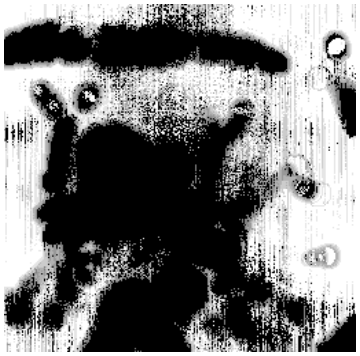
TECHNISCHE UNIVERSITÄT MÜNCHEN

**University Hospital Rechts der Isar:**
**Clinic of Orthopaedics and Traumatology**
**Department of Nuclear Medicine**

# Automated Segmentation and Registration of CT Coronary Blood Cavities based on Spherical Representations

Automatische Segmentierung und Registrierung von CT Herzblutgefäßen basierend auf sphärischer Repräsentierung



Interdisciplinary project in the minor subject
"theoretical medical science"

| | |
|---|---|
| Author: | Martin Schreiber |
| Advisor: | PD Dr. Rainer Burgkart |
| Supervisors: | Dipl. Phys. Dr. Stephan Nekolla |
| Submission Date: | May 15, 2009 |

# Contents

*The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music.*

Donald E. Knuth (1938 - )

# 1 Acknowledgement

First of all I would like to thank my advisor PD Dr. Rainer Burgkart for giving me the freedom to implement new methods and ideas, which were developed during the project. I also want to thank Dipl. Phys. Dr. Stephan Nekolla who provided me with the necessary informations about recent coronary segmentation and registration processes, the useful discussions with him and for the support with 3D CT datasets and the enhanced CT datasets for the 2D test cases used in this paper. Furthermore I would like to thank Dipl. Inf. Univ. Manuel Schröder and Dr. rer. nat. Heiko Gottschling for their support during the IDP.

I also want to thank Dipl. Math. Dr. rer. nat. Stefan Zimmer and PD Dr. Michael Bader for the useful discussions about applying the sphere filter, the mappings, etc. as well as Brian Jensen and Dipl. Math. Johannes Mittmann for proof reading this document.

The basic idea for the filter was to find characteristic lines for coronary datasets representing blood vessels (using the slope to find similar lines in the reference dataset). After figuring out that this is not enough information for a working matching, a graph was constructed as described in this paper. The next try was to compute a graph matching by comparing the similarity of slopes at the graph edges together with a matching of graph forkings or projecting forkings of one graph to an point on an edge of the other graph. This also failed due to the underestimated NP hard complexity. Finally a successful way was found using the affine transformations for the matching based on the graph nodes.

This project was created using only Free Open Source products using Debian Linux [10], the FFTW Library [8], libkdtree++ [7] and the DICOM toolkit [3].

The source of this project is released under GPL and can be found at [13].

*Artificial Intelligence: the art of making computers that behave like the ones in movies*

Bill Bulko

# 2 Introduction

Segmentation and registration of different medical datasets is a wide ranging area using different approaches like edge matchings, support vector machines, modification of the famous RANSAC algorithm, morphological operations or using an iterative method that modifies a geometric transformation mapping the datasets unto each other until an ideal is found. All methods use either direct voxel structures, which are represented by the dataset voxels itself or indirect data like gradients computed from voxel values or edge information (e. g. using a sobel edge detection).

## 2.1 Motivation

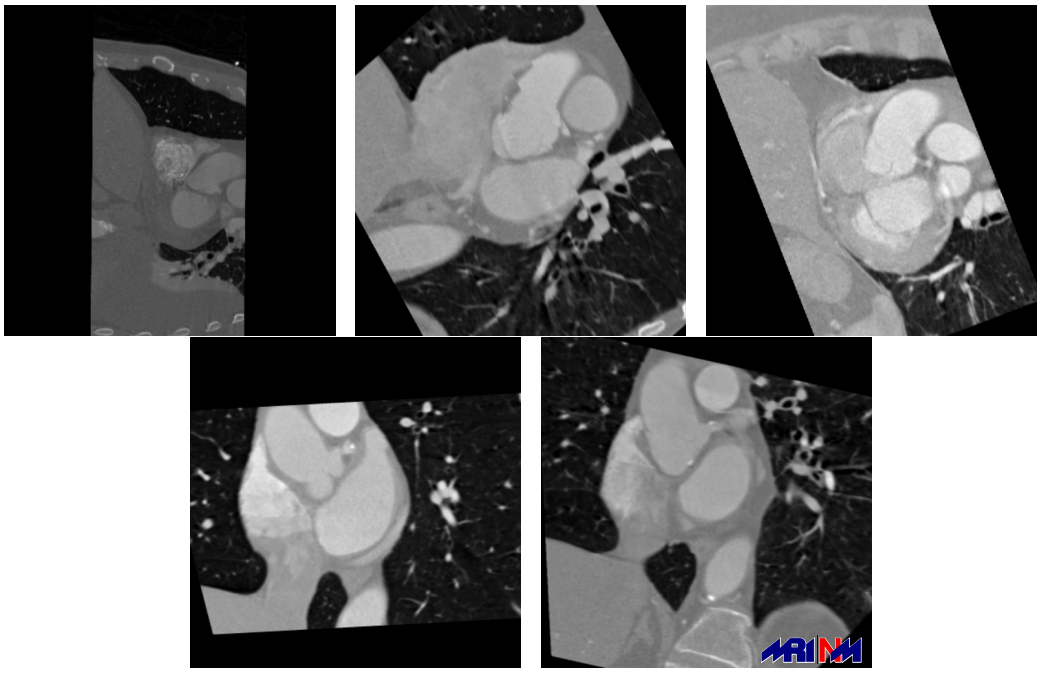


Figure 2.1: Example coronary CT slices

Current algorithms work on a high amount of matching elements from both datasets. Because of this a denser (more specific form of) information should be used, representing characteristic information of both datasets. The aim of this project was to match different CT coronary datasets (2D example slides are given in Figure 2.1) taken from different patients and to registrate those coronary CT scans with a reference coronary dataset. Usually this has to be done by an assistant or a doctor manually to improve the comparison of datasets e. g. finding an unhealthy increase of heart muscles, finding the transformation for PETs taken at the same time as the CT or speeding up the diagnosis by segmenting and registering known areas.
Blood cavities seem to be the most characteristic information if coronary CT datasets have to be registered. Searching for a good representation for blood vessels, spheres seem to be applicable because blood vessels in human bodies appear in a tubular form which can be represented by placing spheres along a line following the centers of the blood vessel. Possible problems like creation of the graph edges and disjointed blood vessels at bifurcations have to be solved to create a representative data. After this step the blood vessels can be represented by joining the volumes of the spheres along the line. Using this line of spheres, the blood vessels are represented by a very dense amount of graph edges and nodes. The graph nodes of both graphs are then used to create and use a matrix, mapping one dataset to the reference dataset. The important part is to find efficient algorithms for the computation of the underlying graph as well as a fast determination of the best matrix mapping one CT scan to the reference CT scan.

*I do not fear computers. I fear the lack of them.*

Isaac Asimov (1920 - 1992)

# 3 Filters and Methods

## 3.1 Overview

From a very general point of view, spheres have to be placed along the center line of blood vessels with the borders of the blood vessels touching the surface of the sphere. This motion of the sphere through the blood vessel can be imagined like the actions a chimney sweeper makes when cleaning a smoke stack (3.1), increasing the sphere radius if the sphere does not touch the vessel borders and shrinking the sphere if the movement gets stuck due to a lower vessel radius. Creating a line out of the sphere centers storing the sphere radii along the line would give the representation of the vessel.

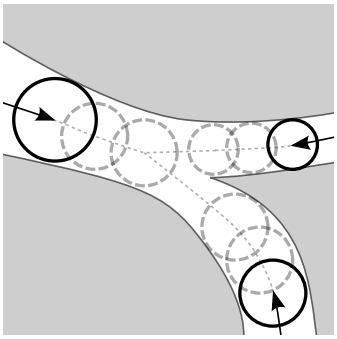


Figure 3.1: Simulated movement of a sphere within the blood vessel

From a computer scientist perspective, this simulation would take a lot of computation time. First of all, the movement itself has to be computed: Where should the sphere be moved to, what alternative positions exist, where should we start moving the sphere from? Secondly the program has to perform collision checking in addition to computing the acting force on the sphere on every movement in all spatial directions. If these computations have to be done just once for every final sphere, this would not be any problem but in reality the algorithms would have to try these collisions for many possible spatial movements. The third main problem is an extra degree of freedom necessary for the simulation: The growth and shrinking operations for the moving sphere makes the algorithm

too inefficient when using the direct implementation of the "chimney-sweeper" method.
Even if a fast method would exist, there still remain some unsolved problems like handling vessel bifurcations or complex blood cavities like those of the heart ventricles.
Instead of implementing the simulation directly, the operations can be represented by a stack of different basic filters handing back almost the same data with the advantage that the previously mentioned problems (bifurcations, non-tubic vessels, ...) can be solved fast and efficiently as well as that the implementation is very fast using advanced mathematics. The filters used include a spherical growing filter used for every point, a gradient filter representing the direction to the next neighboring sphere with a larger radius, a particle emission filter to find vessel centers and the graph construction as the final filter to compute the graph based on the particle distributions (and to solve the bifurcation problem using the gradient field).
Finally the transformation matrix is computed by determining the underlying matrix using the discrete graph nodes. The factorial computation time to compute all possible matrices is reduced by analyzing the decomposed matrices and using conditional criteria.

