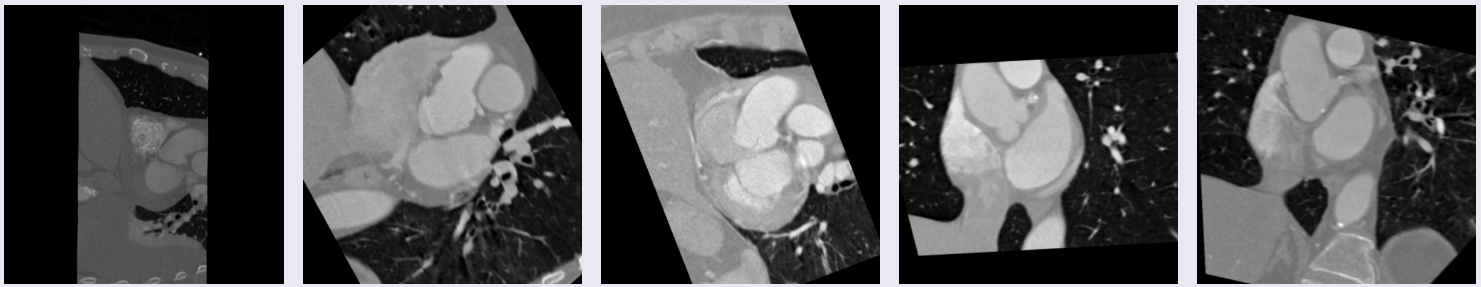
3D Spherical based Segmentation and Registration

Martin Schreiber
martin.schreiber@in.tum.de

December 2, 2008

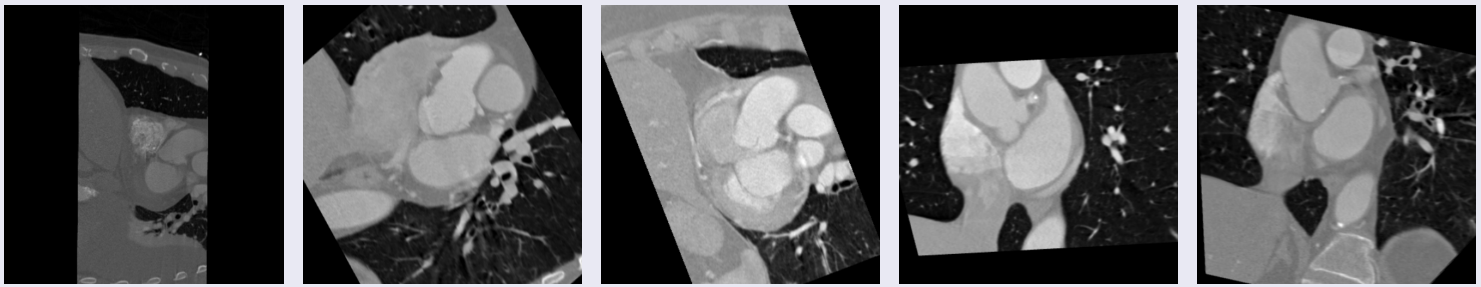
Comparing CT datasets



Selected CT slices from different patients showing similar areas (Source: Nuklearmed. Klinik der TU Muenchen)

- Comparing different CT datasets taken at different time
- Matching of 3D datasets performed by hand takes a lot of time
- Existing algorithms work on projective matrices...
  - - matching a large amount of points created by edge detection
  - - computing the difference of every data domain voxel
  - - ...
- $\Rightarrow$ Use (blood) vessels as (more) characteristic data with **less data for representation**

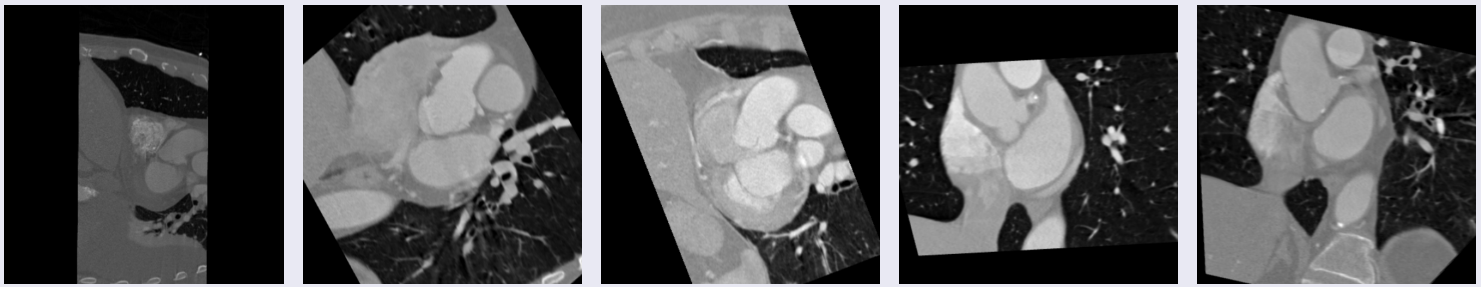
Comparing CT datasets



Selected CT slices from different patients showing similar areas (Source: Nuklearmed. Klinik der TU Muenchen)

- Comparing different CT datasets taken at different time
- Matching of 3D datasets performed by hand takes a lot of time
- Existing algorithms work on projective matrices...
  - - matching a large amount of points created by edge detection
  - - computing the difference of every data domain voxel
  - - ...
- $\Rightarrow$ Use (blood) vessels as (more) characteristic data with **less data for representation**

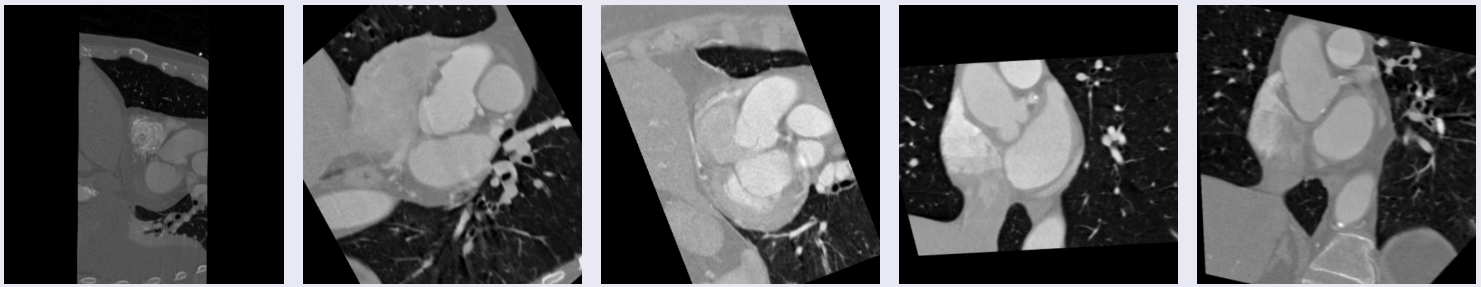
Comparing CT datasets



Selected CT slices from different patients showing similar areas (Source: Nuklearmed. Klinik der TU Muenchen)

- Comparing different CT datasets taken at different time
- Matching of 3D datasets performed by hand takes a lot of time
- Existing algorithms work on projective matrices...
  - - matching a large amount of points created by edge detection
  - - computing the difference of every data domain voxel
  - - ...
- $\Rightarrow$ Use (blood) vessels as (more) characteristic data with **less data for representation**

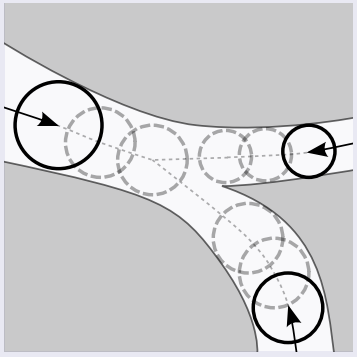
Comparing CT datasets



Selected CT slices from different patients showing similar areas (Source: Nuklearmed. Klinik der TU Muenchen)

- Comparing different CT datasets taken at different time
- Matching of 3D datasets performed by hand takes a lot of time
- Existing algorithms work on projective matrices...
  - - matching a large amount of points created by edge detection
  - - computing the difference of every data domain voxel
  - - ...
- $\Rightarrow$ Use (blood) vessels as (more) characteristic data with **less data for representation**

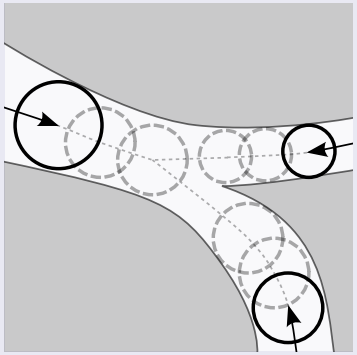
Simulation of spheres moving through vessels



Simulated movement of sphere within the blood vessel

- **Simulate the movement** of spheres through vessels touching the borders like a chimney-sweeper
- **Sphere radii are variable** and grow/shrink to touch the vessel borders
- Spheres stay in the **center of the vessels**
- The radius of **the sphere representing the blood vessels** is stored at each **center point**
- Direct implementation would be too inefficient due to collision tests, realignment of sphere, etc.

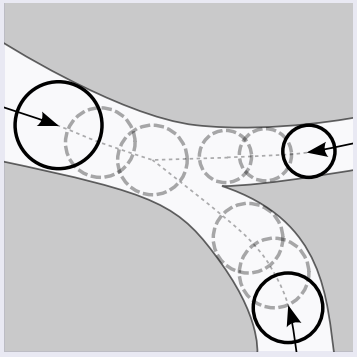
Simulation of spheres moving through vessels



Simulated movement of sphere within the blood vessel

- **Simulate the movement** of spheres through vessels touching the borders like a chimney-sweeper
- **Sphere radii are variable** and grow/shrink to touch the vessel borders
- Spheres stay in the **center of the vessels**
- The radius of **the sphere representing the blood vessels** is stored at each **center point**
- Direct implementation would be too inefficient due to collision tests, realignment of sphere, etc.

