_{S_{edge}} \mathrm{d}G_{ford} \tag{9}$$

and in the continuous case:

$$E_{cont}(S_{point}) := \int_{S_{point}} \mathrm{d}G_{cont} \tag{10}$$

The algorithm Ford and Fulkerson can solve the case of the equation (9), that is to say find the minimum surface for any cost function C_E. It is a good tool for the original problem it has been developed for[9], it

[9]Initially the maximal flow problem was developed for transportation. They were looking for a tool which could minimize the total kilometers for the railway network. It was published in the 1950s but known before, for example it was used in WWII. for military purposes.



(a) (b) (c)

Figure 3: The optimal curves on images. On image (a) there is a locally optimal curve. The globally optimal 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.

cannot be however trivially be extended to the second (10) case. The main problem is, that in the first case we cannot explicitly influence the total flow that flows through a point. In fact at the end of the segmentation this "saturated" flow at v can vary from $\min(C_E(v \to p))$ to $\sum\limits_{v \to p} C_E(v \to p)$. Now if we take the case where G represents an image (that is to say G is a staggered grid) some directions are more preferred than the others (see picture 3.). In the case of equation (10) the limit on the edges can vary according to the actual direction of the flow ensuring a **direction independent** constraint. In the special case, when the edges are parallel with the axes, we can find the optimum surface with continuous flow. The question is how do we find the optimum surface according to $\mathrm{d}G_{cont}$. The next section describes how can the problem be solved with a set of differential equations.

2.2. The continuous case

In the continuous case the set of points is replaced by a continuous scalar field P and the set of the edges is replaced by a continuous vector field $\mathbf{F}$. $\mathbf{F}$ is a flow if it satisfies two conditions:

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

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

$$\frac{\partial P}{\partial \tau} = -\nabla \cdot \mathbf{F} \tag{11}$$

$$\frac{\partial \mathbf{F}}{\partial \tau} = -\nabla P \tag{12}$$

$$|\mathbf{F}| \leq g \tag{14}$$

Here P represents a pressure field and F a vector field. P is forced to 1 on the source and 0 on the sink. The equation can be solved numerically (by simulation). Now let suppose that it converges for a given g. If the system is stable following statements apply:

$$\nabla \cdot \mathbf{F} = 0 \tag{15}$$

$$\nabla P = 0 \qquad \text{if } |\mathbf{F}| < g \tag{16}$$

$$\nabla P = -\lambda \mathbf{F} \qquad \text{if } |\mathbf{F}| = g \tag{17}$$

The equation (15) simply restates the conservation of the flow. Equation (16) applies if the flow have stabilized during the evolution without the (14) constraint. If without the constraint the flow would grow higher, still because the system is stable, it cannot change the direction or decrease the magnitude. From (12,14) we can deduce that $\nabla P \cdot \mathbf{F} \leq 0$, which means that P is a non strictly monotone decreasing function along the flow lines. If $\mathbf{F}$ 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 (14) applies thus:

$$\int\limits_A \nabla \cdot F_Y = \oint\limits_Y N_Y \cdot \mathbf{F} \mathrm{d}Y = \oint\limits_Y g \mathrm{d}Y = \oint\limits_Y \mathrm{d}G \tag{18}$$

This implies, that every iso-surface is minimum. If there is only one minimum this also means, that the pressure field can be $0 \leq P \leq 1$ on a zero Riemann measure set. In the next section we present the algorithm along with the software implementation and the optimization.

The software for segmentation

In this part we present the algorithm for the simulation of the (11, 12, 14) system and the some problems one can met during the implementation. The software is called **uiFibres**. As the system is simple, one code can handle arbitrary dimensions (tested up to 3D). Our code is also parallelized and it can use the resources core independently on every system which supports POSIX-threads.

3. Implementation

Equations (11) and (12) are discretized on a staggered grid using an explicit first-order scheme in time and space. The scalar field P is stored on grid points while the flow $\mathbf{F}$ is stored by component on grid edges. The system of equations is iterated sequentially with the flow magnitude constraint (14) enforced after each time step. Before we present the formulas we explain some conventions. In the formulas, the vectors are typeset bold. For example $\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$. Also $\mathbf{i}_k$ represents the unit vector in the k-th direction. For example $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \mathbf{i}_2 = \begin{pmatrix} x_1 \\ x_2 + 1 \\ x_3 \end{pmatrix}$, as well as $\mathbf{i} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. We mark with $\vec{F}$ the local interpretation of the flow. In a given point $\mathbf{X}$ of the flow, there are six possible directions to go. For example $\vec{F}_{\mathbf{X}} \cdot \mathbf{i}_3 = \begin{pmatrix} [F_1^{\text{in}}, F_1^{\text{out}}] \\ [F_2^{\text{in}}, F_2^{\text{out}}] \\ [F_3^{\text{in}}, F_3^{\text{out}}] \end{pmatrix} \cdot \mathbf{i}_3 = [F_3^{\text{in}}, F_3^{\text{out}}]$. Note that $\vec{F}$ is defined locally. That means that the outflow in direction $\mathbf{i}_2$ of point $\begin{pmatrix} 9 \\ 9 \\ 9 \end{pmatrix}$ will be an inflow of the point $\begin{pmatrix} 9 \\ 10 \\ 9 \end{pmatrix}$. We use colored indexes for summarization. For example: $a_k b_k = \sum_k a_k b_k$. If the colored index is on both sides of the equation it notes a loop. For examples $a_k = b_{k,l} \cdot c_l \Leftrightarrow$ for each k, do $b_k := \sum_l b_{k,l} c_l$.

The iteration begins with updating the pressure. $\triangle\tau$ marks the time step.

$$ {}^{n+1}P_{\mathbf{X}} = {}^{n}P_{\mathbf{X}} - \triangle\tau \left({}^{n}\vec{F}_{\mathbf{X}}^{\text{out}} - {}^{n}\vec{F}_{\mathbf{X}}^{\text{in}} \right) \cdot \mathbf{i} \tag{19} $$

After calculating the equation (12) in each direction:

$$\mathbf{i}_k \cdot \vec{H}^{\text{out}} = \mathbf{i}_k \cdot {}^n\vec{F}^{\text{out}}_{\mathbf{X}} - \triangle\tau \left({}^{n+1}P_{\mathbf{X}+\mathbf{i}_k} - {}^{n+1}P_{\mathbf{X}}\right) \tag{20}$$

$$\mathbf{i}_k \cdot \vec{H}^{\text{in}} = \mathbf{i}_k \cdot {}^n\vec{F}^{\text{in}}_{\mathbf{X}} - \triangle\tau \left({}^{n+1}P_{\mathbf{X}} - {}^{n+1}P_{\mathbf{X}-\mathbf{i}_k}\right) \tag{21}$$

In equations (20,21), H holds the flow as it would evolve without the constraint. The magnitude constraint is applied immediately after the update of the flow velocity field by (20). The application consists of three stages:

1. Determine the maximal outward velocity flow along each axis:

$$\mathbf{i}_k \cdot \mathbf{J} = \max\left\{-\mathbf{i}_k \cdot \vec{H}^{\text{in}}, 0, \mathbf{i}_k \cdot \vec{H}^{\text{out}}\right\} \tag{22}$$

2. Compare the absolute maximal outward velocity to derivate of the measure, g:

$$\text{If} \quad |\mathbf{J}| > g_{\mathbf{X}} \quad \text{then} \quad \mathbf{J} = \mathbf{J}\frac{g_{\mathbf{X}}}{|\mathbf{J}|} \tag{23}$$

3. Apply the velocity constraint to each component:

$$\mathbf{i}_k \cdot {}^{n+1}\vec{F}^{\text{in}}_{\mathbf{X}} = \max\left\{\mathbf{i}_k \cdot \vec{H}^{\text{in}}, -\mathbf{i}_k \cdot \mathbf{J}\right\} \tag{24}$$

$$\mathbf{i}_k \cdot {}^{n+1}\vec{F}^{\text{out}}_{\mathbf{X}} = \max\left\{\mathbf{i}_k \cdot \vec{H}^{\text{out}}, \mathbf{i}_k \cdot \mathbf{J}\right\} \tag{25}$$

With this update scheme we can simulate the equations. As we decrease $\triangle\tau$ the system converges uniformly to the solution. However this finite differences analysis of the image involves a lot of operations. The trivial implementation of the algorithm runs slow. In the next section we explain how did we develop a working code for applications.