## 3.2 Spheres

The first sphere filter calculates the possible sphere positions computing the maximum allowed sphere radii at each voxel. This filter can be imagined by *expanding a sphere outwards at every voxel, fixing the sphere's center* to the recent voxel and to stop the growth if too many voxels covered by the sphere are out of a range given by the contrast medium injected into the blood. For each new voxel - representing the center of the local sphere - the maximum allowed sphere radius is stored to a new array $SphereData$.

### 3.2.1 Filtering in spatial domain

$$FlagData[pos] = \begin{cases} 1 & win_{min} \leq value[pos] \leq win_{max} \\ 0 & else \end{cases} \tag{3.1}$$

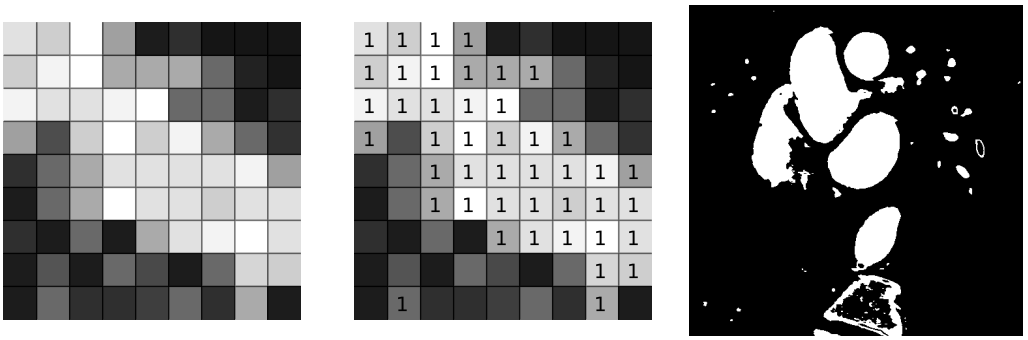


Figure 3.2: Left: Zoomed CT data for small vessel, Middle: flags set by thresholding, Right: thresholded image of example CT slice

Using sphere growing starting with a radius equal to 1 on a CT dataset would cause an early termination of the sphere growing algorithm for many points even if the blood vessel is much larger than described by the sphere radius. The reason is that CT scans underly a noise forcing the spherical growth with a small radius to abort very early. Therefore the spherical growth starts with a minimum radius $StartRadius$ to use an early termination by using the mismatch criterion given by (3.2). If this test fails the radius for the local sphere is set to 0. Apparently the $StartRadius$ specifies the smallest blood vessel which can be segmented by the sphere filter and depends strongly on the intensity range and the noise in the CT dataset.

$$\frac{\sum_{pos \in StartSphere} FlagData[pos]}{|\text{Voxels in Sphere}|} < maxMismatch \tag{3.2}$$

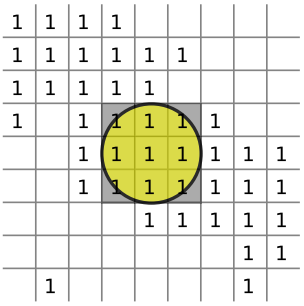


Figure 3.3: Applying sphere filter for start radius

If the first test for the sphere with radius $StartRadius$ was successful, the radius is successively incremented while checking the new surface values against the abortion criterion (3.3). The approach using the overall sphere volume for the abort criterion is avoided due to the case that small non-blood values may undergo the segmentation because they have a small relative amount on the segmentation criterion than using just the surface criterion. One example for this special case is the segmentation of the blood around cardiac valves (even if this is not important for registration of the given CT datasets) which are just visible by a volume with a thickness of just a few voxels in CT datasets. Starting the sphere growth near the cardiac valve could cause the sphere growth over the valve. Interpreting just the new surface voxels improves the spherical growth to stop earlier if the valve surface does not aim straight through the sphere surface. A more apparent reason is the spherical growth touching the vessel borders. Using only the surface values of the sphere gives a better gate to detect more accurate sphere radii which should represent only blood cavities.

$$\frac{\sum_{pos \in SphereSurface} FlagData[pos]}{|\text{Voxels on sphere surface}|} < maxMismatch \tag{3.3}$$

After the sphere growing is finished for each individual voxel, the maximum sphere radius is stored at that point (Figure 3.5). This data can be used to reconstruct the original blood vessels by joining all spheres to a large volume.

### 3.2.2 Sphere filter in the frequency domain

An implementation of this spherical filter like described above would cause a computation time of more than half an hour on recent quad core systems with the usage of a small $maxMismatch$ gate of 5 percent. This computation time increases more and more for larger blood vessels.
One of the main reasons to use spheres for segmentation is that they are rota-

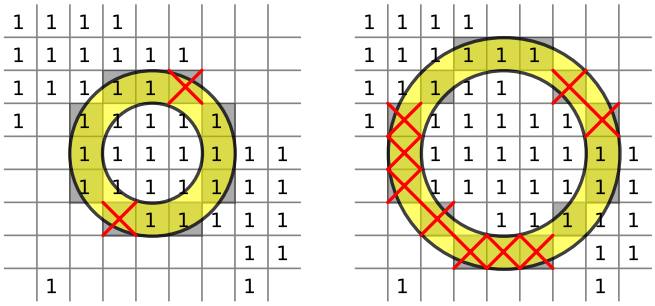


Figure 3.4: Left: Sphere growth for radius 3 with 2 mismatching voxels, Right: Mismatch value 9/16 exceeds the allowed rate $maxMismatch$

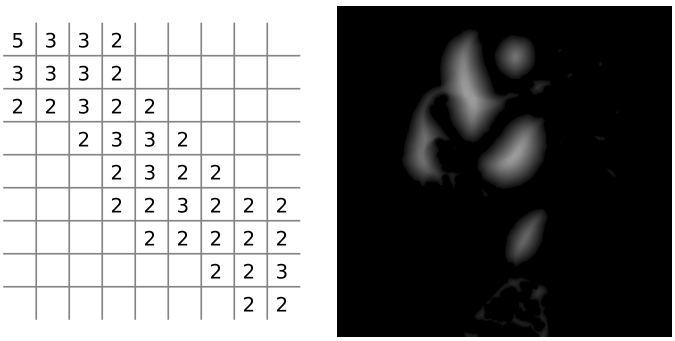


Figure 3.5: Left: Stored radius values after applying spherical filter, Right: Spherical dataset created by the spherical filter

tional invariant. Therefore it is not necessary to handle possible rotations while applying the sphere filter. This is also an advantage if we are only interested in the left-hand side of (3.3) for a constant sphere radius because with this prerequisite the filter for a constant radius can be applied by a convolution on the flag field using a kernel given by the points on the sphere surface. Therefore the sphere filter can also be applied by convolutions as shown in Figure 3.6 while incrementing the radius for the kernel (Algorithm 1).

For each convolution a test is done for every voxel whether any output data has been written yet ($SphereData[pos] >= 0$) followed by a test if the computed mismatching values force the sphere to stop growing ($ConvResult[pos] < maxMismatch$). If both tests are positive, the sphere growth for the recent call is stopped. The variable $writtenData$ is used to count the written values of the output data to stop the spherical growth if there are no growable spheres left ($writtenData \neq voxelCount$).

Obviously this algorithm seems to contain lot of overhead compared to the

**Algorithm 1** Sphere filter using convolution in spatial room

1: $sphereRadius = startRadius$; {current sphere radius}
2: $voxelCount = |\Omega|$; {number of voxels within CT dataset}
3: $writtenData = 0$; {written output pixels}
4: $maxRadius = MAX(\Omega_i)$; {maximum sphere radius}
5: $\forall pos \in \Omega : SphereData[pos] = -1$; {Initialize output data}
6:
7: **while** $writtenData \neq voxelCount$ and $sphereRadius < maxRadius$ **do**
8: $ConvResult = CONV(FlagData, ConvKernel_{sphereRadius})$ {do convolution in spatial room}
9: **for all** $pos \in \Omega$ **do**
10: **if** $SphereData[pos] >= 0$ **then**
11: **if** $ConvResult[pos] < maxMismatch$ **then**
12: $SphereData[pos] = sphereRadius$;
13: $writtenData++$;
14: **end if**
15: **end if**
16: **end for**
17: $sphereRadius++$;
18: **end while**

direct implementation. Performing the convolution in frequency domain instead of spatial domain reduces the computation time dramatically, especially on larger kernel sizes.

Using the Fast Fourier Transformation (FFT) on complex values would result in a periodical continuation of the *FlagData* array as shown in Figure 3.7. Therefore the convolution at the borders with the given kernel would include flag data from the opposite side of the domain $\Omega$ which does not represent the local neighborhood. To solve this problem the *FlagData* array can be enlarged on borders using e. g. a clamping "texture" border condition with the disadvantage of a growing computation time and memory usage depending on the size of the clamped border.

Using a modified version of the Fourier Transformation, assuming mirrored data at borders and real valued spatial data [11], the modified *Fast Fourier Transformation on real data* does not introduce anymore spatial data which does belong to the given local dataset. Because of the mirrored data at the borders the kernel has to just be initialized for $\frac{1}{4}$ of the cells in 2D as shown in Figure 3.8 and $\frac{1}{8}$ of the cells in 3D. The resulting algorithm (2) reduces the computation time from more than half an hour for high resolution CT scans to less than one second for each kernel radius on modern computer systems.