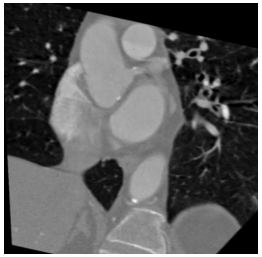
Simulation of spheres moving through vessels



Simulated movement of sphere within the blood vessel

- **Simulate the movement** of spheres through vessels touching the borders like a chimney-sweeper
- **Sphere radii are variable** and grow/shrink to touch the vessel borders
- Spheres stay in the **center of the vessels**
- The radius of **the sphere representing the blood vessels** is stored at each **center point**
- Direct implementation would be too inefficient due to collision tests, realignment of sphere, etc.

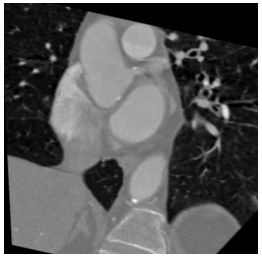
Simulation of Sphere movement by different Filters



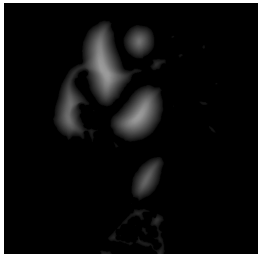
- **Raw** CT data
- Growing **Spheres**
- Computation of **Gradient**
- **Particle** emission
- ...

Simulation of Sphere movement by different Filters

- **Raw** CT data



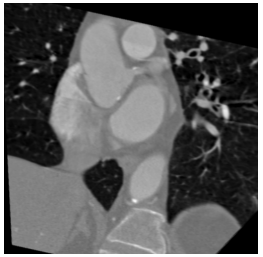
- Growing **Spheres**



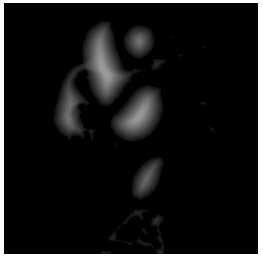
- Computation of **Gradient**
- **Particle** emission
- ...

Simulation of Sphere movement by different Filters

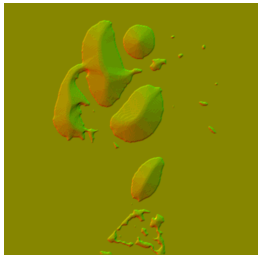
- **Raw** CT data



- Growing **Spheres**



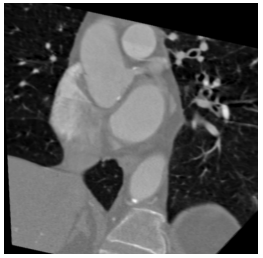
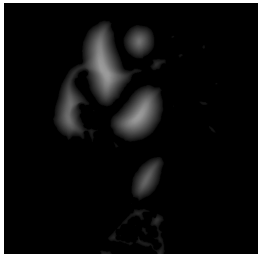
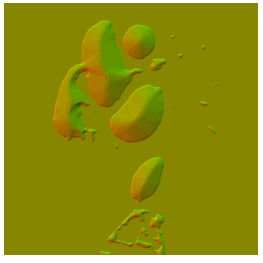
- Computation of **Gradient**



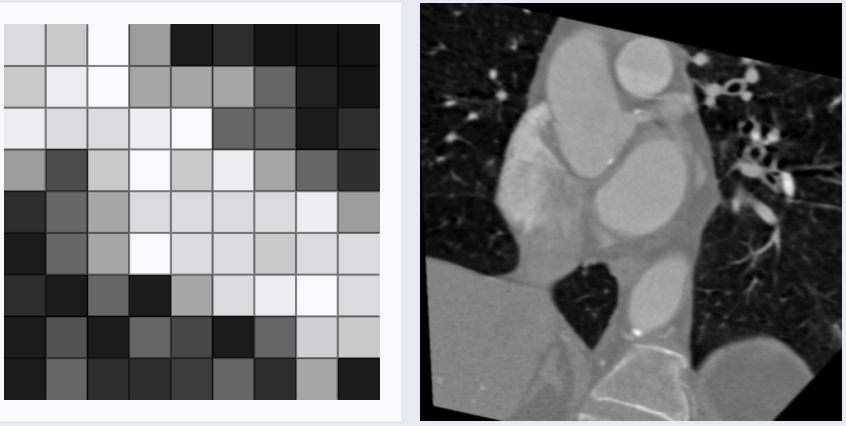
- **Particle** emission
- ...

Simulation of Sphere movement by different Filters

- **Raw** CT data
- Growing **Spheres**
- Computation of **Gradient**
- **Particle** emission
- ...

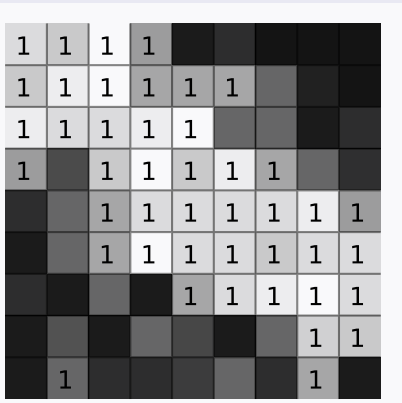




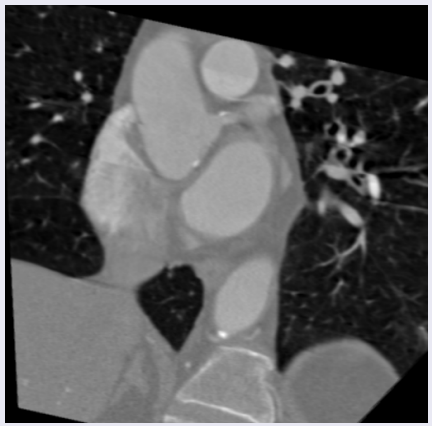
Spheres - Window



- Values for coronary contrast media are usually **within a specific window**

Spheres - Threshold flag field

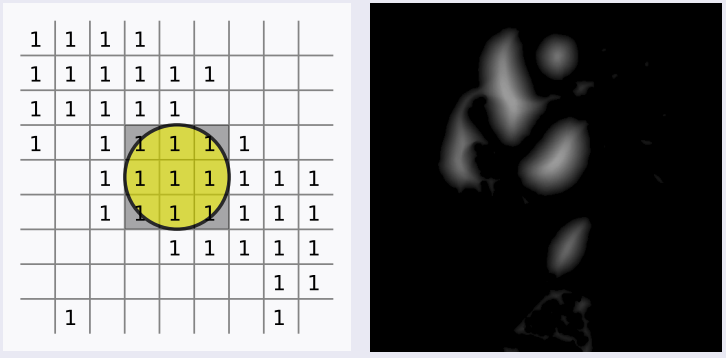




- Thresholding CT data by the window range $[win_{min}; win_{max}]$
  - Flag field **speeds up computations**
  - Important for **convolution** in frequency room (later)
  - Typical values for coronary contrast media: $[150; 1000]$

$$FlagData_{pos} = \begin{cases} 1 & win_{min} < value_{pos} < win_{max} \\ 0 & else \end{cases}$$

Spheres - First spherical test

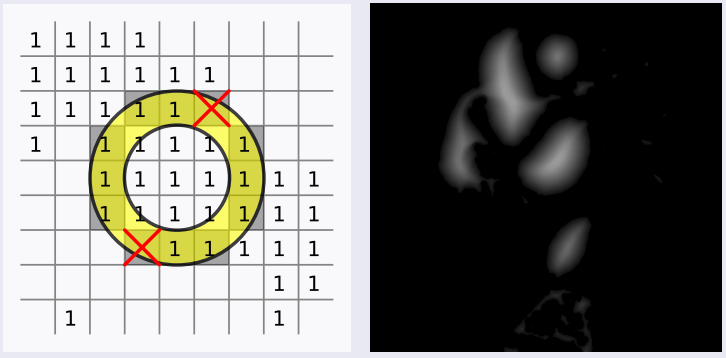


- **Avoiding early** stop of spherical growing on **noisy data**:
  - Start with a radius *StartRadius*
  - Abort if there are **too many mismatching flags** within the sphere
  - Output value $SphereData_{pos}$ of current voxel is set to **0 if first spherical test was not successful**

Start sphere abort criteria

$$\frac{\sum_{pos \in StartSphere} FlagData_{pos}}{|\text{Voxels in Sphere}|} < MaxMismatch$$

Spheres - Growing spheres, radius 3

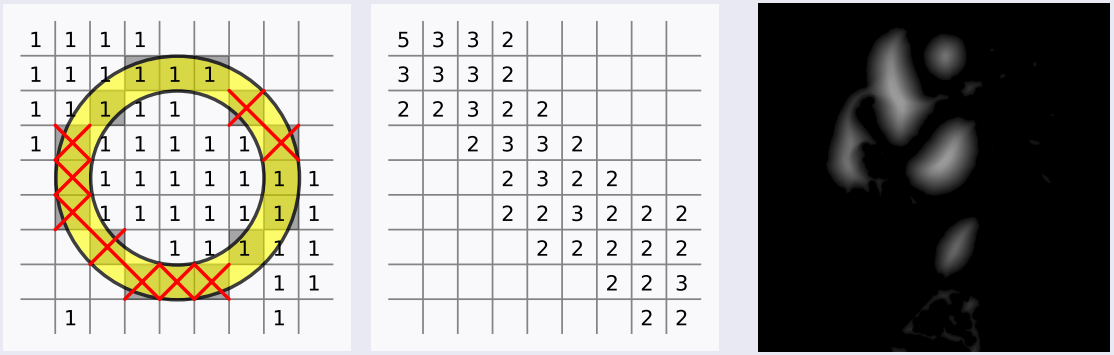


- If there was no output data set continue growing the sphere
  - Growing is stopped if too many **mismatching voxels on the sphere surface exceed a specific error value**
  - Output value $SphereData_{pos}$ is set to the **current sphere radius if the abort criteria is met**