4. Optimization

Ever since the introduction of the computer, the speed has been an issue. In general, resource consumption is the most important property of an algorithm, a software or a computer. Reaching the limits of the technology the processors have become saturated in frequency [KDH+05]. As in sake for power the number of cores raises instead of the frequency of the processor. The algorithms have to be rethought in order to run optimally on new architectures. In this section we describe how can the maxflow algorithm be sufficiently accelerated to use most of the resources of modern systems. We also try to introduce the idea of statistical profilers and they use in such optimizations.

During the development we have released three sub-versions

- uiFibres RC1 The first release candidate, introducing the dibbles and a completely rewriting the original engine, it was the first release optimized for prefetching.
- uiFibres RC2 The second release candidate, using the dibbles and introduced the threads. The use of threads improves the performance quasi-linearly according to the number of the cores[10].
- uiFibres VER1 The first release of uiFibres, introduces a cleaned up code to ensure the ease of any further development, profiling and making the last minor optimizations. The first release can work either with her own engine or with contmaxflow, uses the same parametrization and can be compiled with or without threads[11]. The first release is 4D data ready.

In the next section we describe the profiling and its interest in image segmentation.

4.1. Profiling

Profiling is a run time performance analysis of the code. It tries to find the parts of the code which most occupie the hardware thus extending the runtime the most. It poses a separate problem how to profile the code. There are several theoretical approaches. One of them is the **statistical profiling**. Statistical profilers are guided by the kernel. They interrupt the software time to time and note where the program has been interrupted. We call this place (assembly command or line of the code) interruption a sample. Intuitively if the program spends more time in a given line it is more probable that we find it in that line if we randomly interrupt the system. Let's have a look at the following example:

```
#include <stdio.h>

int main(){
  int i, j;
  double accu = 0.0;
  for (i = 0 ; i < 1000 ; ++i)
```

[10]tested up to two cores
[11]also specifying their number

```
    for (j = 0 ;j < 1000000 ;++j){
      accu += 1.0;
    }
}
```

We would like to know which line is the most time consuming for the processor. In this case we interrupt the program time to time and note in which line the program has been. It can be proven that the probability that the program is in line a equals the proportion of the time the program spent in a with the total time of the program. We have a statistical guarantee that the proportion of the collected samples will tend vers this probability. Lets see the samples collected by the profiler:

```
                :#include <stdio.h>
                :
                :int main(){ /* main total: 538553 99.9967 */
                :  int i, j ;
                :  double accu = 0.0;
                :
                :  for (i = 0; i < 1000; ++i)
   297  0.0551 :    for (j = 0; j < 1000000; ++j) {
538256 99.9415 :      accu += 1.0;
                :    }
                :}
/*
 * Total samples for file : "/home/uj/tmp/oprhug_i/loop.c"
 *
 * 538553 99.9967
 */


/*
 * Command line: opannotate --source -o /home/uj/tmp/oprhug_i/ /home/uj/tmp/oprhug_i/loop
 *
 * Interpretation of command line:
```

```
 * Output annotated source file with samples
 * Output all files
 *
 * CPU: Core 2, speed 1867 MHz (estimated)
 * Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00
 * (Unhalted core cycles) count 10000
 */
```

In harmony with the intuition the most samples where collected in the line where the program makes the addition on a double type variable. In other cases it becomes more difficult to understand the samples.

4.2. Profiled uiFibres

The most time-critical library was the flow simulation. It works with high amount of data (especially when segmenting 3D high resolution images) and it performs lot of iterations. This was a piece of code, we believed to be time consuming:

```
vReset_fast(curr,d);
do {
  loc_sum = 0;
  w = d - 1;
  loc_pos_out = Position_fast(dim, curr, d);//outflow
  do {
    if (curr[w]>0){
    curr[w]--;
    loc_pos_in=Position_fast(dim,curr,d);//inflow
    curr[w]++;
    loc_sum += flow_glob[w*length_glob+loc_pos_out] - flow_glob[w*length_glob+loc_pos_in];
    w--;
  } while ( w>=0 );
  pot_glob[loc_pos_out]-= tau * loc_sum;
} while (uiNextStep_fast( dim, curr,d ));
```

The lines are correspondent with the equation (19). After profiling, we have noticed, that the result showed no jam at any line. This means that the code runs equally slow. In these cases there is a chance, that the

◀◀ ▶▶
◀ ▶
◀ Doc Doc ▶
Back Close

compiler cannot optimize the code. In the next section we describe how a code optimized by the compiler.

• A simple cpu model

In the most of the cases the we use a very simple cpu model. The cpu is a processing unit, which can execute a given set of commands called the assembly. If we do not code in assembly[12], then the “source code” (C++, Python, etc.) has to be compiled. The compilation gives a series of commands that have to be executed. If we do not use parallelization, than our code is called sequential. In a sequential program the commands have to be executed in a given order. For example given the code:

```
1) for (int q=0; q<=10 - 1; q++){
2)   sum+=f[q];
3) }
```

If we look at lines 1), 2), 3) together, then we see, that we read the elements of f in their native order[13] and in the q-th iteration we only alter the q-th element. This means that while we calculate in line 2) we could read the next element $f[q+1]$ as we are going to need it in the next iteration. We can do this, because we know that $f[q+1]$ will not be altered until the next time we execute line 2). In the classical CPU-s we had to wait to any command to finish before we could launch the next one. Introducing the pentium family however the architecture of the cpu-s have changed. Nowadays there are separate units which deal with the floating-point operations (like addition) and the memory reading. This means that line 2) could be launched two at a time if the compiler would know their local dependency. If we want the compiler to find these possibilities, we have to introduce a more complex CPU model. We do this in the next section.

• More Complex CPU model

In the simple model we have the CPU and the RAM. The CPU reads the memory according to the binary, performs instruction on it and writes it back. The problem with this model is, that the processor is much more performant that the memory is. That is to say during one read from the memory the CPU could do hundreds of instructions. The solution developed for this problem is the middle memory, the cache. The CPU has a cache with lower penalty but smaller size[14]. This work very well in problems where one can perform several

[12]The assembly language is the native language of the cpu. The functions used in assembly can be directly executed. It is however not too widespread because the coding in it is very difficult. The high level languages are translated into assembly by the compiler. As the translation is not unique there can be huge differences in the compiled codes.
[13]in the order they are stored in the memory
[14]Usually the size of the cache is a thousand time less then the size of the RAM.

◀◀ ▶▶
◀ ▶
◀ Doc Doc ▶
Back Close

operations on a small patch of the data before writing it back to the RAM. In scientific problems especially in image analysis this works well if one can partition the data and perform at least a hundred iterations on each part. Unfortunately the most of the algorithms are not like this. Imagine a simple bubble sort. In each iteration one needs to access all the elements of the array. We call these problems the streaming. In this case even if one takes an element, one can perform only one operation on it. Note that the partitioning problem is a sub-problem of the parallelization problem.

The solution proposed is an altered version of the cache. To improve the speed of the streaming applications the developers have introduced a new system of memory management, called the "Prefetching". It will be described in the following section.

4.3. Prefetching

First let us describe the reading penalties of the RAM. The RAM is random access memory. In systems with RAM we want to access any stored element in constant time, regardless the previously read elements. The normal access time to an element of the RAM is perpetual to the logarithm of the size of the RAM. As the size of the RAM is not changing in runtime, this penalty is constant for every reading. We call this time the **random access penalty**. If we read the memory sequentially[15], we can save the time of accessing.

To maintain the possibility of random access the developers have proposed the model of prefetching which will be the base of our most complex model in this part. The system works as follows. One can read a random element from the RAM, and together with it the system "fetches[16]" the few[17] following elements too. That is one always reads a packet with reasonable size.