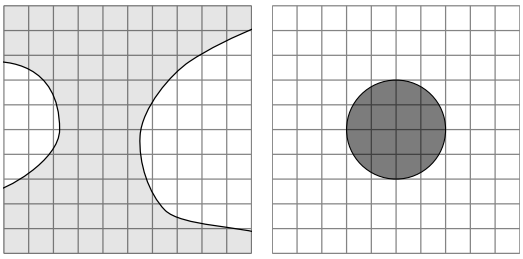


Figure 3.6: Sphere filter using convolution - Left: FlagData, Right: Kernel for first sphere

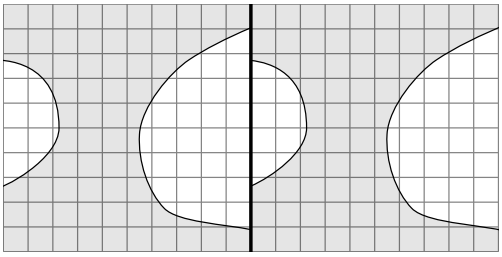


Figure 3.7: FFT based on periodical data

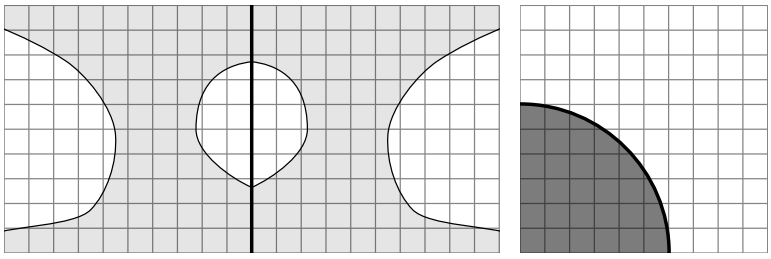


Figure 3.8: Left: Symmetric data - Right: Kernel

**Algorithm 2** Sphere filter using convolution in frequency domain

1: $sphereRadius = startRadius$; {current sphere radius}
2: $voxelCount = |\Omega|$; {number of voxels within CT dataset}
3: $writtenData = 0$; {written output pixels}
4: $maxRadius = MAX(\Omega_i)$; {maximum sphere radius}
5: $\forall pos \in \Omega : SphereData[pos] = -1$; {Initialize output data}
6: $FlagDataFreq = RFT(FlagData)$; {Real Fourier Transformation of $FlagData$}
7:
8: **while** $writtenData \neq voxelCount$ and $sphereRadius < maxRadius$ **do**
9: $ConvKernelFreq = RFT(ConvKernel_{sphereRadius})$; {RFT of kernel}
10: $ConvResultFreq = CONV\_FREQ(FlagDataFreq, ConvKernelFreq)$; {convolution in frequency domain by multiplication}
11: $ConvResult = IRFT(ConvResultFreq)$; {Inverse RFT}
12: **for all** $pos \in \Omega$ **do**
13: **if** $SphereData[pos] >= 0$ **then**
14: **if** $ConvResult[pos] < maxMismatch$ **then**
15: $SphereData[pos] = sphereRadius$;
16: $writtenData++$;
17: **end if**
18: **end if**
19: **end for**
20: $sphereRadius++$;
21: **end while**

## 3.3 Gradient

After the spherical filter is applied, the radius of the maximum sphere is stored at each voxel. This information can be used to get the direction *where a sphere would move if it is enlarged at a position near a vessel border*. The stored radii values are larger for the vessel centers, therefore neighboring sphere radii which are larger than the sphere centered at the current voxel represent the direction where a sphere will move if its size is increased as shown in Figure 3.9. To avoid searching the adjacent cells, the sphere radii can be seen as a scalar potential field $\Phi$ giving the direction to the vessel centers by the spatial derivative of the potential using central differences:

$$GradData[pos] = \frac{1}{2}\begin{pmatrix} SphData[pos+e_1] - SphData[pos-e_1] \\ SphData[pos+e_2] - SphData[pos-e_2] \\ SphData[pos+e_3] - SphData[pos-e_3] \end{pmatrix} \approx \nabla\Phi$$

with $e_1 = (1,0,0)^T, e_2 = (0,1,0)^T, e_3 = (0,0,1)^T$

An important value is the length of the gradient vectors: Those become less for vessel centers because the opposite potential values eliminate themselves due to the symmetric behavior of sphere radii at centers of tubes.

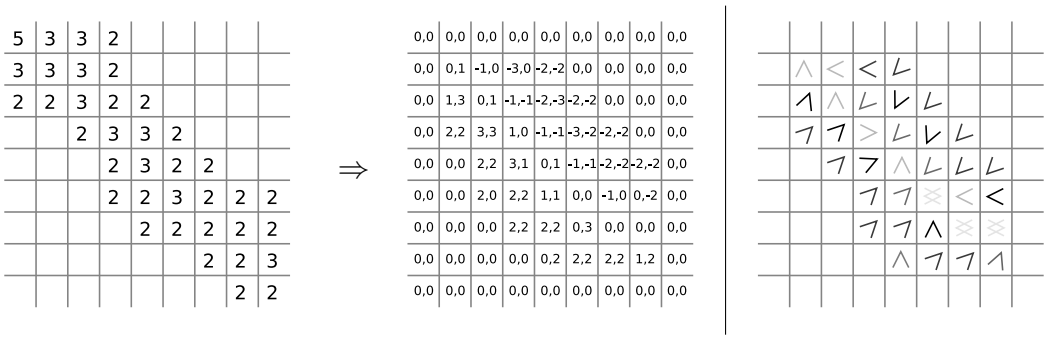



Figure 3.9: Left: Computing gradients from sphere radii using central differences, Right: Gradients represent vectors aiming towards vessel centers. The brighter an arrow is, the shorter is the length of the gradient vector.

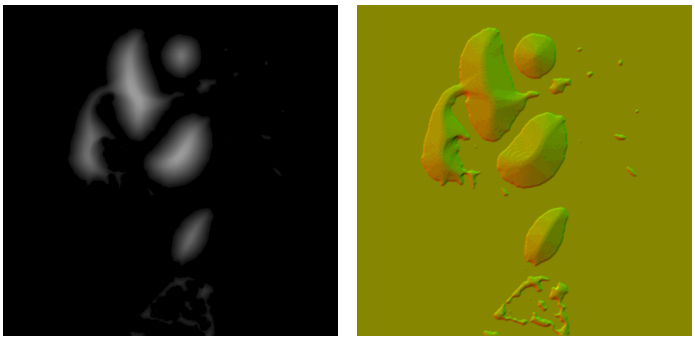


Figure 3.10: Gradients computed for given spherical data

## 3.4 Particles

Because we are interested in vessel centers, particles are injected at each possible position of the domain $\Omega$ when a radius larger than zero is stored at the corresponding voxel in the spherical data array. Each particle follows the gradient travelling a maximum of one cell per iteration until the length of the gradient is below a specified value *stopGradient*.

The second parameter *emitRadius* is used to avoid the injection of particles in small areas like the backbones segmented at the bottom of Figure 3.10. This permits an easy method to eliminate small structures which are not necessary for registration, thus improving the computational performance and avoiding the emission of particles at vessel borders. If the length of a gradient at the recent particle position is below the value *stopGradient*, the *particle position is stored in a dataset* together with the *sphere radius* at this voxel position.

Particles with the same position are eliminated easily by using a set for the storage. Using 1.0 for *stopGradient* as the criteria to stop particles at a respecive gradient with a length below that value results in the right particle distribution of Figure 3.11 for our 2D example. The particle filter is summarized in algorithm 3.

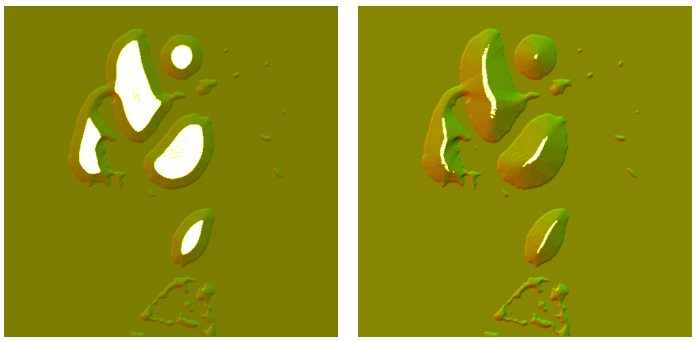


Figure 3.11: Particle emission with parameters *emitRadius* = 22, *stopGradient* = 1, Left: Particle distribution after the 1$^{st}$ iteration, Right: Particle distribution after applying the particle filter

**Algorithm 3** Particle filter

1: **for all** $pos \in \Omega$ **do**
2: $maxIterations = MAX(\Omega_i)$ {Safety limit to avoid particles iterating infinitely in a circular way}
3: $particlePos = pos$
4: **while** $maxIterations \geq 0$ **do**
5: $gradient = gradData[particlePos]$;
6: $length = |gradient|$;
7: **if** $length < stopGradient$ **then**
8: break;
9: **end if**
10: $particlePos = particlePos + gradient/length$;
11: $maxIterations - -$;
12: **end while**
13: **end for**

## 3.5 Graph

One of the most crucial parts is the graph construction, which reduces the particle representation to a graph (joining spheres centered at particles with the sphere radius stored at the particle). To reconstruct the blood vessels, spheres can be distributed along the edges interpolating the radius given by the two nodes which create the edge. By joining the volumes of the edge-distributed spheres we are able to reconstruct almost the original blood vessels assuming that the edges are always aligned in the centers of vessels and that the vessels have a circular cross section. This makes it possible to convert the particle representation to a graph based representation giving us much more information like bifurcations, alignments of the vessels and a structured view of the blood vessels which can also be used for advanced registration methods (e. g. using graph matching). An example of a desired graph construction is given in Figure 3.12. The graph construction is based on three main steps, which are in turn based on neighborhood assumptions to insert edges created from particles within a specific neighborhood.