Sphere growing abort criteria

$$\frac{\sum_{pos \in SphereSurface} FlagData_{pos}}{|\text{Sphere Surface}|} < MaxMismatch$$

Spheres - Growing spheres, radius 4



Left: Mismatch value 9/16 exceeds the allowed rate *MaxMismatch*
Middle: Stored radius values after applying spherical filter
Right: Spherical dataset created by the spherical filter

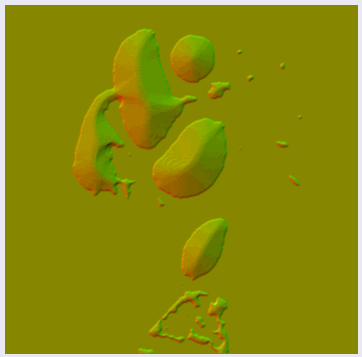
- Sphere growing for every voxel returns data set with the following **properties**:
  - The sphere radii **represent the blood vessels** with a diameter of at least $2 \cdot \mathit{StartRadius}$
  - Blood vessels could be **reconstructed** with the spherical dataset by **joining the sphere volumes**
  - Spheres totally covered by larger spheres can be dropped if we are only interested in a representative data for blood vessels

Gradient - Computation

5	3	3	2					
3	3	3	2					
2	2	3	2	2				
		2	3	3	2			
			2	3	2	2		
			2	2	3	2	2	2
				2	2	2	2	2
						2	2	3
							2	2

⇒

0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,1	-1,0	-3,0	-2,-2	0,0	0,0	0,0	0,0
0,0	1,3	0,1	-1,-1	-2,-3	-2,-2	0,0	0,0	0,0
0,0	2,2	3,3	1,0	-1,-1	-3,-2	-2,-2	0,0	0,0
0,0	0,0	2,2	3,1	0,1	-1,-1	-2,-2	-2,-2	0,0
0,0	0,0	2,0	2,2	1,1	0,0	-1,0	0,-2	0,0
0,0	0,0	0,0	2,2	2,2	0,3	0,0	0,0	0,0
0,0	0,0	0,0	0,0	0,2	2,2	2,2	1,2	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0



Gradient computation with central differences

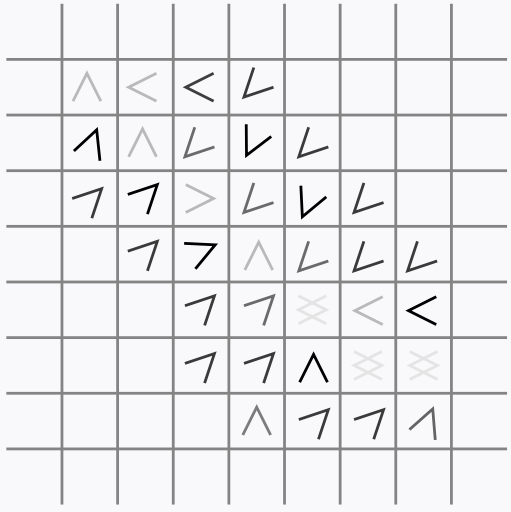
$$GradientData_{pos} = \begin{pmatrix} \frac{\delta SphereData_{pos}}{\delta x} \\ \frac{\delta SphereData_{pos}}{\delta y} \\ \frac{\delta SphereData_{pos}}{\delta z} \end{pmatrix}$$

$$= \begin{pmatrix} SphereData_{pos+(1,0,0)} - SphereData_{pos-(1,0,0)} \\ SphereData_{pos+(0,1,0)} - SphereData_{pos-(0,1,0)} \\ SphereData_{pos+(0,0,1)} - SphereData_{pos-(0,0,1)} \end{pmatrix} \cdot 0.5$$

Gradient - Meaning

Gradient vectors scaled by 2

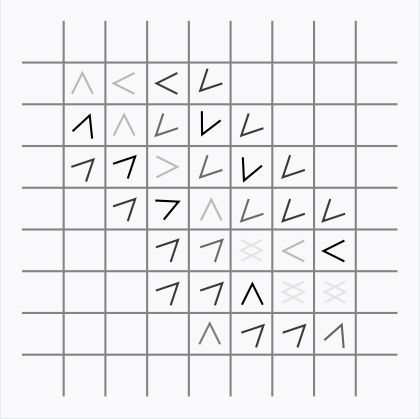
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,1	-1,0	-3,0	-2,-2	0,0	0,0	0,0	0,0
0,0	1,3	0,1	-1,-1	-2,-3	-2,-2	0,0	0,0	0,0
0,0	2,2	3,3	1,0	-1,-1	-3,-2	-2,-2	0,0	0,0
0,0	0,0	2,2	3,1	0,1	-1,-1	-2,-2	-2,-2	0,0
0,0	0,0	2,0	2,2	1,1	0,0	-1,0	0,-2	0,0
0,0	0,0	0,0	2,2	2,2	0,3	0,0	0,0	0,0
0,0	0,0	0,0	0,0	0,2	2,2	2,2	1,2	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

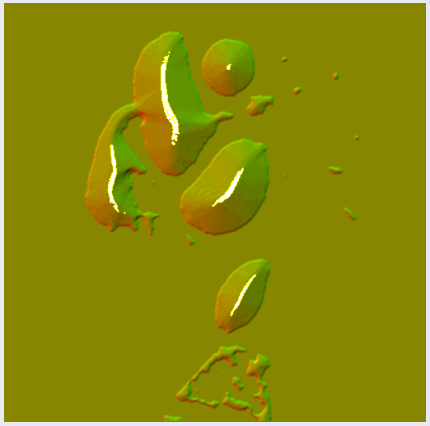


- Gradients aim to the local center of largest sphere in neighborhood
  $\Rightarrow$ **Can be used for efficient simulation of origin problem**
  - **Growing** the sphere **forces movement to the center** of vessel
  - Movement **direction** is **given by gradient**
- Gradient can be **smoothed** if sphere data has a high frequency

Particles - Emission and movement

Particle emission and movement along the gradient

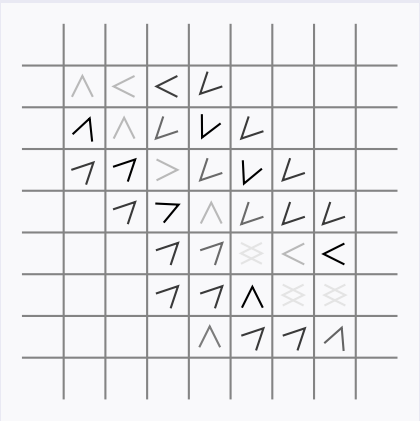


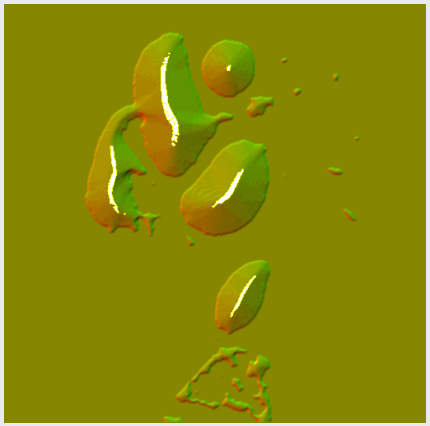


- **Emit particles** starting on voxels with *sphere* > *MinEmissionRadius*
- *MinEmissionRadius* **avoids emitting particles in small vessels** (for registration unnecessary) and positive-false segmented areas like bones
- Particles **follow the local gradient vector**
- Length of gradient vector is **small at the center of vessels**
- Particle **stops** if the length of gradient vector is **below a specific value**