Imagine a packet of size 50. If one wants to calculate $a = \sum b + \sum c$, one reads the first element of b and c, one puts it into the middle memory, the cache. The system also puts together with a and b the next 49 elements of b and c. In the next steps one does not need to read from the RAM because one already has the elements in the cache. This way one detaches herself from the RAM and only reads from the RAM for the next 50 iterations. This is called the prefetching. There is an illustration of the model on fig 4.

The prefetch is controlled in compiler level or sometimes at hardware level. In theory the CPU offers this service automatically. As the module that controls the prefetching is limited in capacity, it misses the opportunity very often. As the internal build of the module is not public, the user have to optimize the code manually achieve the most of the prefetches.

[15] we know, that the next read will be the following cluster
[16] reads
[17] the size of the segment vary along further considerations

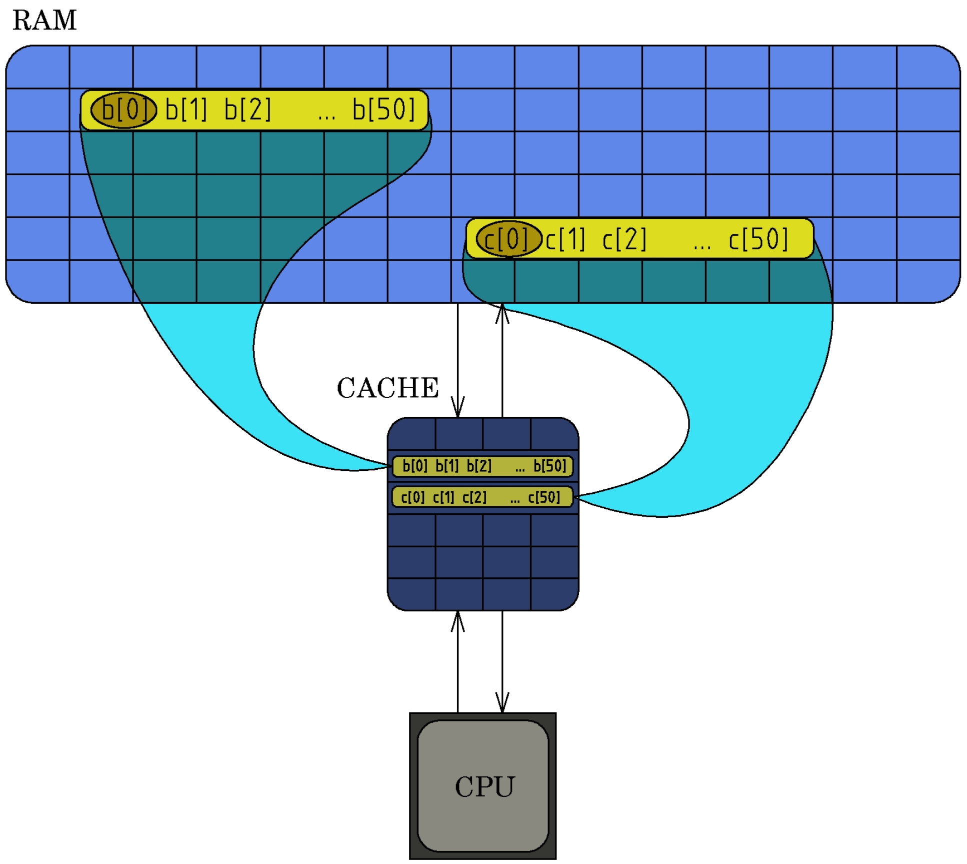


Figure 4: Theory of prefetching. With b_0 and c_0 the next few elements are moved too.

• The keyhole problem

As discussed in the simple cpu model section, to be able to optimize the code we have to do heuristics on several lines of the code. However the heuristics is an exponential problem, so it can be executed only for small inputs. That is why the compiler optimizer looks at once only into a little peace of the code (literally a few lines), so it can easily miss the optimization in **long loops**. This is the case in the example 4.2. There are just so many tests, that the compiler cannot see the forest from the tree. The (manually) optimized code looks like this:

```
uiVal_type * p_c;
uiVal_type * f_out;
uiVal_type * f_in;
int fm1, fm1_vec[d], start,end,length, q, w, e;

for (w=0; w<=d-1; w++){
  vReset_fast(fm1_vec,d);
  fm1_vec[w]=1;
  fm1=Position_fast(dim, fm1_vec, d);
  for (e=startDibble; e<=endDibble-1; e++){
    start=dibPotencial->Values[e].start;
    end=dibPotencial->Values[e].end;

    p_c=&(pot_glob->Values[start]);
    f_out=&(flow_glob->Values[w*length_glob+start]);
    f_in=&(flow_glob->Values[w*length_glob+start-fm1]);
    length = end - start;
    //the hyper-super ultra fast loop
    for (q=0; q<=length-1; q++ ){
      p_c[q] -= tau * ( f_out[q]-f_in[q] );
    };
  };
};
```

◀◀ ▶▶
◀ ▶
◀ Doc Doc ▶
Back Close

As it is noticeable, the length of the code is about the same. However in this case the main part of the code is in lines 19), 20), 21). The other part is just the translation of the memory addresses on which we want to iterate. The profiled version of the code looks like this:

```
               :  uiVal_type * p_c;
               :  uiVal_type * f_out;
               :  uiVal_type * f_in;
               :  int fm1, fm1_vec[d], start,end,length, q, w, e;
               :
     1 2.6e-05 :  for (w=0; w<=d-1; w++){
               :    vReset_fast(fm1_vec,d);
               :    fm1_vec[w]=1;
     1 2.6e-05 :    fm1=Position_fast(dim, fm1_vec, d);
  3725  0.0980 :    for ( e=startDibble; e<=endDibble-1; e++){
  1455  0.0383 :    start=dibPotencial->Values[e].start;
   485  0.0128 :    end=dibPotencial->Values[e].end;
               :
  1297  0.0341 :    p_c=&(pot_glob->Values[start]);
   732  0.0193 :    f_out=&(flow_glob->Values[w*length_glob+start]);
  1368  0.0360 :    f_in=&(flow_glob->Values[w*length_glob+start-fm1]);
   264  0.0069 :    length = end - start;
               :    // the hyper-super ultra fast loop
 36245  0.9539 :    for (int q=0; q<=length-1; q++ ){
386811 10.1803 :      p_c[q] -= tau * ( f_out[q]-f_in[q] );
               :    };
               :  };
               :};
```

This piece of code contains many interesting examples. First we can see that it is indeed the floating point operation that takes the longest time, that is to say the tight traverse section moved from the memory towards the processor, what we know to be the fastest. As mentioned before, the key to the performance is the minimalist loop in the end. This is the loop which the compiler can understand and optimize. We can also see another example of prefetching. In the lines:

```
  1455  0.0383 :    start=dibPotencial->Values[e].start;
   485  0.0128 :    end=dibPotencial->Values[e].end;
```

The Values is a structure:

```
typedef struct{
  int start;
  int end;
} uiDibble
```

In the compiled structure the end follows immediately after start. We can see, that it takes much longer time to read start than to read the prefetched end. In the next section we will explain, how is did we change the algorithm to be able to partition it into one minimalist loop.

4.4. Dibbles

In this section we explain the considerations which helped us to reorder the algorithm for optimization. The main purpose was to perform the iteration without any testing. Typically a major part of the code runs only in a minor part of the execution time. For example, if we do a calculation which involves the neighbor elements, we often need to test if we are not on the border. We do this test at every element, though typically most of the elements are not on the border. If for a given element we know, that the next k elements have the same properties, then we can skip the test on them. This is the idea that we used to eliminate the tests. Before the flow can be added to a potential point, we perform some tests. Some of them will be true, some of them will be false. As we have a finite number of tests, we have limited overall possibilities (namely 2^n). So the points can be partitioned into at most 2^n categories.