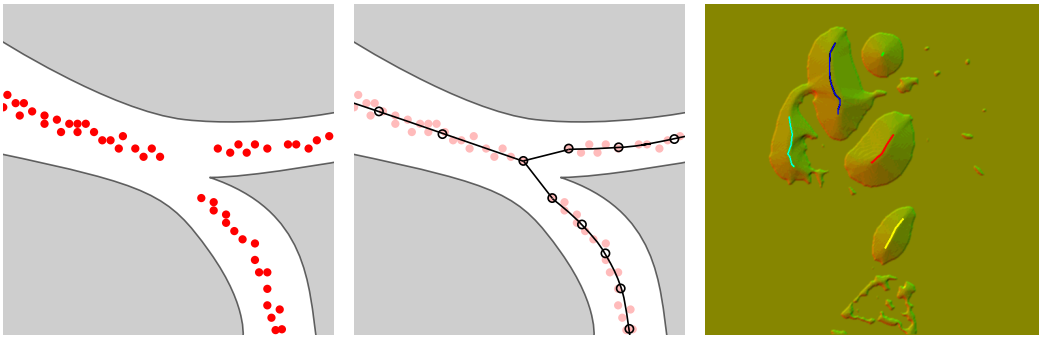


Figure 3.12: Left: Example particle distribution after particle filter, Middle: Constructed graph, Right: Graph construction on CT slice

The *first step* of the edge construction creates an edge based representation for the particles giving a set of striplines (disconnected graph $G$ with $degree(G) \leq 2$). Each particle is extended by an $usedFlag$ which is set initially to false to handle the information if the particle is already representated by an edge. The algorithm iterates as long as there is no particle left which is not yet represented by an edge. The reconstruction is based on the idea of drawing a line to connect particles. We start at the particle with an unset $usedFlag$ and the maximum sphere radius because this particle is probably the most important one centered in a large blood cavity. This particle is used as the node $initialNode$ for the new stripline. To insert a new edge to the graph a well fitting neighbored node has to be determined. To avoid creating edges that are too small, which would produce an undesired graph node density as well as edges that are too small (to avoid breaking our prerequisite for a representative edge) the nodes with an

unset $usedFlag$ are searched within a specific range $[min_{dist}, max_{dist}]$ as shown in Figure 3.13. To produce the sparsest graph-node representation, the particle within a specified range with the largest distance is taken as $nextNode$. If more particles fulfill this restriction, the second criterion is to use the particle with the maximum sphere radius as $nextNode$. After determination of this node all flags $usedFlag$ within the radius $max_{dist}$ are set to $true$ to avoid creating edges backwards if we continue extending the graph at $nextNode$.

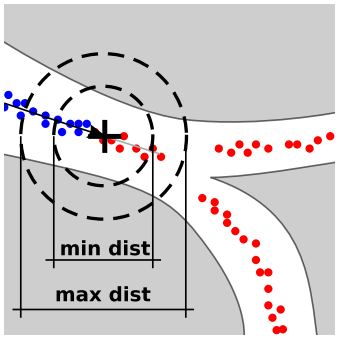



Figure 3.13: Searching neighbor nodes for graph construction

The *second step* handles the *probably existing edges in the opposite direction*: After creation of a stripline starting at the remaining particle with the largest sphere radius there may be particles which should be included into this stripline at the starting node. E. g. when the $initialNode$ lies in the center of a line. These particles would have a set $usedFlag$ which avoids the immediate creation of edges with this particles. Therefore the flag for the $initialNode$ is handled specially: Only the flags of nodes within the range of the $startNode$ and in the range of $nextNode$ are modified as given in Figure 3.14. This avoids walking backwards and makes it possible to extend the stripline to the opposite side by starting the edge creation for a second time at $startNode$.

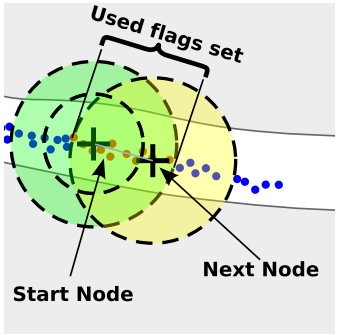



Figure 3.14: Handling of first strip node

The *last step* is necessary to connect the striplines at bifurcations: Because the gradient at bifurcations forces the particles to spread away, avoiding a connection of the striplines (Figure 3.12), those connections have to be specially handled. One method to create the connections is the emission of a particle near the corner nodes: A virtual particle is emitted starting at a small displacement aimed in the direction of the two ending stripline nodes (Figure 3.15). The particle then follows the gradient with the same parameters as it was done for the particle filter. If the particle stops, the neighborhood is searched for possible neighbored nodes, and if such nodes are found, the nearest one is taken as $neighboredNode$. With this the junctions of the corner nodes of the striplines and $neighboredNode$s can be found and those junctions are used to create a graph with a degree greater than 2, thus providing the final graph with a representation for the bifurcations by inserting respective junction edges.

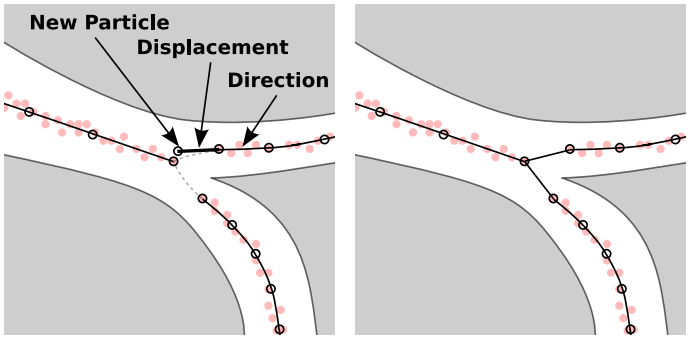



Figure 3.15: Connecting strips to grahps

The virtual particle emission handles the two main different cases of bifurcations: The case of an unsymmetrical vessel bifurcation with one large vessel and two smaller vessels with a particle distribution as shown in Figure 3.15 and the case of a large sphere at the bifurcation where the particles gather at the center point creating a "one node strip" of the sphere. The last one is also handled by the given algorithm because a particle which is emitted with a small displacement from an outgoing stripline would move to the center of the large sphere. Therefore this kind of bifurcation is represented correctly if every outgoing stripline emits such a particle connecting the corner nodes to the "one node strip" at the sphere's center.

Storing a few particles with a full grid would be a *waste of memory* and also the *computation time for nearest points* with $O(r^d)$ increases with the resolution $n^d$. To override this situation we use a *kd-tree* (Figure 3.16) which can access the particles efficiently reducing the runtime to $O(log\ N)$ on average (where N is the number of particles) because finding the nearest neighbors within a kd-tree is done by a depth-first-search.

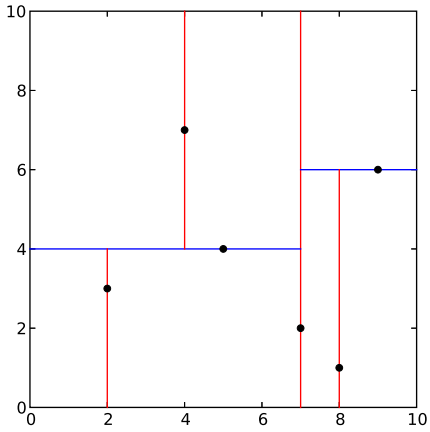



Figure 3.16: Storing and accessing particles using kd-trees, http://en.wikipedia.org/wiki/Image:Kdtree_2d.svg

## 3.6 Matching and Registration

To improve the comparsion of different CT scans the datasets should be automatically aligned, matching blood cavities like the example given in Figure 3.17. This enhances the quality of the medical care in addition to saving a lot of time for the Doctors, who would otherwise be forced to do the registration manually, becase a straight comparison of two unaligned CT scans is quite difficult. E. g. some unhealty increase of heart muscle may be overseen. Providing an automatic registration process which has to work within a minute would enhance the diagnostic quality using the automatically registrated CT scans, saves time for the registration process which is now obsolete and finally leaves more time for the diagnosis using the enhanced alignment of the CT datasets.

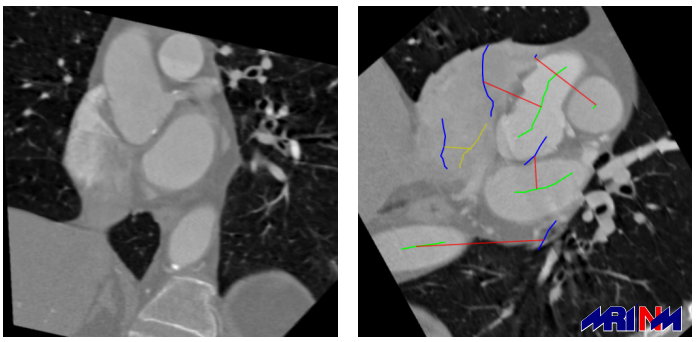


Figure 3.17: Matching of 2 different CT sets

### 3.6.1 Transformation Matrix

A transformation matrix $M$ has to be found, which maps the *nodes $P$ from the matching graph* to the *nodes $P'$ in the destination space* (3.4). The different instances of the matrix $M$ are found by taking *four marker points each from both graphs*. Assuming that the mapped points $P$ are exactly transformed to the points $O$ which belong to the graph in destination space we can set up 3 linear equations (3.5), (3.6) and (3.7) to obtain the coefficients for the matrix $M$ using the 8 marker points $P_{i,j,k,l}$ and $O_{i,j,k,l}$. Whenever the linear equation cannot be solved numerically stable the matrix is directly dropped because it probably would not give us an accurate transformation.