Particles - Emission and movement

Particle emission and movement along the gradient

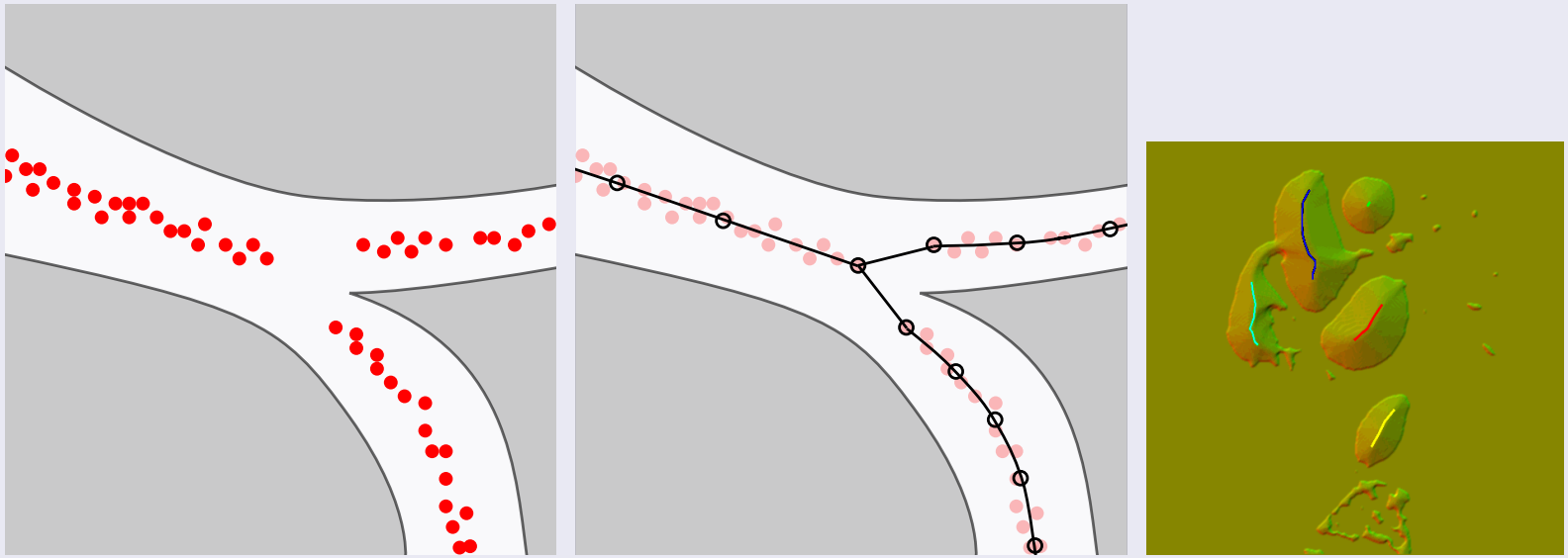




- **Emit particles** starting on voxels with *sphere* > *MinEmissionRadius*
- *MinEmissionRadius* **avoids emitting particles in small vessels** (for registration unnecessary) and positive-false segmented areas like bones
- Particles **follow the local gradient vector**
- Length of gradient vector is **small at the center of vessels**
- Particle **stops** if the length of gradient vector is **below a specific value**

Graph - Construction

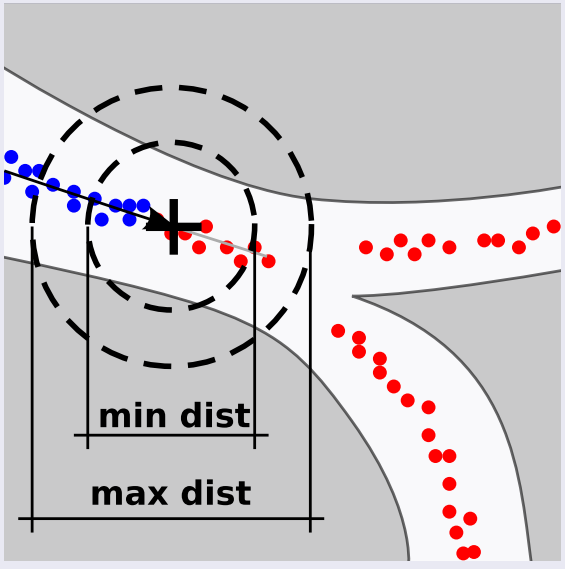
Graph reconstruction



- Representing blood vessels by graphs
- Radius is also stored for each node for advanced registration
- **Reduces** matching of the **large particle amount** to matching of **sparse graph nodes**

Graph - Construction - Step 1 of 3

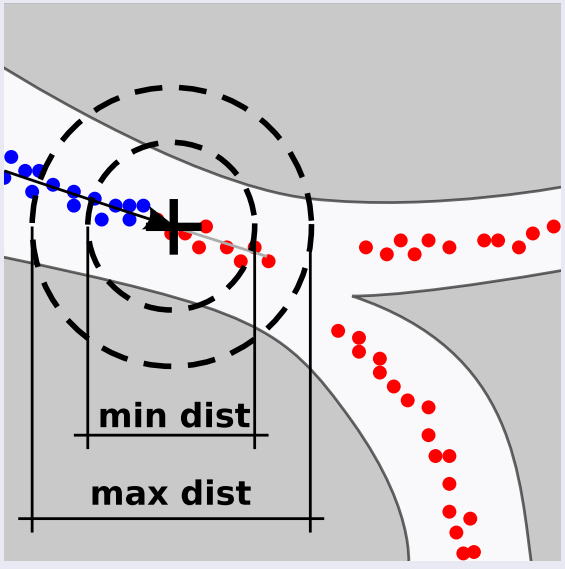
Connecting neighbored particles to strips



- Each particle has a **flag** *used* which is set if the particle is already **represented by an edge**
- Search for **neighbored particle within a specific range** $[min\_dist, max\_dist]$ where the used flag is not yet set
- Take particle which is **furthest away** as node *NextNode*
- **Set used flag** for all particles within the range *max_dist*
- Continue at the node *NextNode*

Graph - Construction - Step 1 of 3

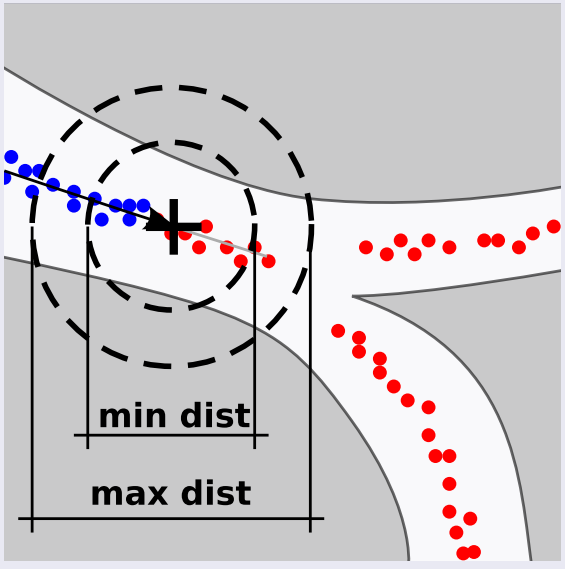
Connecting neighbored particles to strips



- Each particle has a **flag** *used* which is set if the particle is already **represented by an edge**
- Search for **neighbored particle within a specific range** $[min\_dist, max\_dist]$ where the used flag is not yet set
- Take particle which is **furthest away** as node *NextNode*
- **Set used flag** for all particles within the range *max_dist*
- Continue at the node *NextNode*

Graph - Construction - Step 1 of 3

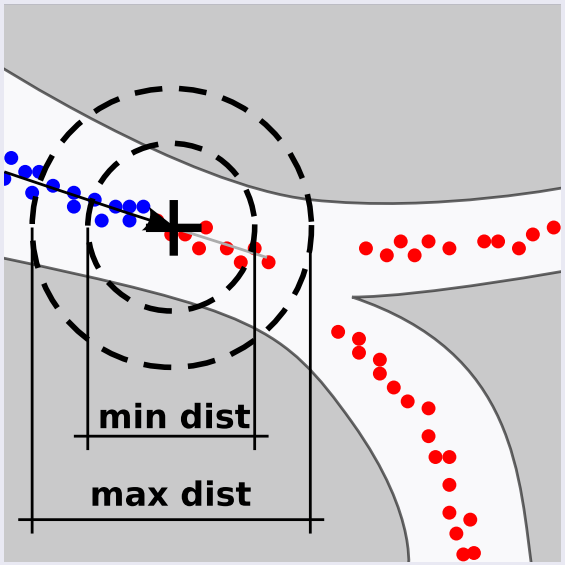
Connecting neighbored particles to strips



- Each particle has a **flag** *used* which is set if the particle is already **represented by an edge**
- Search for **neighbored particle within a specific range** $[min\_dist, max\_dist]$ where the used flag is not yet set
- Take particle which is **furthest away** as node *NextNode*
- **Set used flag** for all particles within the range *max_dist*
- Continue at the node *NextNode*

Graph - Construction - Step 1 of 3

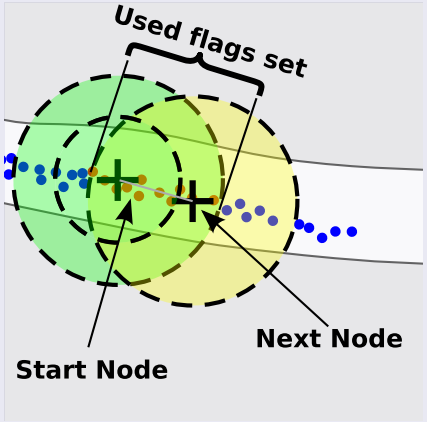
Connecting neighbored particles to strips



- Each particle has a **flag** *used* which is set if the particle is already **represented by an edge**
- Search for **neighbored particle within a specific range** $[min\_dist, max\_dist]$ where the used flag is not yet set
- Take particle which is **furthest away** as node *NextNode*
- **Set used flag** for all particles within the range *max_dist*
- Continue at the node *NextNode*

Graph - Construction - Step 2 of 3

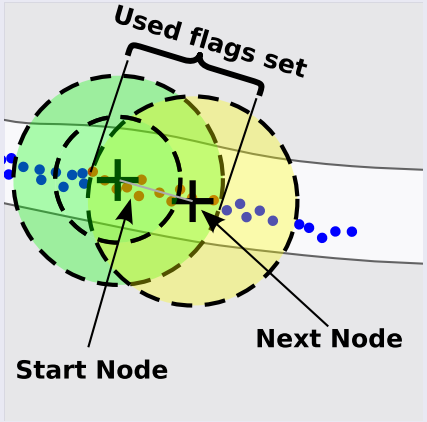
Handling of first strip node within non-forking area



- **Start** each edge construction at unused particle with **maximum radius** $\Rightarrow$ First node can have **2 neighbors**
- Setting all used flags of first node within the range *max_dist* **avoids creating edges in the opposite direction**
- $\Rightarrow$ Set *used* flags only for particles which are **also in the range** *max_dist* **of NextNode**
- After creation of a stripline "in one direction", **restart again at first node** to extend stripline in opposite direction