Definition 6. *Dibble: A dibble is a subset of points meeting three conditions. They are situated on the same line, they belong to the same category and all their neighbors but two are in the same dibble.*

A dibble is a segment on which we can iterate with performing the test only on its first element. On fig 5. we illustrate the therm in 2D. First we break the space into dibbles, than we iterate on the dibbles without performing any additional tests.

In uiFibres the categories are the following:

- Potential
  Every point if it is not sink our source but the side points have all their flows accessible. So we simply

Dibbles

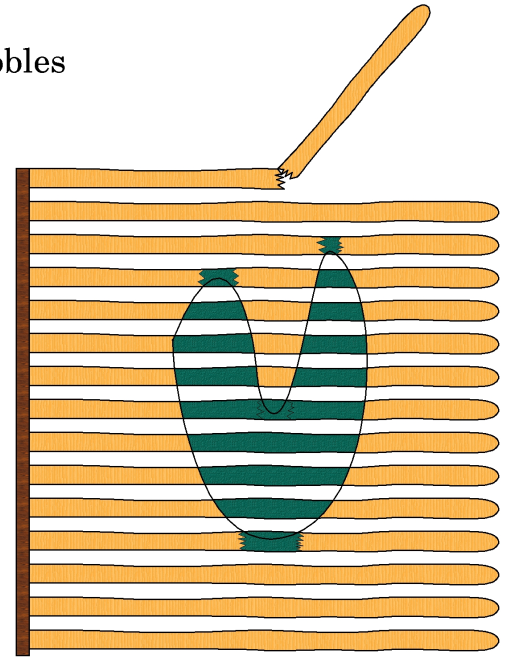


Figure 5: The image can be understood as lots of “dibbles” next to each other. The brown dibbles have the same characteristics. Each brown dibble can access the flow in all direction. The green dibbles are those, which are connected to the source. As the source stays the same, they need not be calculated accelerating the run of the program.

demand that all side points be sink. This way every regular point can belong to a dibble. Than we can perform the default operation with the dibbles.

- Flow
  Every flow arrow having at least for one end a regular point can be accessible. As all the side points are sink we have no trouble with flow arrows elongating into the next line.

- Constrain
  It is the most difficult to define a condition that would work and all good points would belong to the same category. The condition is the following: all the points that are either normal or are source and have at least one 4-neighbor normal shall belong to dibbles. We will not prove here the correctness of this condition, just remark that with this carefully chosen condition we can treat all the points the same way, easing the development, and avoiding all the source and sink but from one hyperplane around the source.

At the first iteration every point comes under a category. At the second iteration the field is partitioned to dibbles with the following algorithm:

```
• while current element $\neq$ last element:
    – dibbles$\rightarrow$append(create dibble from(current element))
• create dibble from(current element)
    – start:=current element
    – while next element is in the same category:
        ∗ current element:=next element
    – return dibble(from start to current element)
```

From the third iteration the dibbles are sent to the floating point processor unit. As these operations are highly deterministic, the compiler and the processor module can easily optimize them. The fact that the dibbles are independent opens an opportunity to provide these operations in parallel. The parallelization is explained in the next section.

◀◀ ▶▶
◀ ▶
◀ Doc Doc ▶
Back Close

4.5. Parallelization

As we saw in this example, the optimized code puts a high workload on the floating-point unit (FPU) of the processor. As all of the cores have an FPU, we would like to perform the iterations on all of them in parallel. The field is already partitioned, so we just create the POSIX-threads and send them a dibble each one by one. If one thread is finished with the dibble, it asks for the next. When all the dibbles are processed, we can move to the next iteration.
note: This is not true parallelization in the sense that in all iterations we access all the of points, but it at least saturates the system's bus.
The parallelization should shorten the time of execution. On systems with few cores (less then 8) we can expect linear growth in the execution time. The real benchmarks are presented in the next section.

4.6. Benchmarks

In this section we present the real time of execution of our implementation. The test were performed on the 1.86GHz dual core lab machine. The codes are compiled with gcc zero level optimization and debug mode leaving all the optimization to the processor.

- **The plane**
  - Without parallelization:

    Ben

    ```
    doing iteration 1550
    doing iteration 1560
    doing iteration 1570
    contmaxflow.c - Exiting
    total time of iteration: 00:00:12
    Image.max = 1.079079
    Image.min = -0.094653
    finished running
    ```

    UjoImro

    ```
    estimated time remaining: 00:00:00
    iteration 1401/1570
    ```

```
estimated time remaining: 00:00:00
iteration 1501/1570
total time of iteration: 00:00:10
Image.max = 1.149101
Image.min = -0.122896
finished running
```

– **With parallelization:**

Ben

```
doing iteration 1540
doing iteration 1550
doing iteration 1560
doing iteration 1570
Shutting down processor threads
Main: Waiting for join
Main: Join successful
contmaxflow.c - Exiting
total time of iteration: 00:00:08
Image.max = 1.079079
Image.min = -0.094653
finished running
```

UjoImro

```
iteration 1401/1570
estimated time remaining: 00:00:00
iteration 1501/1570
total time of iteration: 00:00:05
Image.max = 1.147976
Image.min = -0.125701
finished running
```

- **The Space**
  The test were performed on the 1.86GHz dual core lab machine. The codes are compiled with icc with all the fancy optimization flags, there are. Both are parallelized.

  Ben
  ```
  doing iteration 1890
  doing iteration 1900
  doing iteration 1910
  doing iteration 1920
  Shutting down processor threads
  Main: Waiting for join
  Main: Join successful
  contmaxflow.c - Exiting
  total time of iteration: 00:01:11
  -0.000228 < Image < 1.010605
  finished running
  ```

  UjoImro
  ```
  iteration 1701/1920
  estimated time remaining: 00:00:08
  iteration 1801/1920
  estimated time remaining: 00:00:01
  iteration 1901/1920
  total time of iteration: 00:01:19
  -0.000380 < Image < 1.011574
  finished running
  ```

  In the previous section we have compared the softwares to each other. Now let's see how it performs compared to the original. The original software run on this sample for over a hundred minutes.

  ```
  estimated time remaining: 00:00:02
  iteration 4901/5000
  ```

```
estimated time remaining: 00:00:00
iteration 5001/5000
total time of iteration: 00:03:24
-0.000527 < Image < 1.010330
finished running
```

The new optimized of uiFibres managed to outperform contmaxflow in 2D in both parallel and serial compilations. It has unfortunately lost this gain 3D however it always stayed within 11% compared to contmaxflow. It has an overall **35 times** performance gain compared to the trivial implementation.

Segmentation and image analysis in nanotomography

In this part we the segmentation methods and strategies developed in the scope of this internship. We will present current cases, where even the optimum framework have to be adjusted in order to achieve correct segmentation.

5. Characteristic of the input images and the interest of the segmentation

The images where captured with a Transmission Electron Microscope (TEM) in nanotomographic mode. This modality is similar in principle to standard X-Ray tomography. The images are reconstructed from a movie of the objects. The object is turned around its axe like on fig 6. and full 2D attenuation pattern is recorded at each angle. The sample is injected with high intensity particles before the taping, so the exact position of the bar can determined. However as the bar can only be turned around with about 140 degrees[18], there are some parts of the object which are not present on the movie.

The result of the microscopy are quasi-three-dimensional images. This means, that near object poles, the signal to noise ratio becomes very low, due to the incomplete tomography reconstruction.

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 surfaces[19].

We require a segmentation method with objective optimization criteria, few arbitrary parameters, that is little sensitive to noise and able to optionally interpolate missing data due to the incomplete tomography procedure. In this case we can greatly benefit from the properties of minimum surfaces. They features an optimum framework, ensuring noise robustness and accurate contour placement as well as useful interpolation and topology preservation features. It is particularly effective in the case of weak gradients and low signal-to-noise ratio images, which is the case here. In the next section we briefly present the possibilities in the choices of the metric.