$$M \cdot P = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{pmatrix} \cdot P = P' \tag{3.4}$$

$$\begin{pmatrix} P_x^i & P_y^i & P_z^i & 1 \\ P_x^j & P_y^j & P_z^j & 1 \\ P_x^k & P_y^k & P_z^k & 1 \\ P_x^l & P_y^l & P_z^l & 1 \end{pmatrix} \begin{pmatrix} m_{1,1} \\ m_{1,2} \\ m_{1,3} \\ m_{1,4} \end{pmatrix} = \begin{pmatrix} O_x^i \\ O_x^j \\ O_x^k \\ O_x^l \end{pmatrix} \tag{3.5}$$

$$\begin{pmatrix} P_x^i & P_y^i & P_z^i & 1 \\ P_x^j & P_y^j & P_z^j & 1 \\ P_x^k & P_y^k & P_z^k & 1 \\ P_x^l & P_y^l & P_z^l & 1 \end{pmatrix} \begin{pmatrix} m_{2,1} \\ m_{2,2} \\ m_{2,3} \\ m_{2,4} \end{pmatrix} = \begin{pmatrix} O_y^i \\ O_y^j \\ O_y^k \\ O_y^l \end{pmatrix} \tag{3.6}$$

$$\begin{pmatrix} P_x^i & P_y^i & P_z^i & 1 \\ P_x^j & P_y^j & P_z^j & 1 \\ P_x^k & P_y^k & P_z^k & 1 \\ P_x^l & P_y^l & P_z^l & 1 \end{pmatrix} \begin{pmatrix} m_{3,1} \\ m_{3,2} \\ m_{3,3} \\ m_{3,4} \end{pmatrix} = \begin{pmatrix} O_z^i \\ O_z^j \\ O_z^k \\ O_z^l \end{pmatrix} \tag{3.7}$$

### 3.6.2 Matching differences

The transformation and matching quality is given by the "distance" between the reference graph after transformation using $M$ and the destination graph. To handle situations like missing nodes in both graphs, the final distance is computed by the sum of distances matching the mapped graph P' to the graph G and the other way around assuming that G is the mapped graph. Each of both distance computations is divided by the number of points in the mapped graph. The distance between two graphs itself is given by the sum of the square distances of the nodes $P'$ from the mapped graph to the nearest edges $E(O)$ of the other graph. A good estimation for the nearest edge is to determine the nearest graph node $O_{nearest}$ for every point $P'$. Then both neighbors of node $O_{nearest}$ are taken to compute the length of the vector perpendicular to the two outgoing edges of $O_{nearest}$ to the node $P'$ (Figure 3.18). The smallest distance to both edges is used as the minimal distance.

### 3.6.3 Matrix decomposition

Using the previously described criteria to decide which transformation matrix is optimal, there may exist some matrices which map the points in a totally wrong way like mirroring the data by obtaining a smaller computed minimum distance as the real matching would have. To give the algorithm a hint for elimination of a large amount of such cases the transformation matrix $M$ is decomposed into basic transformations which can be used to restrict the transformations. For instance it does not make sense to rotate the whole dataset around 180 degrees because the patient probably would not lie turned around in the CT scanner. A more obvious example is given by the shearing components. Because shearing does not preserve similarity the shearing components should be restricted to values near

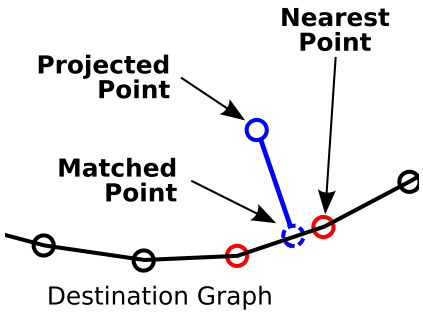



Figure 3.18: Computation of matching difference

zero. It is also impossible that the patients heart is mirrored between two CT scans.

To find the *basic transformation matrices* the transformation matrix $M$ is decomposed to the matrices

$$M = H \cdot S \cdot R_z \cdot R_x \cdot R_y \cdot T$$

Because we work with homogenous coordinates for points $P$ and $O$ and there are no projective components in our system the last row of matrix $M$ is already $(0, 0, 0, 1)$. Then the 12 degrees of freedom given by the matrix $M$ can be seen as parameters for *basic transformations*:

- 3 Translation components $t_x, t_y, t_z$ by matrix $T$
- 3 Rotation matrices $R_n$ around axis n: $R_x, R_y, R_z$
- 3 Scaling components $s_x, s_y, s_z$ with matrix $S$
- 3 Shearing components $h_1, h_2, h_3$ in the shearing matrix $H$

To understand the meaning of the transformations better we can imagine them as being performed by a doctor for registration knowing the parameters for the basic transformations: First the dataset is translated ($T$) for a coarse alignment of the CT scans, then it is rotated around y, x and finally the z axis ($R_y$, $R_x$, $R_z$). The last transformations are done by scaling ($S$) and shearing ($H$) of the dataset. The scaling can be used e. g. if the CT data was taken with different CT scanners having different resolutions to allow the algorithm an automatic scaling of the matching CT scan to the size of the destination data. The shearing matrix is the only one with shearing coefficients which should be near to zero even if similar datasets should be matched. Possible circumstances like different patient

placements or the beating of the heart are usually transformations without large shearing coefficients. If we want to obtain only similarity transformations we are also able to rebuilt the transformation matrix $M$ with the basic transformations omitting the shearing matrix $H$.

To decompose the matrix we try to sucessively eliminate matrix entries by splitting up the matrix into a matrix with more zero components and a basic transformation. If any step of the decomposition encounters *operations that indicate numerical instability* (like division numbers very close to zero) the matrix is immediately dropped to avoid unpredictable mappings.

### Translation decomposition

The last column of the matrix $M$ is decomposed by a translation matrix which sets the last column of the matrix $M$ to $(0,0,0,1)^T$. The decomposition of $M$ can be written as the new matrix $M'$ multiplied by the transformation components:

$$M = M' \cdot T = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Computing the rightmost column values for matrix $M$ by multiplying the last column from T with the corresponding rows of matrix $M'$ hands us the implicit solution for the translation matrix T.

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = \begin{pmatrix} m_{1,4} \\ m_{2,4} \\ m_{3,4} \end{pmatrix} \tag{3.8}$$

Because the components $m_{[1,3],[1,3]}$ remain unchanged due to the identity diagonal and the zero components of the transformation matrix the new matrix $M'$ is a copy of matrix $M$ with the last column set to $(0,0,0,1)^T$.

### Rotation decomposition

The Givens rotation [14] is used to eliminate the remaining entries different from zero below the diagonal using rotation matrices. The first rotation around the y axis is used to eliminate the entry $m'_{3,1}$ of the new matrix $M''$. The rotation matrix $R_y$ is given by

$$R_y = \begin{pmatrix} cos(\alpha) & 0 & -sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ sin(\alpha) & 0 & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To obtain the new matrix $M''$ with an entry $m''_{3,1}$ set to zero the matrix multiplication can be written reordered:

$$M' = M'' \cdot R_y \Longleftrightarrow M' \cdot R_y^{-1} = M''$$

The inverse rotation matrix $R_y^{-1}$ equals the matrix $R_y$ using the inverse angle. Assuming that $m''_{3,1}$ should be eliminated we obtain the corresponding rotation angle for the matrix $R_y$:

$$0 = m'_{3,1} cos(\alpha') - m'_{3,3} sin(\alpha') \Longleftrightarrow \alpha' = atan\left(\frac{m'_{3,1}}{m'_{3,3}}\right)$$

The final matrix $M''$ is computed by creating the rotation matrix $R_y^{-1}$ with the angle $\alpha'$ and multiplication of $M'$ with the rotation matrix. The rotation angle for the forward transformation is given by the angle $-\alpha'$.

The elimination of the two remaining components below the diagonal works with the same procedure using rotations around the x and z axes giving angles $\beta$ and $\gamma$.

### Scaling and Shearing decomposition

After the decompositions using translation and rotation matrices the remaining matrix $M^{(3)}$ has zero-components at the translation components and at the lower triangle of the matrix. Now we assume that the upper diagonal matrix can be eliminated by a shearing matrix with components for *shearings acting from y and z on the x components* and to the *y components from the z axis*. The final decomposition can be rewritten using the shearing and scaling matrix $S$:

$$M^{(3)} = H \cdot S = \begin{pmatrix} m^{(3)}_{1,1} & m^{(3)}_{1,2} & m^{(3)}_{1,3} & . \\ . & m^{(3)}_{2,2} & m^{(3)}_{2,3} & . \\ . & . & m^{(3)}_{3,3} & . \\ . & . & . & 1 \end{pmatrix} = \begin{pmatrix} 1 & h_1 & h_2 & . \\ . & 1 & h_3 & . \\ . & . & 1 & . \\ . & . & . & 1 \end{pmatrix} \begin{pmatrix} s_x & . & . & . \\ . & s_y & . & . \\ . & . & s_z & . \\ . & . & . & 1 \end{pmatrix}$$

This can be computed by reordering the single equations for the components of the matrix $M^{(3)}$:

$$s_x = m^{(3)}_{1,1} \qquad s_y = m^{(3)}_{2,2} \qquad s_z = m^{(3)}_{3,3}$$

$$\begin{aligned} h_1 \cdot s_y = m^{(3)}_{1,2} &\Longleftrightarrow h_1 = \frac{m^{(3)}_{1,2}}{s_y} \\ h_2 \cdot s_z = m^{(3)}_{1,3} &\Longleftrightarrow h_2 = \frac{m^{(3)}_{1,3}}{s_z} \\ h_3 \cdot s_z = m^{(3)}_{2,3} &\Longleftrightarrow h_3 = \frac{m^{(3)}_{2,3}}{s_z} \end{aligned}$$

Given all decomposition components the original matrix $M$ can be recomputed by the 12 degrees of freedom $(h_1, h_2, h_3, s_x, s_y, s_z, \alpha, \beta, \gamma, t_x, t_y, t_z)$ creating

the well understandable basic transformation matrices:

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & h_1 & h_2 & . \\ . & 1 & h_3 & . \\ . & . & 1 & . \\ . & . & . & 1 \end{pmatrix} \cdot \begin{pmatrix} s_x & . & . & . \\ . & s_y & . & . \\ . & . & s_z & . \\ . & . & . & 1 \end{pmatrix} \cdot R_z \cdot R_x \cdot R_y \cdot \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

### 3.6.4 Restrictions

So far the parameters for the basic transformations were determined. Now we use this information to check whether this matrix represents a meaningful transformation or if the matrix should be ignored in further processing. If the values are outside of an acceptable margin the transformation matrix $M$ is dropped, thus saving much computation time which would be used for the computation of the transformation quality.