Graph - Construction - Step 2 of 3

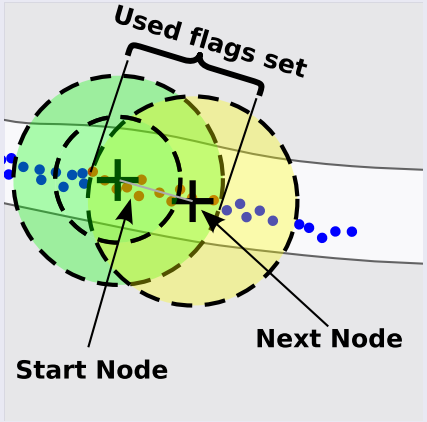
Handling of first strip node within non-forking area



- **Start** each edge construction at unused particle with **maximum radius** $\Rightarrow$ First node can have **2 neighbors**
- Setting all used flags of first node within the range *max_dist* **avoids creating edges in the opposite direction**
- $\Rightarrow$ Set *used* flags only for particles which are **also in the range** *max_dist* **of NextNode**
- After creation of a stripline "in one direction", **restart again at first node** to extend stripline in opposite direction

Graph - Construction - Step 2 of 3

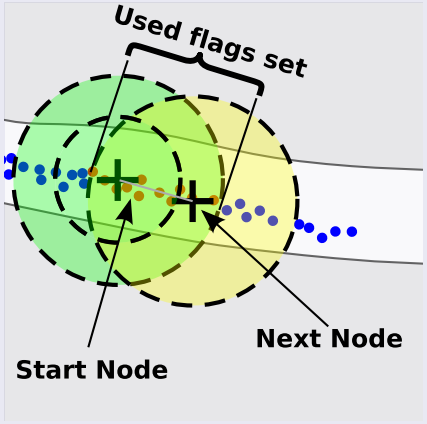
Handling of first strip node within non-forking area



- **Start** each edge construction at unused particle with **maximum radius** $\Rightarrow$ First node can have **2 neighbors**
- Setting all used flags of first node within the range *max_dist* **avoids creating edges in the opposite direction**
- $\Rightarrow$ Set *used* flags only for particles which are **also in the range** *max_dist* **of NextNode**
- After creation of a stripline "in one direction", **restart again at first node** to extend stripline in opposite direction

Graph - Construction - Step 2 of 3

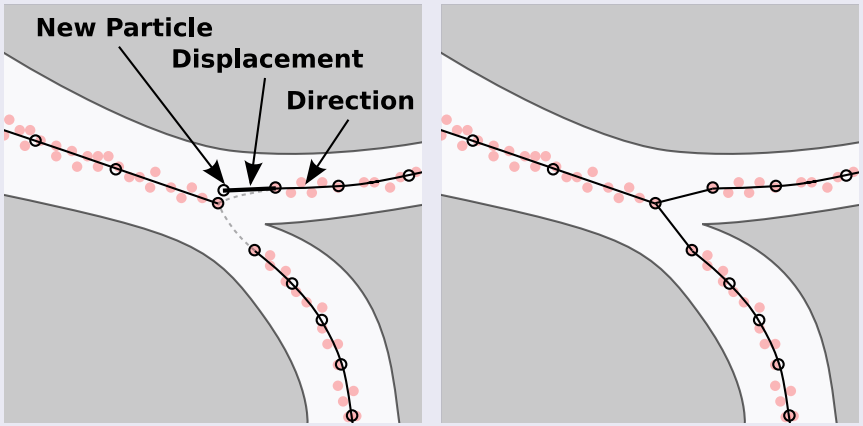
Handling of first strip node within non-forking area



- **Start** each edge construction at unused particle with **maximum radius** $\Rightarrow$ First node can have **2 neighbors**
- Setting all used flags of first node within the range *max_dist* **avoids creating edges in the opposite direction**
- $\Rightarrow$ Set *used* flags only for particles which are **also in the range** *max_dist* **of NextNode**
- After creation of a stripline "in one direction", **restart again at first node** to extend stripline in opposite direction

Graph - Construction - Step 3 of 3

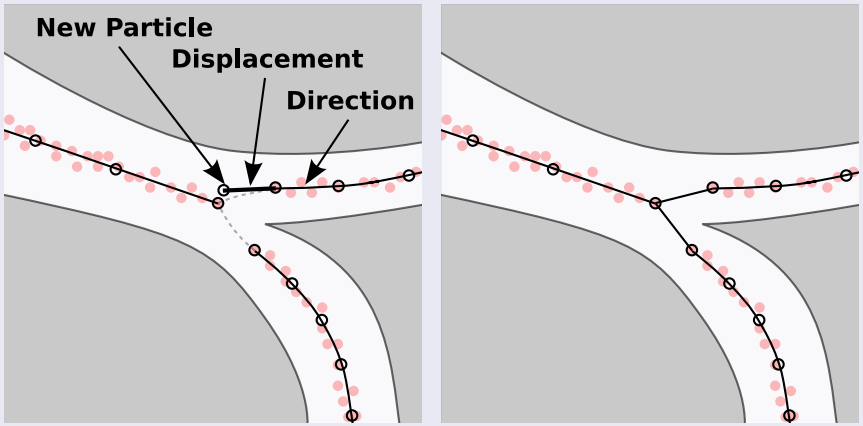
Connecting strips to graphs



- The Gradient is very high on vessel **forkings** $\Rightarrow$ strips are **disconnected**
- Use particle emission to create a connection at forkings
  - Particle is emitted with a **displacement** in the direction described by the **two corner nodes**
  - This particle **follows the gradient** until the **gradient value is below a specific value**
  - If there's a node N_n within the range max_dist: **connect corner node with node** N_n

Graph - Construction - Step 3 of 3

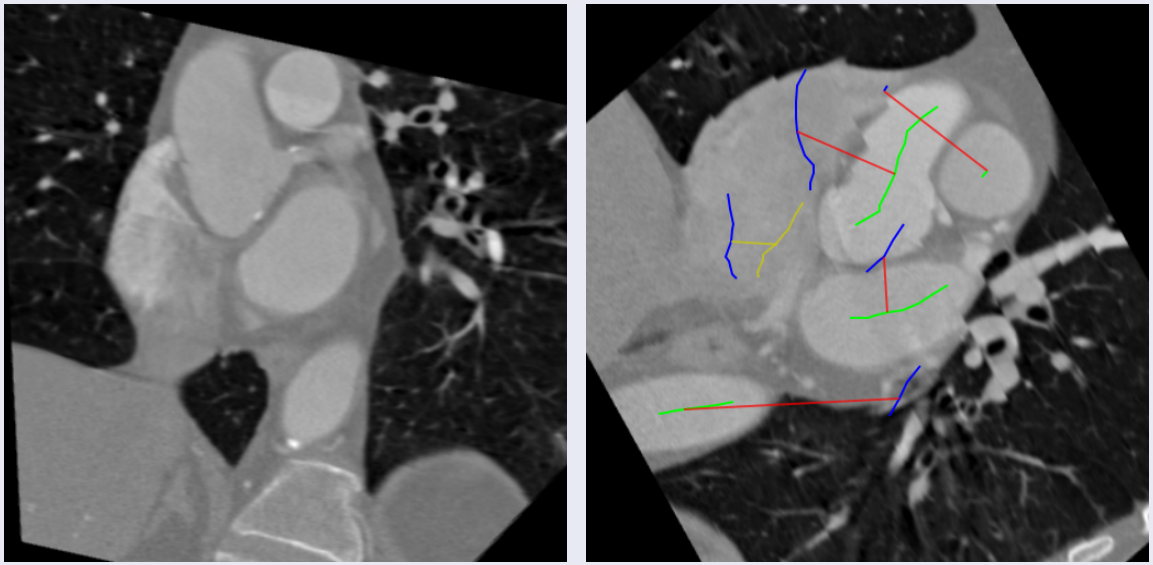
Connecting strips to graphs



- The Gradient is very high on vessel **forkings** $\Rightarrow$ strips are **disconnected**
- Use particle emission to create a connection at forkings
  - Particle is emitted with a **displacement** in the direction described by the **two corner nodes**
  - This particle **follows the gradient** until the **gradient value is below a specific value**
  - If there's a node N_n within the range max_dist: **connect corner node with node** N_n

Matching - Terminology

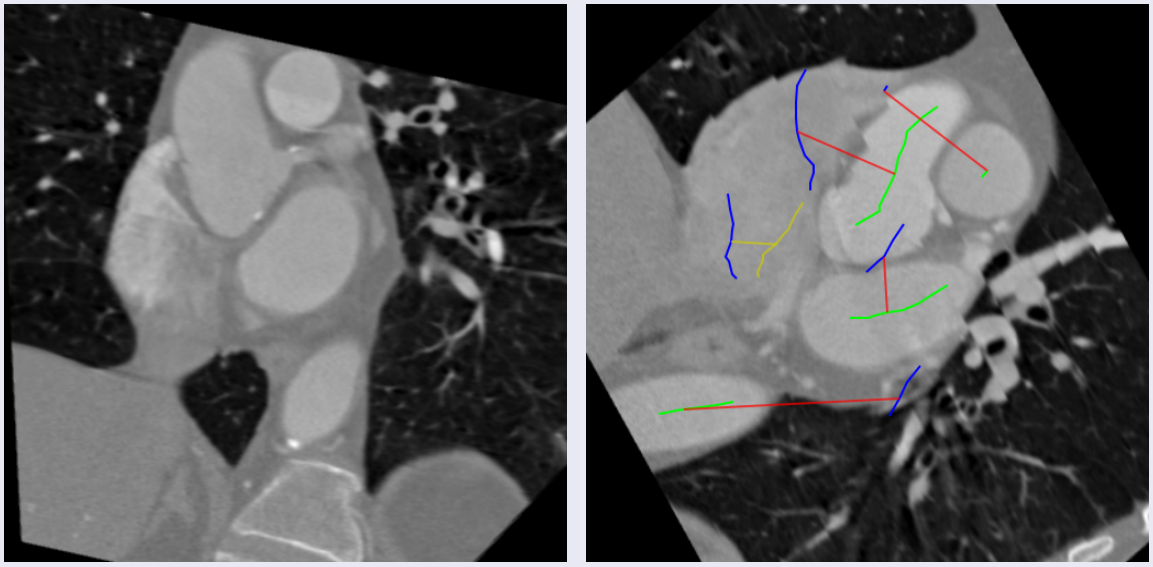
Matching graphs



- Finding **transformation matrix** *M*
- Projecting **nodes P from matching graph** (applying matrix *M* for each point) results in **points P' in "destination space"**
- **Minimize distance** between **nodes P'** and **nearest edge** of destination graph