6. The choice of the metric

In the framework we are seeking the optimal curve (the curve which has the optimal integral) around the source. However on point wise sources $E[s] = \oint_s \mathrm{d}G = \oint_s g\mathrm{d}x \leq \text{surf}(s) \cdot \max_s g \rightarrow 0$ if we take smaller and smaller surfaces around S. Evidently if we take a small surface around S, it will have small cost, but it is

[18]due to the thickness of the bar

[19]In some of our results they are presented separately in different colors.

not on the contour of the object. This difficulty can be fought two different ways. The first possibility is that we take a non-zero-measure S. The boundary ∂S will have non-zero cost, so we can find the contours if they are smaller then the cost of the boundary. The second possibility is to use some weighting. For example in a constant image L, lets define g as:

$$g_I(P) := \frac{1}{\mathrm{d}(P,S)\cdot(1+\nabla L)} \tag{26}$$

Here $d(A,B)$ denotes the distance of A and B. Now let us look at the circles with ray r around the pointwise S:

$$\oint\limits_{|\mathbf{x}|=r} g_I(\mathbf{x})\,\mathrm{d}\mathbf{x} = \oint\limits_0^{2\pi} \frac{r}{\underbrace{\mathrm{d}\left(\begin{bmatrix} r\sin(t) \\ r\cos(t)\end{bmatrix}, S\right)}_{r}\cdot(1+\nabla L)}\,\mathrm{d}t = \oint\limits_0^{2\pi} \frac{\mathrm{d}t}{1+\nabla L} \tag{27}$$

If L is constant than (27) gives 2π for all the circles around S. It can also be proven, that these circles are the minimum cost curves of this metric. If ∇I is different from 0 in a point[20] A, than the "cheapest" curve will be the circle which contains A. The other direction is also true: if an image I is constant between the parts of the contour, than on these parts the solution will be interpolated with a circle. This approach is used in section 7.

We can extend this thinking, 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.

One the conveniences of this differential geometric approach is when only parts of the contour are known. These are parts of the image where the metric is low. In this case, this approach interpolates the known area with patches of minimal surfaces (in the geometric sense).

This approach still naturally "prefers" the high frequency places before the curves of D. So it has a limited usage if the noise is at level of the contours, like in section 7. Also there are cases where the set of curves is more difficult to define, like in section 8.3.

The remaining challenges are now to exhibit a relevant metric to our problem, and to choose sources and sinks. In the next few section we present the ways we have developed to extend the capabilities of the segmentation in the cases of weak gradient or/and high noise.

[20] for example there is one point in I

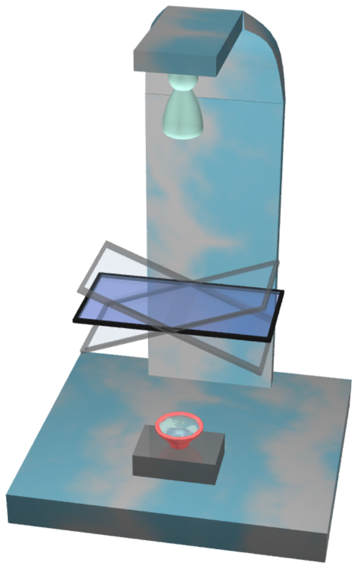


Figure 6: The model of the microscop. The sample bar is turned around its axe in approximately 140 degrees. The captured movie is then reconstructed into a quasi-3D image

7. Segmentation of 3D Nano-Scale Polystyrene Beads

In this project, we were interested in segmenting 3D images of nano-scale precursors of polystyrene beads. The material consisting of polystyrene and silica beads embedded in a substrate. These roughly spherical objects of size range 100-300 nm were nucleated around an existing silica bead. The aim was to find the size and location of all the polystyrene beads with respect to the silica one.

In order to detect circles from 2D slices of the image volume, the Hough circle transform was used. The original Hough transform [Hou62] and its derivatives have been largely applied and recognized as a robust technique [IK88].

For the current image, the 2D Hough circle transform was used in order to detect the circles on each fully reconstructed (horizontal) 2D slice. On such slices, insufficiently reconstructed bead poles appear very dim, while well-reconstructed bead slices near the equator appear well separated from the background. The method succeeded in localizing circle centers and radii even from incomplete initial circles. As a result, a number of arcs with one center and different radii were produced per circle. Circles were reconstructed by means of the radii of all the arcs associated with one center.

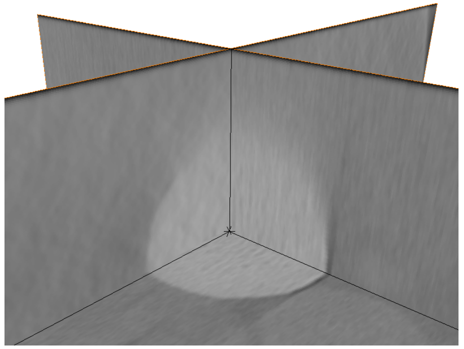


Figure 7: Beads image. Cross section of the fully (horizontal) and partly (vertical) reconstructed image planes.

Since a circle center is obtained for each bead on every slice where the method succeeded, in the 3D data all detected centers are vertically aligned. To locate the center of each bead with a good approximation, we selected the center with the largest radius. These centers have been used as the source sets, for the minimum surfaces. However in this case the noise near the poles becomes very close to the signal level (fig 7.). This means that the minimum surfaces will be stopped by the global noise. **In this case the key observation was, that the reconstruction created noise show parallelism with the axes.** So in this case we have used a modified metric. We used a separable spline-interpolated gradient, available in [Fou] in direction from the source to the point and the sphere weighting introduced in section 6. The formal definition follows: For a one-dimensional vector $\mathbf{x}$, its continuous spline interpolation is denoted with $f_{\mathbf{x}}$. For a 3D image I, the three axial vectors that contain the point P are denoted with $\mathbf{x}_P$, $\mathbf{y}_P$ and $\mathbf{z}_P$ respectively. The gradient of image I

in point P is defined by $\nabla I_P = \begin{pmatrix} \partial f_{\mathbf{x}_P} \\ \partial f_{\mathbf{y}_P} \\ \partial f_{\mathbf{z}_P} \end{pmatrix}$. To calculate the directional gradient in point P we use the direction vector $\mathbf{c} := \overline{\mathrm{CP}}$. The final measure function used in this case is

$$g(P) = \frac{1}{\underbrace{\mathrm{d}^2(P,S)}_{*} \cdot (1 + \underbrace{\mathbf{c} \cdot \nabla I}_{**})} \tag{28}$$

Here (**) is the gradient in direction of **c** and (*) is the square of the distance from the source. The square is needed because the surface of the square is quadratic to the ray. This measure filters the artifacts in the direction z and closer we are to the direction, more we ignore the noise. The result of the segmentation is presented in fig 14.

8. Nanoparticle transport across phospholipid membrane

Nanoparticle transport across cell membrane is important in the development of drug delivery systems, as well as in the question of nanoparticle poisoning. We know that hydrophilic nanoparticles 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.

Nanoparticles represent a growing direct interest for the nanotechnology, biotechnology or medicine, but there is also an indirect interest in the health risk associated with their use [NXML06]. In some cases the nanoparticles are poisonous, and it requires further research to determine, how can we predict these situations. Among the models, which could predict the nanoparticle toxicity, those concerning its transport through the wall of the cell appear to be the most promising. Lots of experimental and theoretical studies have focused on this issue. They identified crucial parameters that could affect the particle behavior, and to determine if they succeed in transpassing the membrane. These conditions, like size, chemical surface of nanoparticles and lipid membrane composition.

It was generally believed, that the particles did not enter into mammalian cell by endocytosis[21]. As evidence [GRRS+05] and [MSB+06] argumented with the entry of ultra fine particles into the red blood cells and cyt-D blocked macrophages[22]. Both of these cells are known for their lack of endocytotic capabilities.

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

[22] white blood cells, that absorb material foreign to the body (bacteria, etc)

However, [SM07] revealed[23], that in some cases the molecules did not pass through the membranes as expected. This suggests that the nanoparticle transport requires an interaction with the membrane. Unlike the nanoparticles larger than 30 nm, these 20 nm particles could not “break into” the membrane.