The following parameters were determined empirically for determining valid matching matrices $M$:

- relative rotation angle: $MAX(|\alpha|, |\beta|, |\gamma|) < 45°$
- scale factor $s_x, s_y, s_z \in [0.7; 1.1]$
- shear factor $h_1, h_2, h_3 \in [-0.3; 0.3]$
- maximum relative translation: $MAX(|t_x|, |t_y|, |t_z|) < 128.0$

The parameters for the rotation and shearing matrix are independent of the resolution. Because the translation and scaling restriction parameters depend strongly on the resolution, they should be automatically adapted for different sizes of CT datasets.

### 3.6.5 Pretranslation

To create an automatic registration method with relative translation restrictions we pretranslate the centers of both datasets to the origin. Therefore a pretranslation matrix $M_t$ is used to map all points $P$ and $O$ to align both centers to the origin $(0, 0, 0)$ before doing any computations. E. g. for a dataset with the resolution $(256, 256, 200)$ the pretranslation matrix would be

$$P = \begin{pmatrix} 1 & . & . & -128 \\ . & 1 & . & -128 \\ . & . & 1 & -100 \\ . & . & . & 1 \end{pmatrix}$$

Matching datasets with different resolutions is accomplished by using an individual pretranslation matrix for every dataset. The matching algorithm as previously described is summarized in algorithm (4) in short form.

### 3.6.6 Reduce matching runtime

Using a direct implementation of the algorithm would lead to a runtime of

$$O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$$

because there are $\frac{N!}{(N-4)!}$ combinations to select 4 nodes from the graph $G^p$ and also $\frac{N!}{(N-4)!}$ combinations to select 4 nodes of graph $G^o$. Reducing the possible combinations using a monte carlo method should be avoided because there is always the possibility that the matching gets stuck in a local extremum.

#### Using far distant points

To find a better method we consider three different types of errors:

The *first error type* is introduced by the *discretization*. The algorithm uses a grid where every sphere center as well as the particles and the respective graph nodes are placed on the grid. To reconstruct the transformation matrix $M$ the algorithm takes four points of matching space as markers to transform them to the corresponding marker points in the destination space. Let us assume that this four points lie very close together describing a cube with a unit sidelength and that this marker points are transformed by the identity matrix to the four corresponding nodes in destination space. If a voxel in the matching CT dataset is modified slightly this difference may lead to a jump of one cube node in destination space to an adjacent gridpoint (caused by to the discretization). Creating a transformation matrix out of this jittered cube nodes would cause only a small distance between the mapped cube nodes $P'$ and the destination cube nodes $O$ but a transformation of a point which is far away from the cube using the matrix based on the jittered cube would cause a much larger distance. The reason is that the transformation properties of matching space are described by 4 points lying close together. Jittering of the nodes on the cube produces small distance errors for points within the cube but large distance errors for points with a large distance to the cube. Thus using four points with a *large mutual distance* avoids this conditional problem.

The *second error type* is caused by *anatomical differences*. For instance assuming that the heart is moving with every beat and therefore the points do not lie in the same position, the same assumptions can be made as for the first error type: The points used to compute the transformation matrix should be restricted by a minimum distance.

The *third error type* is also circumvented by demanding a minimum distance for the four marker points: If the points are close together with a distance less than one on a grid and all possible node placements on this grid are allowed, the only possible rotation angles around the x, y and z axes would be $\{n \cdot \pi : n \in \mathbb{Z}\}$ radians. It is obviously that far distant points allow finer angles for rotations.

Another improvement used in the algorithm is the usage of a *maximum distance constraint between unmapped points*, because the probability is high that the distance between the marker point $P$ and the corresponding point $O$ is below some value (e. g. half the size of the domain).

As already mentioned the solution to the three problems is using only nodes with a given *minimum distance* to create a *better conditioned problem*. This avoids the computation of the transformation matrix for many points. *Omitting impossible node combinations* and *nodes producing a bad conditioned problem* decreases computation time strongly in the empirical tests.

### Selection of marker Points

The possible selections of the marker points without loss of accuracy was also improved with the following idea: The four loops iterating over the possible nodes of the reference graph should select the nodes incrementally according to their index if the graph nodes are stored in an array. If the indices of the nodes are ordered, no combination of the four marker points is selected twice for one matching. This enhancement is possible because the *necessary permutation to the corresponding marker nodes* of the other graph is done by selecting the nodes of the destination graph with the usual nested loops.

### Dynamic adaptive minimum Distance

A fixed *minimum distance* for the selection of marker points would produce unpredictable runtimes. If the algorithm is going to run automatically, this can lead to runtimes of one or more days in the worst case if there are too many graph nodes and those are aligned in a special order. Therefore a more careful marker point selection was implemented specifying the *maximum allowed points to use from one graph to construct the transformation matrix*.

Starting at a large value for the desired value *minimum distance* where only a few or no points fulfill this distance, the possible combinations are counted in every round. If the allowed amount of possible matching combinations was not reached, the *allowed minimum distance is decreased* and the possible combinations are counted again.

After the possible combinations exceeded the allowed value, we have the lower border for the *minimum distance* constraining the possible combination of the marker nodes of one graph. Computing these constraints for both graphs, an almost constant runtime can be achieved offering dynamic adaptibility to the graph resolution as well as specifying the accurateness of the matching for both graphs by a single parameter.

**Algorithm 4** Structure of matching algorithm with given restriction parameters

**Input:**
matching graph $G^p$ with nodes $P$ stored in vertex vector $V^p = (P_0, P_1, ..., P_{n_p})$ and edges $E^p = \{e|e \in (i,j), 0 \leq i < j \leq n_p - 1\}$
destination graph $G^o$ with nodes $O$ stored in vertex vector $V^o = (O_0, O_1, ..., O_{n_o})$ and edges $E^o = \{e|e \in (i,j), 0 \leq i < j \leq n_o - 1\}$

{**Pretranslation**}
**for all** $n \in [0; n_p]$ **do**
  $V^p[n] = M_t \cdot V^p[n]$
**end for**
**for all** $n \in [0; n_o]$ **do**
  $V^o[n] = M_t \cdot V^o[n]$
**end for**

**var** $minimum\_distance = infinite$;
**matrix** $best\_matrix = \left(0_{[0..4],[0..4]}\right)$;

**for all** $P^{i,j,k,l} \in G(P), |\{i,j,k,l\}| = 4$ **do**
  **for all** $O^{i,j,k,l} \in G(O), |\{i,j,k,l\}| = 4$ **do**
    {**Compute matching matrix** $M$}
    ...

    {**Decompose matrix** $M$}
    ...
    {**Check restrictions**}
    **if** $MAX(|t_x|, |t_y|, |t_z|) < 100.0$ and $MAX(|\alpha|, |\beta|, |\gamma|) < 90°$ **then**
      **if** $s_x, s_y, s_z \in [0.7; 1.3]$ and $h_1, h_2, h_3 \in [-0.3; 0.3]$ **then**
        {**Compute quality of matching**}
        distance = 0
        **for all** $P \in V^p$ **do**
          $P' = M \cdot P$
          $distance+ = DIST(P', \text{"nearest edge of graph } G^o\text{"})^2$;
        **end for**
        **if** $distance < minimum\_distance$ **then**
          {**Better matching was found**}
          $mimimum\_distance = distance$;
          $best\_matrix = M$;
        **end if**
      **end if**
    **end if**
  **end for**
**end for**

*Computer Science is no more about computers than astronomy is about telescopes.*

E. W. Dijkstra

# 4 Results

## 4.1 2D Matching

A slightly different implementation was used for testing purposes to match 2D scans and to test the general algorithm. Because we work on 2D images there are only short tubic structures in the given cross sections of the vessels giving small striplines and circular areas due to cross sections of tubes represented by a one-node-stripline. To get 2D slices out of the 3D datasets the slices with the same characteristic vessels were roughly preselected. A CT dataset with clear vessel borders was used to generate the reference graph given in Figure 4.1 for further matchings.