Matching - Terminology

Matching graphs



- Finding **transformation matrix** *M*
- Projecting **nodes P from matching graph** (applying matrix *M* for each point) results in **points P' in "destination space"**
- **Minimize distance** between **nodes P'** and **nearest edge** of destination graph

Matching

Transformation Matrix M projecting point P to P':

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

Transformation matrix M for 3D CT data can be computed with **4 nodes** P^i, P^j, P^k, P^l of source graph and **4 points** O^i, O^j, O^k, O^l of destination graph (given in homogeneous form)

$$\begin{pmatrix} P^i_x & P^i_y & P^i_z & 1 \\ P^j_x & P^j_y & P^j_z & 1 \\ P^k_x & P^k_y & P^k_z & 1 \\ P^l_x & P^l_y & P^l_z & 1 \end{pmatrix} \begin{pmatrix} m_{1,1} \\ m_{1,2} \\ m_{1,3} \\ m_{1,4} \end{pmatrix} = \begin{pmatrix} O^i_x \\ O^j_x \\ O^k_x \\ O^l_x \end{pmatrix}$$

$$\begin{pmatrix} P^i_x & P^i_y & P^i_z & 1 \\ P^j_x & P^j_y & P^j_z & 1 \\ P^k_x & P^k_y & P^k_z & 1 \\ P^l_x & P^l_y & P^l_z & 1 \end{pmatrix} \begin{pmatrix} m_{2,1} \\ m_{2,2} \\ m_{2,3} \\ m_{2,4} \end{pmatrix} = \begin{pmatrix} O^i_y \\ O^j_y \\ O^k_y \\ O^l_y \end{pmatrix}$$

$$\begin{pmatrix} P^i_x & P^i_y & P^i_z & 1 \\ P^j_x & P^j_y & P^j_z & 1 \\ P^k_x & P^k_y & P^k_z & 1 \\ P^l_x & P^l_y & P^l_z & 1 \end{pmatrix} \begin{pmatrix} m_{3,1} \\ m_{3,2} \\ m_{3,3} \\ m_{3,4} \end{pmatrix} = \begin{pmatrix} O^i_z \\ O^j_z \\ O^k_z \\ O^l_z \end{pmatrix}$$

Matching - Matrix decomposition

- Creating a matrix based on all possible node combinations can produce **unlikely mappings**
  - large shearings
  - large scalings / negative scalings
  - large rotations
  - large translations
- Even an unlikely mapping can produce **wrong matching with best computed matching**
- **Decompose matrix** to basic transformations and **restrict transformations**

Matching - Matrix decomposition

- Creating a matrix based on all possible node combinations can produce **unlikely mappings**
  - large shearings
  - large scalings / negative scalings
  - large rotations
  - large translations
- Even an unlikely mapping can produce **wrong matching with best computed matching**
- **Decompose matrix** to basic transformations and **restrict transformations**

Matching - Matrix decomposition

- Creating a matrix based on all possible node combinations can produce **unlikely mappings**
  - large shearings
  - large scalings / negative scalings
  - large rotations
  - large translations
- Even an unlikely mapping can produce **wrong matching with best computed matching**
- **Decompose matrix** to basic transformations and **restrict transformations**

Matching - Matrix decomposition (cont')

- Matrix M is decomposed in
  - 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 $sh_{1,2,3}$ in shearing matrix H
- $M = H \cdot S \cdot R_z \cdot R_x \cdot R_y \cdot T$
- Decompositions "simulate" different **basic transformations**
  - First the translation is done to align both datasets
  - Secondly the translated dataset is rotated around y axis for better matching
  - ...
- Decomposition to basic transformations give the information for an **"early drop"** (omit the current matrix)

Matching - Matrix decomposition (cont')

- Matrix M is decomposed in
  - 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 $sh_{1,2,3}$ in shearing matrix H
- $M = H \cdot S \cdot R_z \cdot R_x \cdot R_y \cdot T$
- Decompositions "simulate" different **basic transformations**
  - First the translation is done to align both datasets
  - Secondly the translated dataset is rotated around y axis for better matching
  - ...
- Decomposition to basic transformations give the information for an **"early drop"** (omit the current matrix)

Matching - Translation

- Translation Matrix $T = \begin{pmatrix} 1 & . & . & t_x \\ . & 1 & . & t_y \\ . & . & 1 & t_z \\ . & . & . & 1 \end{pmatrix}$
- Translation decomposition:

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

(Search values for T to eliminate the rightmost column)

- Computation of the rightmost column values of M gives the implicit solution for T

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} m_{1,4} \\ m_{2,4} \\ m_{3,4} \end{pmatrix}$$

Matching - Translation

- Translation Matrix $T = \begin{pmatrix} 1 & . & . & t_x \\ . & 1 & . & t_y \\ . & . & 1 & t_z \\ . & . & . & 1 \end{pmatrix}$
- Translation decomposition:

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

(Search values for T to eliminate the rightmost column)

- Computation of the rightmost column values of M gives the implicit solution for T

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} m_{1,4} \\ m_{2,4} \\ m_{3,4} \end{pmatrix}$$

Matching - Translation

- Translation Matrix $T = \begin{pmatrix} 1 & . & . & t_x \\ . & 1 & . & t_y \\ . & . & 1 & t_z \\ . & . & . & 1 \end{pmatrix}$
- Translation decomposition:

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

(Search values for T to eliminate the rightmost column)

- Computation of the rightmost column values of M gives the implicit solution for T

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} m_{1,4} \\ m_{2,4} \\ m_{3,4} \end{pmatrix}$$

Matching - Rotation

Rotation matrices (example for given y axis):

- Use rotation matrices (from QR decomposition) to set **values below diagonal to 0**

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

- Eliminating entry $m''_{3,1}$:

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

Assuming $m''_{3,1}$ should be set to 0:

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

Computation of R_y is done by using the inverted angle α

Matching - Rotation

Rotation matrices (example for given y axis):

- Use rotation matrices (from QR decomposition) to set **values below diagonal to 0**

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

- Eliminating entry $m''_{3,1}$:

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

Assuming $m''_{3,1}$ should be set to 0:

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

Computation of R_y is done by using the inverted angle α

Matching - Scaling and Shearing

- Result **after translation and rotation** elimination is an **upper diagonal matrix** $M^{(3)}$
- Final decomposition returns the **shearing and scaling matrix**

$$M^{(3)} = H \cdot S \Longleftrightarrow$$

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

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

$$h_1 \cdot s_y = m^{(3)}_{1,2} \Longleftrightarrow h_1 = \frac{m^{(3)}_{1,2}}{s_y}$$

$$h_2 \cdot s_z = m^{(3)}_{1,3} \Longleftrightarrow h_2 = \frac{m^{(3)}_{1,3}}{s_z}$$

$$h_3 \cdot s_z = m^{(3)}_{2,3} \Longleftrightarrow h_3 = \frac{m^{(3)}_{2,3}}{s_z}$$

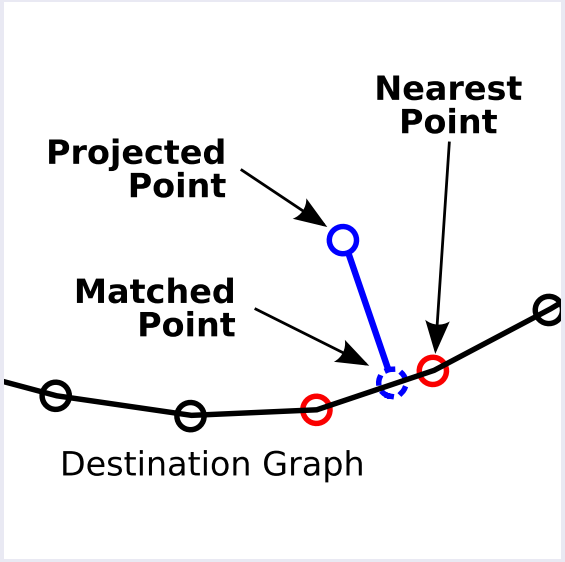
Matching

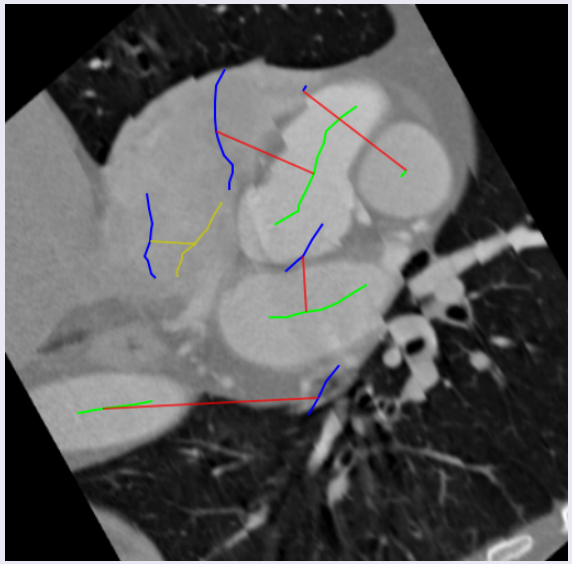
- To work with minimal transformations of permitted matrices, **pretranslate the center of image** to $(0, 0, 0)$
- Empirical values to match $256x256x200$ CT scans:
  - maximum relative rotation angle: $90°$
  - scale factor $\in [0.7; 1.3]$
  - shear factor $\in [-0.3; 0.3]$
  - maximum relative translation: 100
  - pretranslation matrix: $P = \begin{pmatrix} 1 & . & . & -128 \\ . & 1 & . & -128 \\ . & . & 1 & -100 \\ . & . & . & 1 \end{pmatrix}$