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

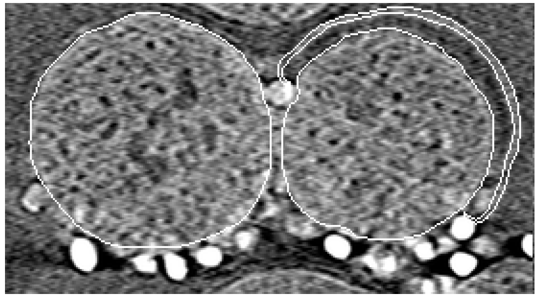


Figure 8: Summary of segmentation results

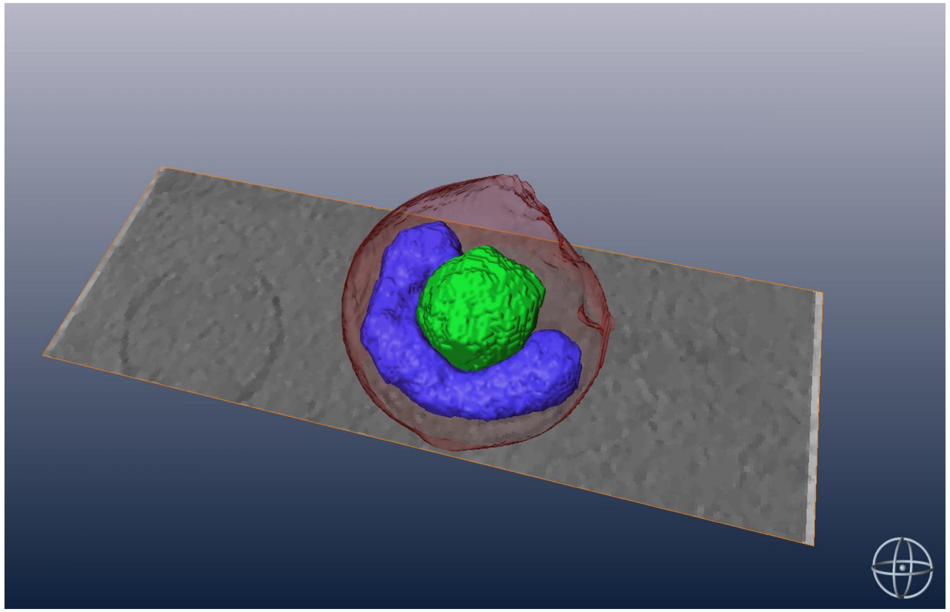
The result corresponds with [VS07], where the data provided by AFM[24] reveals a similar nanoparticle size effect. They showed that 20-140 nm silica nanoparticles put on polished silicon wafers were wrapped by SLB[25], while smaller particles were not. Although their results and their interpretations were mainly focused on SLB’s size and shape, this is also an evidence, that there is a threshold of nanoparticle size, which depends on the adhesion and the membrane curvature balance.

8.1. GoldSI

In this case we used the Canny-Deriche gradient operator described in [Der90][26]. In some cases the form can not be as easily defined as in the case of the beads. For example in fig 9. the phospolipide membrane (denoted with blue color) part has a more complex form, which is not trivial to generally define as a preferred surface. **In this case[27] we can achieve a good segmentation with a more complex source**. A simple choice of a bigger source would however need more knowledge about the internal characteristic or it would be a trade-off with objectivity.

For this we have developed methods which can stretch the source based on semi-local characteristics, namely the property, that in the direction of the equator, we have a good information ratio. We use reli-

[23] in a study made with gold molecules and a liposome that mimics the biological membrane
[24] Atomic Force Microscope
[25] Supported Lipid Bilayer
[26] available in [Cou]
[27] as the noise is still maximum around a point wise source



Default view

View 1

View 2

Figure 9: Nanoparticle wrapped around with a lipid membrane

able and easily segmented area found in some relatively noise-free 2D slices of our images. We topologically connect these segmentations to form a 3D source which is much larger than a point, but still derived automatically from the image data.

In the case of the membrane (red) this approach gives correct results directly. In other cases we use an iterative approach. We do the distance transform of the source and we set the iso-surfaces to be the preferred surfaces. The result of this segmentation is presented in fig 9.

8.2. Tapaus

In addition to the problems and strategies described above in fig 1. we have faced an other artifact of nanotomography[28]. The standard function of image reconstruction forces low-contrast areas around high contrast areas. In certain images there is a discontinuity in the membrane, even though we know from the physical properties, that the membrane is continuous. In this case we have segmented the visible parts of the membrane, then we wanted to cut the shell in two parts. We selected a source in the part we wanted to keep, and a sink in tho other part. The parts should be separated by the markers and the cut should be made on the smallest possible surface. This problem is also solvable with minimum surfaces. For the measure we use the internal distance transform of the source and as the result we get the good part of the shell.

8.3. Autap

In the case of complex objects it is particularly difficult to define good sources. In the case of fig 11. we defined the source of the internal space from the other objects, already segmented. As the particle moved into the cell the wall of the cell did not break. In fact it have wrapped the particle. The model is presented on fig 10. However at the point of the entrance the membrane became discontinuous due to the artifact of the contrast. In this case we have dilated the ball to a level, so it would get outside the and we used the internal area between the particle and the wall as a source to the minimum surfaces. The result (fig 11.) is the actual wall of the cell. The wall is continuous and wraps the particle.

After the segmentation we provided measurements of the segmented surface to provide correspondence with the physical model. With the measure of the curvature, we could provide evidence that the segmentation of the particle meets the electrostatic properties of the phospolipide material. The measure of the curvature is discussed in the next section.

[28]Transmission Electron Microscopy

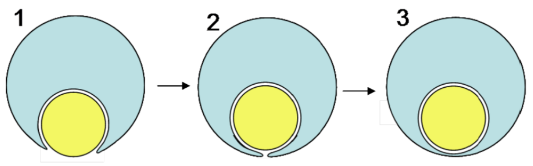
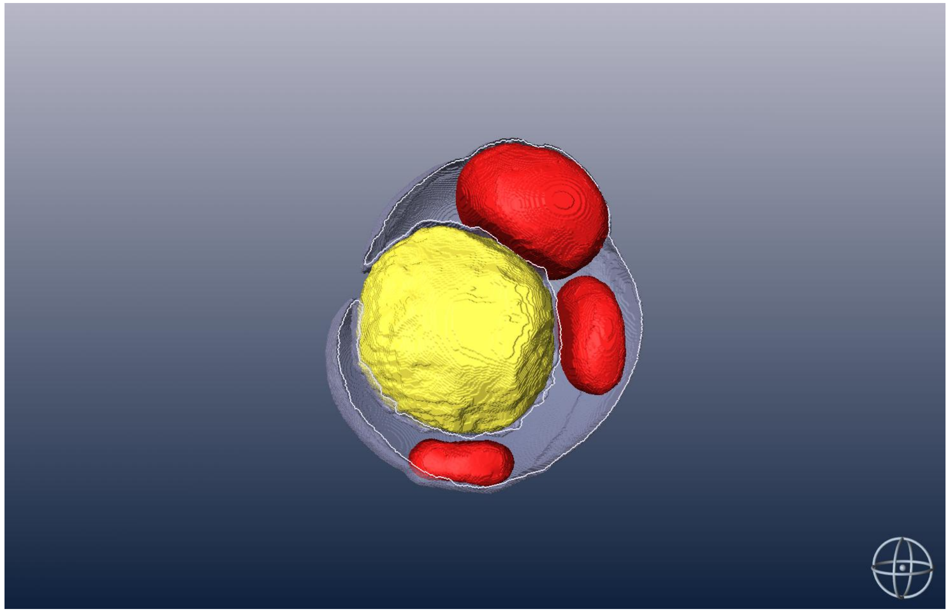


Figure 10: Nanoparticle entering a liposome



Default view
View 1

Figure 11: Mechanism of entering the silica nanoparticle into a liposome.