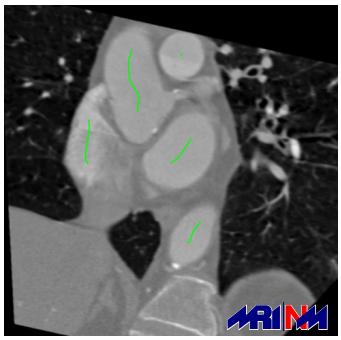


Figure 4.1: Reference CT slice and computed reference graph

The main difference between 2D and 3D matchings is that three points are enough to create the transformation matrix $M$. Applying the algorithm with the same parameters as for the creation of the reference graph leads to exact matchings of the 3 testing cases given in Figure 4.2. The matching distance was computed using only one point for every stripline given by the *midpoint of the stripline*. Even having this dense amount of information a successful matching could be obtained in the slices given by Figure 4.2.

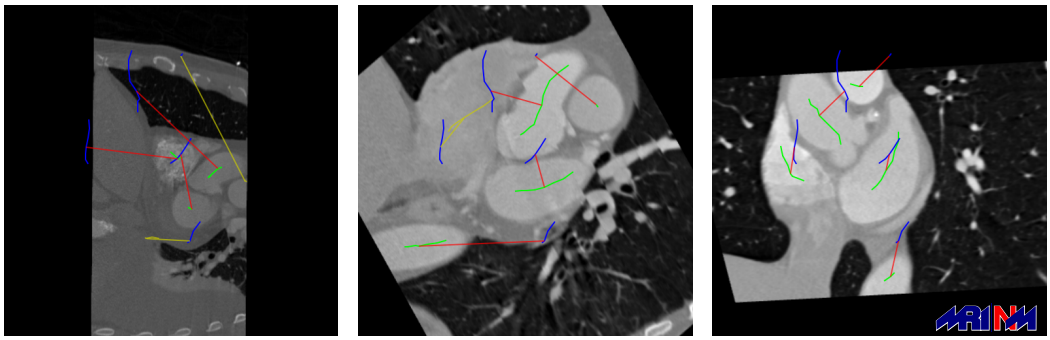


Figure 4.2: Correctly matched CT slices

An example of a mismatched slice is given in Figure 4.3. The reason for the failure of the algorithm is the missing information about the artery going down at the backbone and because only midpoints of striplines are used to compute the transformation matrix $M$.

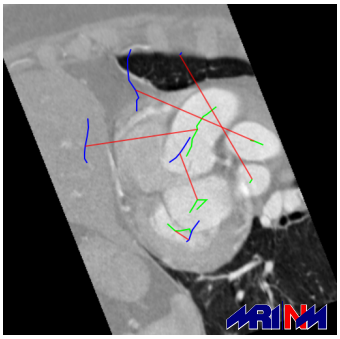


Figure 4.3: Mismatched CT slices

Selecting the 2D slices while caring for the same characteristics as the reference data and implementing a matching using all nodes (not only the midpoint selector for the striplines) should make it possible to obtain perfect matchings in 2D. Because the task was to match and registrate 3D scans, no further work was done on 2D slices.

## 4.2 3D Matching

To test the presented algorithm, a reference dataset was mapped to 13 arbitrary datasets. The datasets were downscaled to half of their size ($256 \cdot 256 \cdot slices/2$) to decrease the memory usage as well as the computation time for the sphere filter. The reference graph was created from a dataset with the most regular heart structure and a good distribution of the contrast medium at the blood cavities. To reduce the number of nodes and therefore the computation time for matching, the parameters were modified to create larger edges and less nodes for the reference graph. The matching to the destination graphs was done using the same parameters for every CT dataset to test a full automatical registration. To restrict the allowed nodes for comparison avoiding a computation time depending on the number of nodes, the minimum distance between the matching nodes of one graph is precomputed for every graph restricting the allowed number of test-matchings. The FFTW [8] for the sphere filter creates so called plans for every time the filter is used. This time is not included into the statistics because those plans can be precomputed for every possible case if the sphere filter is used in a productive system.

The parameters used in the program are given in figure 4.4. The duration for each filter and the results can be found in table 4.1. The last row in the table shows the percentage of the concordances found.

The value for the correct matching was manually taken. To find the concordance nodes representing the corresponding graph edges and blood cavities, the following method was applied: Each node of the reference graph was projected to the destination graph. The nearest node of the projected node to the destination graph is assumed to be the corresponding one, if the distance was less than the sphere radius at the projected node multiplied by a factor of 1.5. Otherwise no correspondance was found for the projected node. The matching of one area is assumed to be positive, only if at least one concordance node was found and if no mismatching concordance exists for this area.

| Dataset: | BK1945 | FIN1928 | KH1952 | KM1935 | MP1947 | NEU1950 | PM1938 |
|---|---|---|---|---|---|---|---|
| **Seconds for filters:** | | | | | | | |
| *(additional information)* | | | | | | | |
| **Sphere:** | 8.796 | 5.157 | 17.756 | 9.427 | 23.519 | 5.421 | 21.811 |
| **Gradient:** | 0.876 | 0.462 | 1.132 | 0.830 | 1.012 | 0.467 | 1.565 |
| **Particles:** | 0.876 | 0.362 | 1.144 | 0.391 | 0.724 | 0.402 | 1.493 |
| *(# particles)* | 1123 | 2688 | 2297 | 1228 | 1867 | 2343 | 3906 |
| **Graph:** | 0.002 | 0.005 | 0.006 | 0.003 | 0.005 | 0.007 | 0.010 |
| *(# graph nodes)* | 29 | 29 | 55 | 32 | 48 | 33 | 79 |
| *(# graph strips)* | 4 | 4 | 12 | 8 | 11 | 10 | 21 |
| **Matching:** | 19.974 | 4.869 | 11.782 | 17.326 | 11.471 | 2.617 | 1.267 |
| **Total seconds:** | 30.52 | 10.85 | 31.81 | 27.97 | 36.73 | 8.91 | 26.14 |
| | **Concordances: Nodes of reference dataset connected to destination dataset** | | | | | | |
| **Right heart chamber:** | op | √ | √ | op | √ | √ | M |
| **Right atrium:** | o | √ | √ | op | √ | √ | M |
| **Left jeart chamber:** | √ | √ | √ | √ | √ | √ | M |
| **Left atrium:** | √ | √ | √ | √ | M | √ | √ |
| **Aorta:** | √ | √ | √ | √ | √ | √ | √ |
| **Aorta descendens:** | √ | √ | √ | √ | √ | √ | √ |
| **Correct matchings (1.0=100%):** | **1.0** | **1.0** | **1.0** | **1.0** | **0.83** | **1.0** | **0.5** |

| Dataset: | PS1930 | PS1944 | SAH1935 | SFX1930 | TA1946 | VM1941 |
|---|---|---|---|---|---|---|
| **Seconds for filters:** | | | | | | |
| *(additional information)* | | | | | | |
| **Sphere:** | 17.785 | 10.112 | 5.465 | 4.477 | 12.327 | 16.538 |
| **Gradient:** | 1.346 | 0.684 | 0.407 | 0.505 | 1.098 | 0.991 |
| **Particles:** | 1.818 | 0.644 | 0.214 | 0.100 | 0.692 | 1.321 |
| *(# particles)* | 5313 | 1034 | 1305 | 566 | 1927 | 2484 |
| **Graph:** | 0.015 | 0.003 | 0.003 | 0.002 | 0.004 | 0.005 |
| *(# graph nodes)* | 87 | 31 | 19 | 19 | 40 | 53 |
| *(# graph strips)* | 20 | 6 | 4 | 7 | 6 | 10 |
| **Matching:** | 6.148 | 10.572 | 1.772 | 3.071 | 6.302 | 22.366 |
| **Total seconds:** | 27.11 | 22.01 | 7.86 | 8.15 | 20.42 | 41.22 |
| | **Concordances: Nodes of reference dataset connected to destination dataset** | | | | | |
| **Right heart chamber:** | √ | √ | √ | o | √ | P |
| **Right atrium:** | √ | √ | √ | o | √ | √ |
| **Left heart chamber:** | √ | √ | √ | F | √ | √ |
| **Left atrium:** | √ | √ | √ | √ | √ | M |
| **Aorta:** | √ | √ | M | M | √ | √ |
| **Aorta descendens:** | √ | √ | √ | √ | √ | √ |
| **Correct matchings (1.0=100%):** | **1.0** | **1.0** | **0.83** | **0.5** | **1.0** | **0.67** |

√: Correct matching
M: Mismatch - some nodes of the reference graph are mapped to a wrong area in the destination dataset
F: Failed - the matching failed to find *any* corresponding points
o: Blood cavity is not visible in destination dataset due to missing contrast medium
op: Blood cavity is not visible but the reference graph is mapped to the correct correct areas in the destination dataset
P: Partially mismatched nodes

Computer: Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz, 2 GB RAM
OS: Debian, Linux version 2.6.26-1-amd64

Table 4.1: Results applying the algorihm to arbitrary CT datasets

| | |
|---|---|
| **Sphere Filter:** | |
| min_value_contrast_medium | 180 |
| max_value_contrast_medium | 850 |
| min_sphere_radius | 6 |
| valid_error | 0.1 |
| fft_threads | 4 |
| **Particle Filter:** | |
| initial_particle_distance | 1 |
| stop_gradient | 10 |
| min_start_radius | 10 |
| **Graph Filter:** | |
| min_next_distance | 15 |
| max_next_distance | 20 |
| **Matching Filter:** | |
| max_unprojected_distance | 128 |
| min_matrix_generation_point_distance | 20 |
| mutual_graph_distance | 1 |
| adaptive_max_points | 2000 |
| threads | 4 |
| Matrix restrictions: | |
| max_rotation_angle | 45 |
| min_scale_factor | 0.3 |
| max_scale_factor | 1.3 |
| min_shear_factor | -1.1 |
| max_shear_factor | 1.1 |
| max_translation_dist_x | 128 |
| max_translation_dist_y | 128 |
| max_translation_dist_z | 100 |
| pre_translation_x | 128 |
| pre_translation_y | 128 |
| pre_translation_z | 100 |
| **Visualization for the Registration:** | |
| matching_max_connection_distance | 1.5 |

Figure 4.4: Parameters used for the different filters (fixed for every mapping)

## 4.3 Conclusions

The table 4.1 shows, that all desired areas were found in 61% of the given datasets. This are datasets which have the most similarity to the reference dataset, even if the datasets have not been aligned in a preprocess step. This leads to the assumption, that CT datasets of the same patient would be matched correctly in almost every case.
An improvement of the alignment could be obtained in the remaining 39% even if not all areas could be registrated. The reason for the partially mismatchings are CT datasets with an unregular heart structure (Enlarged muscles, deformed left atrium, etc.). A way to find better concordances for deformed hearts would be to use more than one reference graph representing different heart deformations. This can be done because the matching filter takes less than 10 seconds in average for the test datasets.
Finally, the aorta descendens was found in 100% of the datasets. The algorithm improved the alignment in all cases with an average time of 23 seconds applying the whole algorihm.