Matching

- To work with minimal transformations of permitted matrices, **pretranslate the center of image** to $(0, 0, 0)$
- Empirical values to match $256x256x200$ CT scans:
  - maximum relative rotation angle: $90°$
  - scale factor $\in [0.7; 1.3]$
  - shear factor $\in [-0.3; 0.3]$
  - maximum relative translation: 100
  - pretranslation matrix: $P = \begin{pmatrix} 1 & . & . & -128 \\ . & 1 & . & -128 \\ . & . & 1 & -100 \\ . & . & . & 1 \end{pmatrix}$

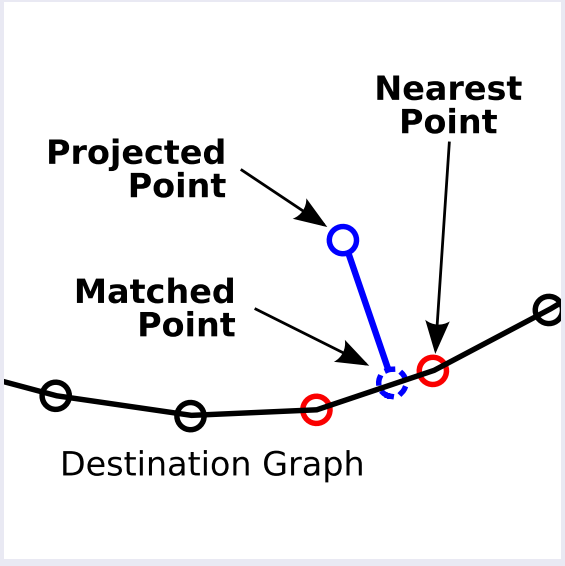
Matching - Computation of matching difference





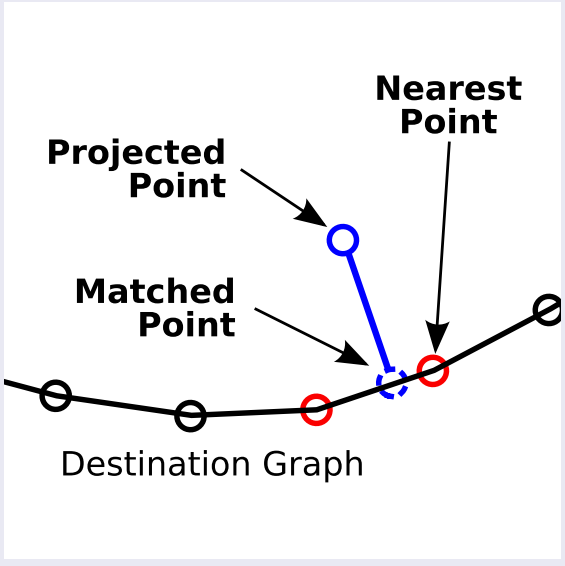
- If the matrix is valid, **nodes P are projected to the destination graph space** giving P'
- For every projected node P', **the nearest node P" of the destination graph** is searched
- For both adjacent nodes of P" the **smallest distance to the edge** is taken
- Best matching matrix M: Projection with the **smallest sum of distances** for all **nodes P' to the corresponding edges**

Matching - Computation of matching difference



- If the matrix is valid, **nodes P are projected to the destination graph space** giving P'
- For every projected node P', **the nearest node P" of the destination graph** is searched
- For both adjacent nodes of P" the **smallest distance to the edge** is taken
- Best matching matrix M: Projection with the **smallest sum of distances** for all **nodes P' to the corresponding edges**

Matching - Computation of matching difference



- If the matrix is valid, **nodes P are projected to the destination graph space** giving P'
- For every projected node P', **the nearest node P" of the destination graph** is searched
- For both adjacent nodes of P" the **smallest distance to the edge** is taken
- Best matching matrix M: Projection with the **smallest sum of distances** for all **nodes P' to the corresponding edges**

Efficient implementation

- Direct implementation would take **more than half an hour** to match a 256x256x200 CT dataset to reference data
- **Sphere filter**:
  - Sphere filter has to be applied **for every point**!
- **Graph filter**:
  - Storing a few particles with a full grid would be a **waste of memory**
  - Computation time for nearest points with $O(r^d)$ increases with resolution n^d
- **Matching**:
  - Using naive approach for a graph with N nodes there would be $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ **matching possibilities** taking hours to compute

Efficient implementation

- Direct implementation would take **more than half an hour** to match a 256x256x200 CT dataset to reference data
- **Sphere filter**:
  - Sphere filter has to be applied **for every point**!
- **Graph filter**:
  - Storing a few particles with a full grid would be a **waste of memory**
  - Computation time for nearest points with $O(r^d)$ increases with resolution n^d
- **Matching**:
  - Using naive approach for a graph with N nodes there would be $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ **matching possibilities** taking hours to compute

Efficient implementation

- Direct implementation would take **more than half an hour** to match a 256x256x200 CT dataset to reference data
- **Sphere filter**:
  - Sphere filter has to be applied **for every point**!
- **Graph filter**:
  - Storing a few particles with a full grid would be a **waste of memory**
  - Computation time for nearest points with $O(r^d)$ increases with resolution n^d
- **Matching**:
  - Using naive approach for a graph with N nodes there would be $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ **matching possibilities** taking hours to compute

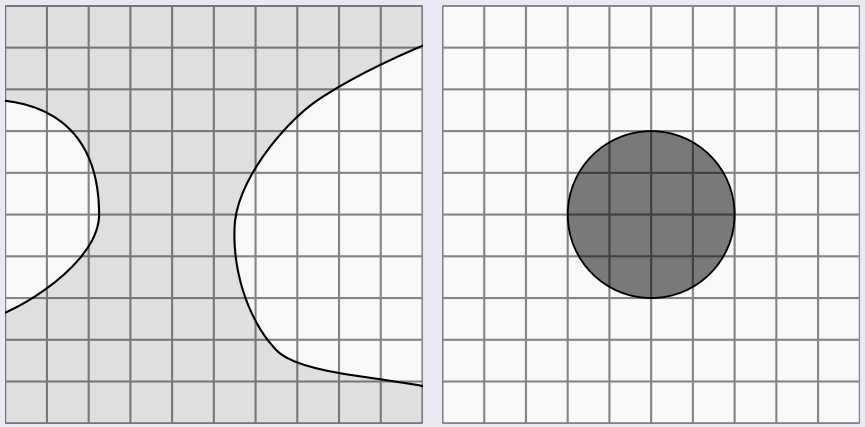
Efficient implementation

- Direct implementation would take **more than half an hour** to match a 256x256x200 CT dataset to reference data
- **Sphere filter**:
  - Sphere filter has to be applied **for every point**!
- **Graph filter**:
  - Storing a few particles with a full grid would be a **waste of memory**
  - Computation time for nearest points with $O(r^d)$ increases with resolution n^d
- **Matching**:
  - Using naive approach for a graph with N nodes there would be $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ **matching possibilities** taking hours to compute

Efficient implementation

- Direct implementation would take **more than half an hour** to match a 256x256x200 CT dataset to reference data
- **Sphere filter**:
  - Sphere filter has to be applied **for every point**!
- **Graph filter**:
  - Storing a few particles with a full grid would be a **waste of memory**
  - Computation time for nearest points with $O(r^d)$ increases with resolution n^d
- **Matching**:
  - Using naive approach for a graph with N nodes there would be $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ **matching possibilities** taking hours to compute

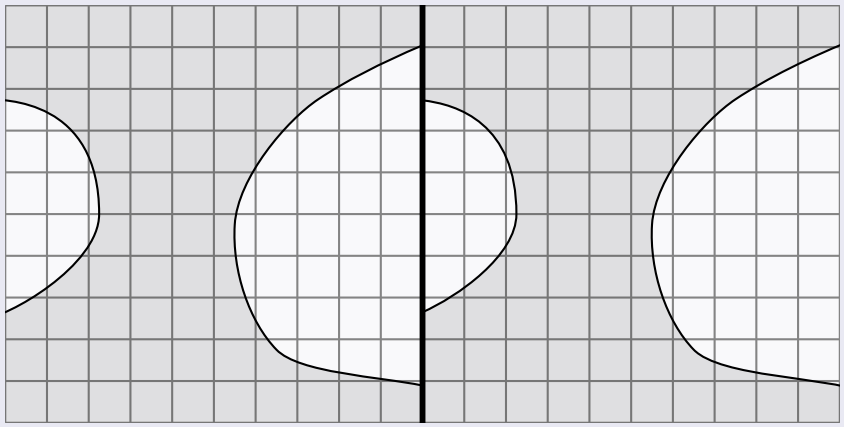
Sphere filter using FFT



- Applying sphere filter is a **convolution for each sphere size**
- Convolution can be done very efficiently in **frequency space**, specially for large kernels