◀◀ ▶▶
◀ ▶
◀ Doc Doc ▶
Back Close

8.4. Curvature estimation

Curvature estimations were provided in some key areas of the image, for instance near points of contact between nano-particles and the membrane. 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 (the Hessian tensor). Curvature estimation is also possible from implicit surface representations [Gol05], however in our case we found that the precision of these methods was not good enough due to discretization. A scale must be chose at which to estimate the curvature and appropriate smoothing must be applied with some caveat, in particular regarding 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 media axis representation of our segmented result S [Blu61] and found the extremities of this representation. The medial axis is the locus of the centers of maximal disks (2D) or spheres (3D) included in S. (Maximal spheres are such that no sphere 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 [LVG80, ZC05]. Skeleton extremities can be detected using local configurations (e.g. points with a pre-determined number of connected neighbors). With the computed skeleton, a robust estimation of the curvature near skeleton extremities is given by the radius of the disk that is centered at a detected extremity and tangent to the nano-particle or to the external membrane, as illustrated in Fig 12. The curvature can be seen in Fig. 13 on our image data.

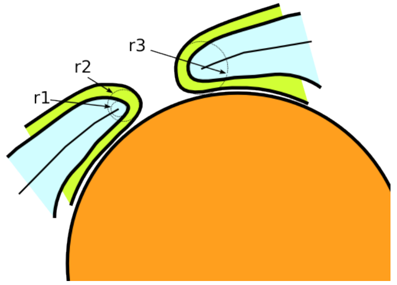
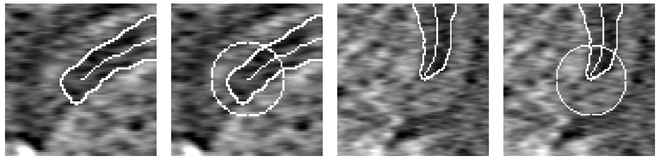


Figure 12: Curvature estimation.

8.5. Results

Our segmentation results are summarized on Figures 9, 11 and 1. Fig 8. shows 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 the places of interest. The median curvature radii measured are summarized in table 8.5, expressed in pixel dimensions.



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

Figure 13: The estimated curvatures

image	curvature	resolution
13(a),13(b)	18	552x311x305
13(c),13(d)	21	552x311x305
12(R_3)	15.0[1]	676x726x438
12(R_2)	13.5[1]	676x726x438

Table 1: The measured curvature radii

Conclusion and future work

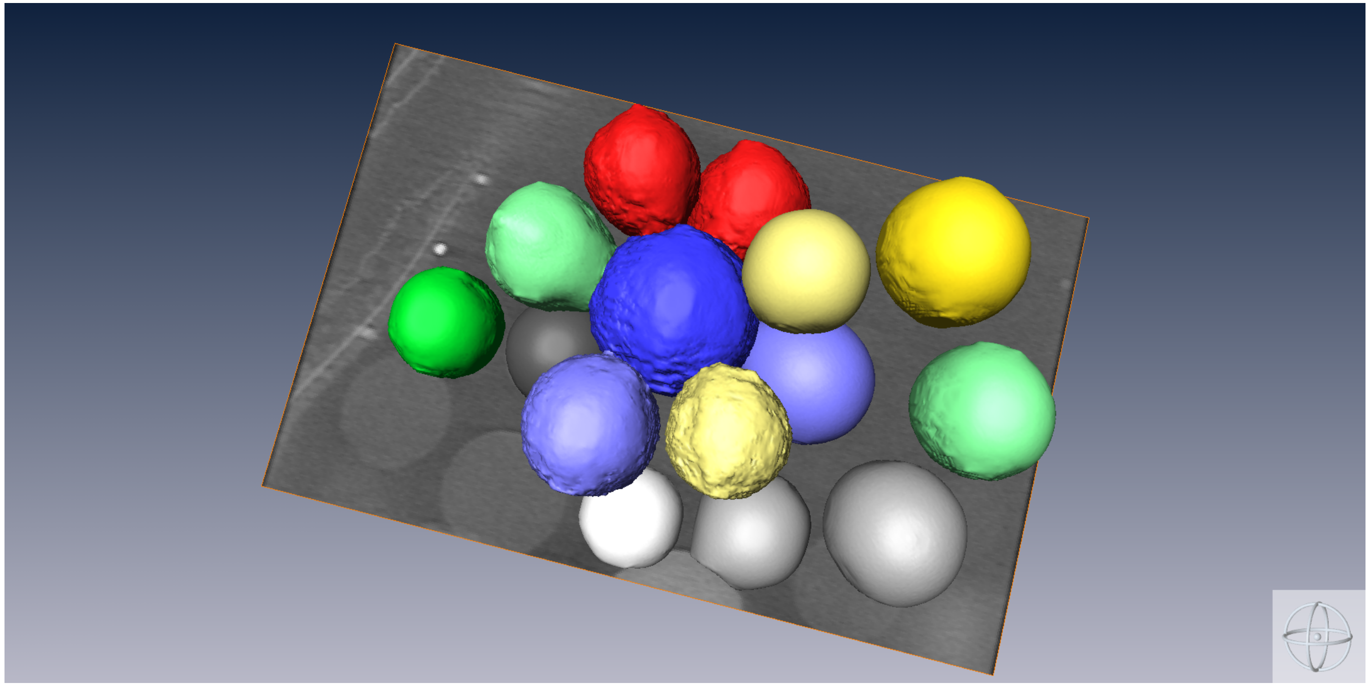
In the case of this work, he have re-implemented the continous maxflow algorithm. The implementation is optimized for parallel computing, and is prepared for the computer architectures of the near future. Unlike the preceeding implementation it is a free software and will be publicly avaible in the future.

We have presented some application of minimum surfaces in segmentation of 3D images. We have seen that a global optimization framework can be useful in several application. In the cases, where the noise represents a limit to this framework we have developed strategies to extend the framework with more sophisticated choice of source or advanced measures. These techniques were mainly applied to provide evidence to physical conjectures.

Our work supported the study of the toxic effects of nanoparticles, providing evidence of the case when the nanoparticle crosses the lipid barrier. This study can be used for the prediction of the toxicity of the nanoparticles.

[1] median of the measurements on several slices

◀◀ ▶▶
◀ ▶
◀ Doc Doc ▶
Back Close



Default view

View 1

View 2

View 3

Figure 14: Result of the segmentation. The effect of interest is the approach of the particles to the central (dark blue) particle

This is a good exemple of how certain fields of research intersect, as for example the study of the nanoparticle transport can create a need of further understanding the computer architectures.

In the future, we would like explore the capabilities in minimum surfaces. Our main interests are questions of the convergence of the flow, segmentation automatization and extension of the optimized functional.

References

[AT06] Ben Appleton and Hugues Talbot. Globally minimal surfaces by continuous maximal flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):106–118, 2006. 10, 11

[BF07] David G. Barnes and Christopher J. Fluke. Incorporating interactive 3-dimensional graphics in astronomy research papers, 2007. 6

[BK03] Yuri Boykov and Vladimir Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *International Conference on Computer Vision*, pages 26–33, Nice, France, October 2003. 9

[Blu61] H. Blum. An associative machine for dealing with the visual field and some of its biological implications. In *Biological Prototypes and synthetic systems*, volume 1, pages 244–260. 2nd Annual Bionics Symposium, Cornell Univ., E. E. Bernard and M. R. Kare eds., Plenum Press, New-York, 1961. 40

[CKS97] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79, 1997. 9, 10

[Cou] Michel Couprie. Pink image processing library. ESIEE, Paris. 36

[Der90] R. Deriche. Fast algorithms for low-level vision. *IEEE Transactions on PAMI*, 12(1):78–87, 1990. 36

[FB08] Christopher J. Fluke and David G. Barnes. The interactive astronomy textbook. *The Astronomy Education Review*, 7(1), 2008. 6

[FF62] J. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962. 9, 10, 11

[Fou] Free Software Foundation. Gnu scientific library. `http://www.gnu.org/software/gsl/`. 34

[Gol05] Ron Goldman. Curvature formulas for implicit curves and surfaces. *Comput. Aided Geom. Des.*, 22(7):632–658, 2005. 40