## 4.4 Further utility

The presented algorithm is not only applicable for blood vessels. Also trabecular bone structures can be represented with graphs as shown in Figure 4.5 if the tubular structures have a tubular form. For further work, using a combination of graph and mesh representations as well as a local tresholding filter (because the centered structures become darker) should make it possible to represent even the plate like structures of the trabecular bone structures for a structural analysis of the bone.

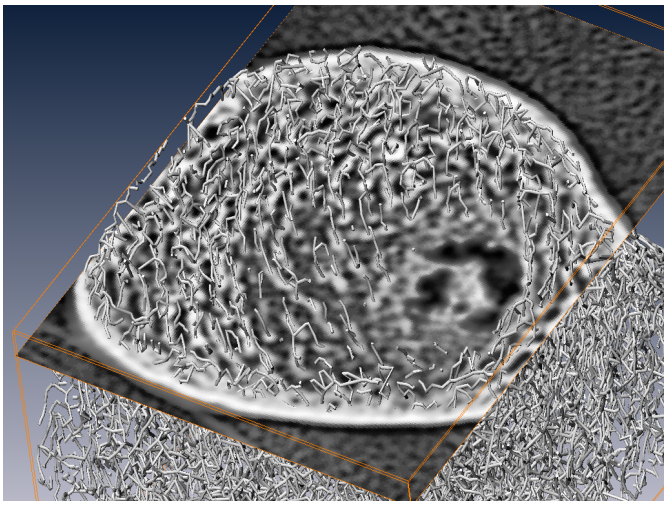


Figure 4.5: Snapshot showing representation of bone structures: Tubic like structures are represented very well in the outer area whereas the plate like structures are not representated by the graph

*I think computer viruses should count as life. I think it says something about human nature that the only form of life we have created so far is purely destructive. We've created life in our own image.*

Stephen Hawking (1942 - )

# 5

# Possible further Improvements and Usage

This chapter was written to give ideas for possible improvements which can be implemented to enhance the quality, adaptability and computation time for the presented and also other segmentation and registration problems. Some improvements try to handle problems which occur due to the discretization with the segmentation of smaller tubic structures. This also makes it necessary to improve the registration speed by selecting the marker nodes more carefully, thus avoiding the computation of unnecessary transformation matrices.

**Bilinear interpolation for particle trajectories:** For large tubic structures as they are in current coronary CT data scans, the results are satisfying without any interpolation for particle trajectories. It would be better to use a bilinear interpolation for finer tubic structures because the particles would be denser at the vessel centers following the interpolated gradient instead of "jumping" on the discrete grid.

**Sparse graph representation using spline curves:** The graph which was built from the particle set represents the tubic centers using straight lines. The graph nodes can be extended with parameters describing a spline curve, which minimizes the distance between the particles and the curve. This would reduce the amount of graph nodes and produce a better representation of the blood vessels and a higher accuracy for the computation of the matching distance between the transformed points $P'$ and the vessel centers represented with splines.

**Aliased kernel for spherical filters:** Better spherical results could be obtained by using an aliased kernel. Again, this is important for smaller blood vessels. An aliased kernel is created by setting the *kernel values to the fraction of the cell which is covered by the sphere surface band* described by the space between the sphere with radius $r - 1$ and radius $r$.

**Segmentation of omitted nodes at stripline endings:** Nodes at *stripline endings are sometimes omitted* because they lie within the radius *min_dist* of

the stripline ending node. Using a small value for *min_dist* would decrease this problem with the undesired behaviour that the created graph has more nodes. Therefore a final test at line endings has to be made if there are omitted nodes which can be used to extend the stripline.

**Monte Carlo and hierarchical methods for marker point selection:** One possibility to speed up the registration process is to use randomly selected points for the transformation matrix $M$. As already described in the registration chapter using a monte carlo based method may lead to a local extremum.
An alternative could be to use a hierarchical method like an octtree. First this accellerates the selection of points within a minimum distance. Secondly the transformation matrix can be dropped for cluster nodes, which are given by the centers of the octtree cells near the leaves representing graph nodes. Dropping a matrix constructed from cluster nodes also drops all possible combinations using nodes covered by the cells of the cluster nodes. This method demands a modified version of the restrictions which are necessary for an accurate implementation of the different sizes of the cluster cells. E. g. the restriction of the translation transformation has to be extended by the possible movements of the cluster nodes within the corresponding cluster cell.

**Maximum distance of construction nodes:** An easy method to decrease the possible matchings is to introduce a maximum distance between the construction nodes. This makes the method conditionally slightly more unstable but decreases the possible matchings. Therefore the maximum distance between the construction nodes specifies the best conditionality of the registration algorithm for the resulting transformation matrix $M$, the minimum distance gives the border for the worst conditionality for the selection of marker points.

**Matching using properties of graph:** So far only the distance of nodes to graph edges was used to compute the quality of the matching. This can be improved by using hints of the graphs. E. g. the information if registrated tubes are described by neighboring edges in both the matching and destination graph can be used to add bonus points to the matching quality. Another improvement would be to search for similarities in the slopes of the matched edges. Therefore using graphs for advanced computation of the matching quality makes it possible to improve the prevention of possible totally wrong mismatchings even if the computed matching quality is better than the expected matching.

**Matrix construction on graph nodes:** Using the discretized graph nodes (representing the particles) for the matrix construction leads to the problem that it is still possible that at least one point for the optimal matrix construction is not given by a graph node but lies on a graph edge. The restriction for the

marker nodes to a minimum distance solves this problem slightly and should hand back a good solution for the tranformation matrix. To get better results for the transformation matrix, the matching points of the destination graph (or matching graph) can be moved virtually along their corresponding neighbored edges while performing the same computations as done for the registrations to get a better matching.
Another method would be to move the marker points within a rectangle assuming that the graph does not represent the blood vessels accurately.

**Matching positive abort:** Abort the matching immediately with the recent transformation matrix if the computed overall distance is below a heuristic value which specifies the accurateness of the transformation matrix and the matching.

**Spherical shrinking to get matching points:** Another method would be a spherical shrinking to select matching nodes: A sphere is set arount the whole domain centered either at the domain center or the particle center of gravity. To get more matching points for registration this sphere is shrinked successively and the new points outside the sphere are included into the set of allowed points used to create the transformation matrix $M$.

**Avoid edge construction:** If the graph construction algorithm creates a new edge, it should be verified if this edge lies within a blood vessel by walking voxelwise along the edge while checking the sphere data because it is possible that an edge is created over 2 neighbored blood vessels

**Improving neighbor connection of striplines:** In special situations it can happen that the connections at vessel bifurcations are not found to represent the cavaties correctly. A reason is e. g. that the last 2 ending nodes dont aim in the correct direction. This can be solved by emitting more virtual particles at a sphere surface with the radius $[min\_dist; max\_dist]$ around the stripline ending nodes.

# Bibliography


[1] Stephan Achenbach. Computed tomography coronary angiography, 2006.

[2] Nuklearmed. Klinik der TU München; Germany.
`http://www.nuk.med.tu-muenchen.de/`.

[3] DICOM-Toolkit.
`http://libkdtree.alioth.debian.org/`.

[4] Marco Francone; et al. Ecg-gated multi-detector row spiral ct in the assessment of myocardial infarction: correlation with non-invasive angiographic findings, 2005.

[5] Mehdi Namdar MD; Thomas F. Hany MD; et al. Integrated pet/ct for the assessment of coronary artery disease: A feasibility study, 2005.

[6] Paul Schoenhagen MD; Sandra S. Halliburton; et al. Noninvasive imaging of coronary arteries: Current and future role of multidetector row ct, 2004.

[7] KD-Tree C++ interface.
`http://libkdtree.alioth.debian.org/`.

[8] FFTW Library.
`http://www.fftw.org/`.

[9] R. Stunken; R. Logen. Perfect graph matching in linear runtime, 2009.

[10] Debian Linux OS.
`http://www.debian.org/`.

[11] Bader Michael PD. Real fourier transformation.
`http://www5.in.tum.de/lehre/vorlesungen/algowiss/ss08/material.html`.

[12] Markus Schwaiger MD; Sibylle Ziegler PhD; Stephan G. Nekolla PhD. Pet/ct: Challenge for nuclear cardiology, 2005.

[13] Martin Schreiber. Project webpage.
`http://home.in.tum.de/~schreibm/idp/`.

[14] Huckle Thomas; Schneider Stefan. *Numerik für Informatiker*. Springer, Berlin, September 2002.