[Gra06] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006. 7

[GRRS$^+$05] M. Geiser, N. Rothen-Rutishauser, B. andK app, S. Schürch, W. Kreyling, H. Schulz, Semmler M, 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.*, (113):1555–1560, 2005. 35

[Hou62] P.V.C. Hough. Methods and means for recognizing complex patterns. US Patent 3069654, 1962. 34

[IK88] J. Illingworth and J. Kittler. A survey of the hough transform. *Comput. Vision Graph. Image Process.*, 44(1):87–116, 1988. Important reference. 34

[KDH$^+$05] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM J. RES. and DEV.*, 49(4/5):589–604, July/September 2005. 15

[KEK04] Yan Kang, Klaus Engelke, and Willi A. Kalender. Interactive 3d editing tools for image segmentation. *Medical Image Analysis*, 8(1):35–46, March 2004. 7

[KWT88] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. *International Journal of Computer Vision*, 1:321–331, 1988. 10

[LVG80] S. Lobregt, P.W. Verbeek, and F.C.A Groen. Three-dimensional skeletonization: Principle and algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):75–77, January 1980. 40

[MFTM01] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *iccv*, 02:416, 2001. 7

[MSB$^+$06] Rothen-Rutishauser B. M., Schurch S., Haenni B., Kapp N., and Gehr P. Interaction of fine particles and nanoparticles with red blood cells visualized with advanced microscopic techniques. *Environ. Sci. Technol*, 40(14):4353–4359, 2006. 35

[NXML06] Andre Nel, Tian Xia, Lutz Mädler, and Ning Li. Toxic potential of materials at the nanolevel. *Science*, (311):622–627, 2006. 35

[Set99] J.A. Sethian. *Level set methods and fast marching methods*. Cambridge University Press, 1999. ISBN 0-521-64204-3. 10

[SM07] Banerji SK and Hayes MA. Examination of nonendocytotic bulk transport of nanoparticles across phospholipid membranes. *Langmuir*, 23(6):3305–3313, 2007. 35

[Str83] Gilbert Strang. Maximal flow through a domain. *Mathematical Programming*, 26:123–143, 1983. 9

[VS07] Ginzburg V. V. and Balijepalli S. Modeling the thermodynamics of the interaction of nanoparticles with cell membranes. *Nano Letters*, 7(12):3716, 3722 2007. 36

[Wik] the free enciklopedia Wikipedia. http://en.wikipedia.org/wiki/euler-lagrange. `Euler-Lagrange equation`. 48

[ZC05] R. Zrour and M. Couprie. Discrete bisector function and euclidean skeleton. In E. Andres and P. Damiand G., Lienhardt, editors, *Discrete Geometry for Computer Imagery: 12th International Conference, DGCI 2005*, volume 3429 of *LNCS*, pages 216–227, Poitiers, France, April 2005. Springer. 40

Annexes

How to use this document

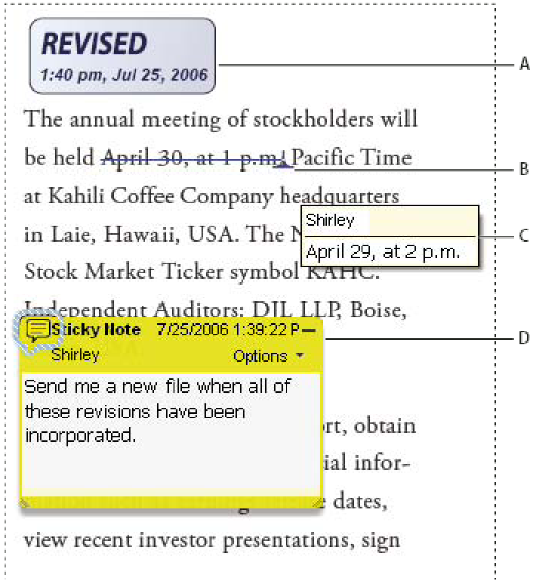
In this section we present some advantages of electronic publishing used in this document. The three main features are linking, commenting and 3D models. The introduction to their usage is presented in this annex. Most of the usage information is extracted from Adobe Acrobat Reader's manual.

Commenting

You use commenting and markup tools (View > Toolbars > Comment & Markup) to add **comments**. Comments are notes and drawings that communicate ideas or provide feedback for PDFs. You can type a text message using the Sticky Note tool, or you can use a drawing tool to add a line, circle, or other shape and then type a message in the associated pop-up note. Text-editing tools let you add editing marks to indicate changes you want in the source document. Most commenting and markup tools don't appear in the toolbar until you add them.

Most comments include two parts: the icon, or markup, that appears on the page, and the text message that appears in a pop-up note when you click or double-click the icon or place the pointer over the icon.

After you add a comment, it stays selected until you click elsewhere on the page. A selected comment is highlighted by a blue halo to help you find the markup on the page. A wireframe with selection handles appears so you can adjust the size and shape.



REVISED
1:40 pm, Jul 25, 2006

The annual meeting of stockholders will be held ~~April 30, at 1 p.m.~~ Pacific Time at Kahili Coffee Company headquarters in Laie, Hawaii, USA. The N

Shirley
April 29, at 2 p.m.

Stock Market Ticker symbol KAHC.
Independent Auditors: DIL LLP, Boise,

Sticky Note 7/25/2006 1:39:22 P
Shirley Options
Send me a new file when all of these revisions have been incorporated.

rt, obtain
ial infor-
dates,
view recent investor presentations, sign

Figure 15: Types of comments in a PDF. a) Stamp, b) Text edit, c) Comment rollover d) Sticky note

◄◄ ►►
◄ ►
◄ Doc Doc ►
Back Close

Links

In the document most of the references are linked. This means, that by clicking on them you jump to the point of the reference. To go back to the origin you can use the **back** button on each page. If you click on it, it takes you back to the previously displayed page.

Displaying 3D models

In Reader, you can view and interact with 3D content embedded in the PDF. For example, you can selectively hide and show parts of a 3D model, remove a cover to look inside, and turn parts around as if holding them in your hands.

3D content are marked with 3D and initially appear as a two-dimensional preview image. Clicking the 3D model with the **Hand** or **Select** tool enables (or activates) the model and opens the 3D toolbar.

3D models are composed of individual parts. You can use the **Model Tree** to hide or isolate parts, zoom in to parts, or make parts transparent.

The Euler-Lagrange equation[1]

The Euler-Lagrange equation or Lagrange's equation, developed by Leonhard Euler and Joseph-Louis Lagrange in the 1750s, is the major formula of the calculus of variations. It provides a way to solve for functions which extremize a given cost functional. It is widely used to solve optimization problems, and in conjunction with the action principle to calculate trajectories. It is analogous to the result from calculus that when a smooth function attains its extreme values its derivative goes to zero.

Theorem 1. *The Euler-Lagrange equation is an equation satisfied by a function f of a real parameter t which extremises the functional*

$$J = \int_a^b F(t, f(t), f'(t))\, \mathrm{d}t \tag{29}$$

where F is a given function

$$F : \mathbb{R} \times X \times Y \ni (t, x, y) \mapsto F(t, x, y) \in \mathbb{R} \tag{30}$$

with continuous first partial derivatives. Here $\mathbb{R}$ denotes the set of real numbers and f is an X-valued function on the reals

$$f : \mathbb{R} \ni t \mapsto f(t) \in X \tag{31}$$

whereas the derivative of f is defined as

$$f' : \mathbb{R} \ni t \mapsto f'(t) \in Y \tag{32}$$

so Y is the space of values of the derivative of f, i.e., $Y = TX$ (the space of tangents to X).
The Euler-Lagrange equation then is the ordinary differential equation

$$F_x(t, f(t), f'(t)) - \frac{\mathrm{d}t}{\mathrm{d}t} F_y(t, f(t), f'(t)) = 0. \tag{33}$$

*where F_x and F_y denote the partial derivatives of **F** with respect to the second and third argument, respectively.*

[1]The following annex is extracted from [Wik]. It is included just as a curiosity.