Sphere filter using FFT

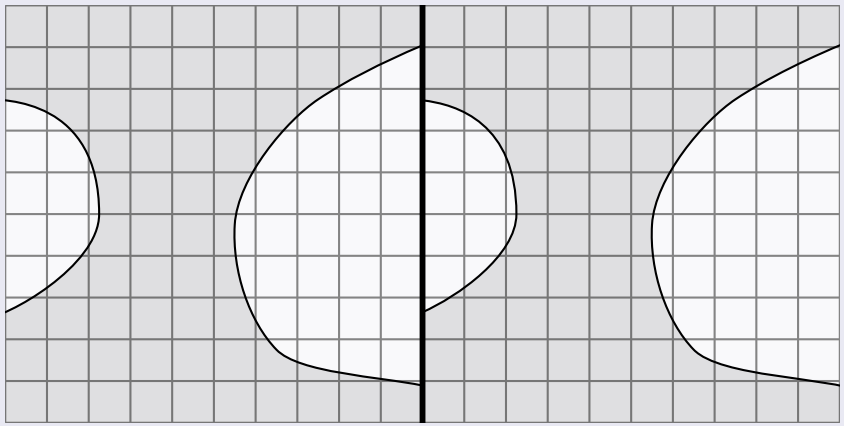
FFT based on periodical data



- **Boundary conditions**: Data fields have to be **padded with extra data** to work with existing FFT libraries (e.g. FFTW)
- Applying the **standard FFT** is based on a **periodical function**
- Multiplying the kernel in frequency room would take **values for spheres from opposite side**
- Large spheres at borders don't represent the local data

Sphere filter using FFT

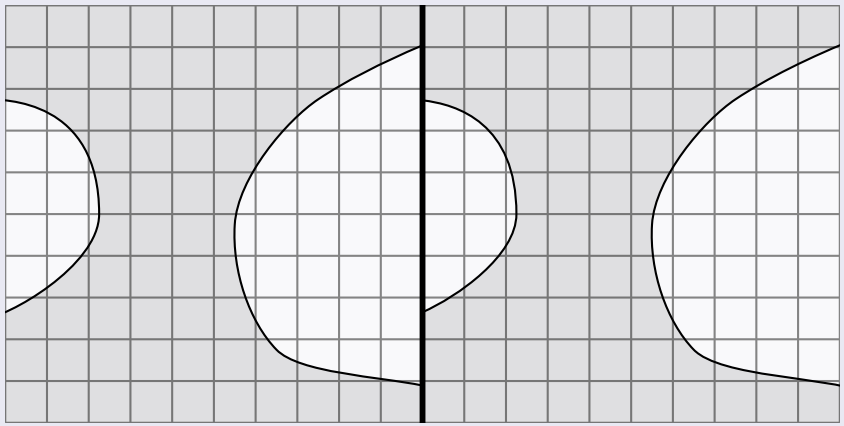
FFT based on periodical data



- **Boundary conditions**: Data fields have to be **padded with extra data** to work with existing FFT libraries (e.g. FFTW)
- Applying the **standard FFT** is based on a **periodical function**
- Multiplying the kernel in frequency room would take **values for spheres from opposite side**
- Large spheres at borders don't represent the local data

Sphere filter using FFT

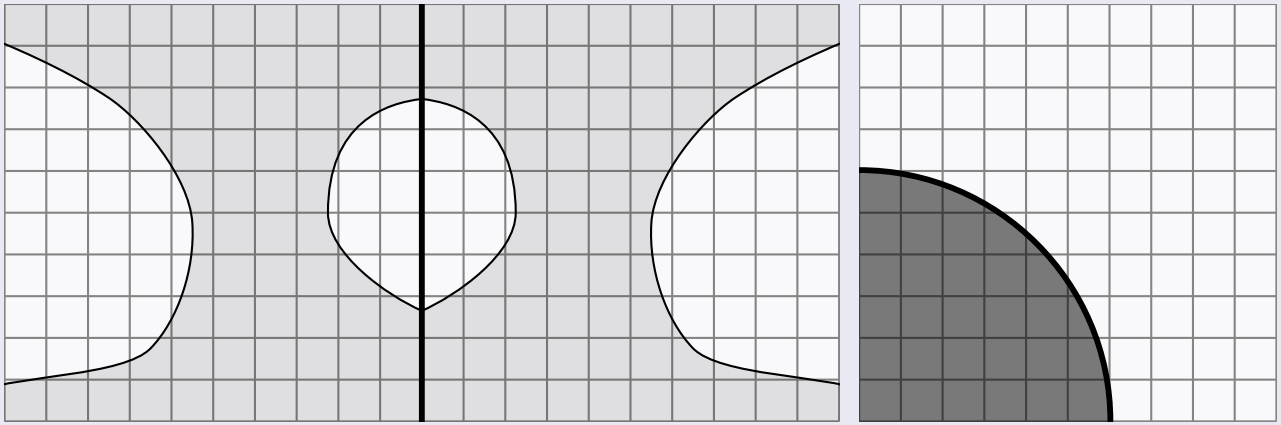
FFT based on periodical data



- **Boundary conditions**: Data fields have to be **padded with extra data** to work with existing FFT libraries (e.g. FFTW)
- Applying the **standard FFT** is based on a **periodical function**
- Multiplying the kernel in frequency room would take **values for spheres from opposite side**
- Large spheres at borders don't represent the local data

Sphere filter using FFT

FFT based on symmetric data

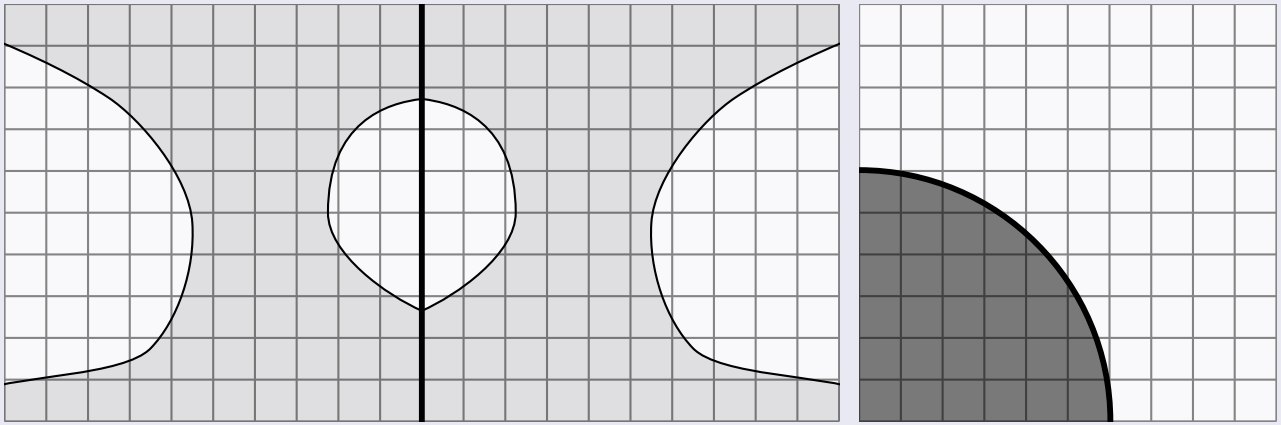


Left: Symmetric data - Right: Kernel

- Using **real data FFT** assuming domain is **symmetric on borders**
- Introduced errors just depend on the local data
- **Kernel** has to be **initialized for only $\frac{1}{8}$ of the data domain**
- Reduces the computation from $>$ 20 minutes to a few seconds

Sphere filter using FFT

FFT based on symmetric data

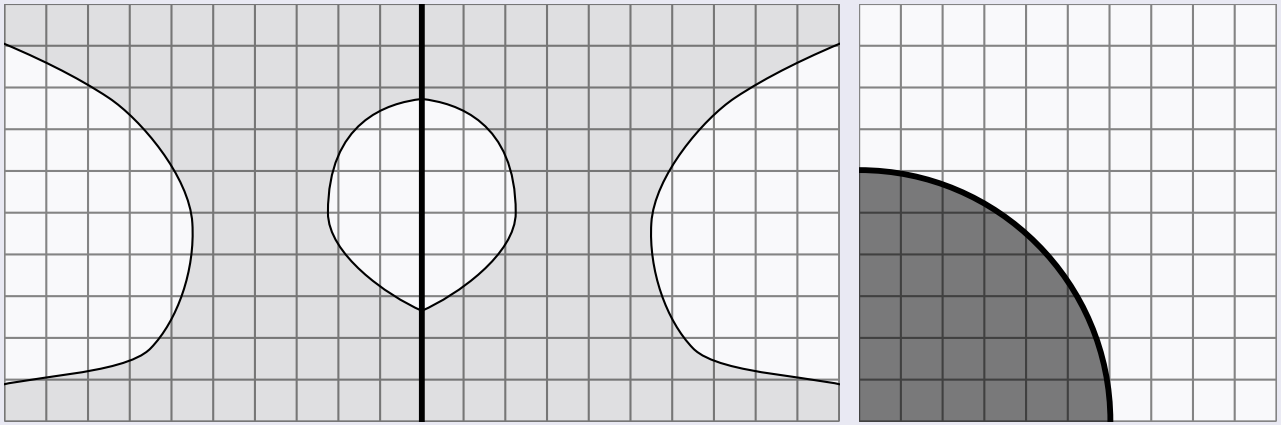


Left: Symmetric data - Right: Kernel

- Using **real data FFT** assuming domain is **symmetric on borders**
- Introduced errors just depend on the local data
- **Kernel** has to be **initialized for only $\frac{1}{8}$ of the data domain**
- Reduces the computation from $>$ 20 minutes to a few seconds

Sphere filter using FFT

FFT based on symmetric data

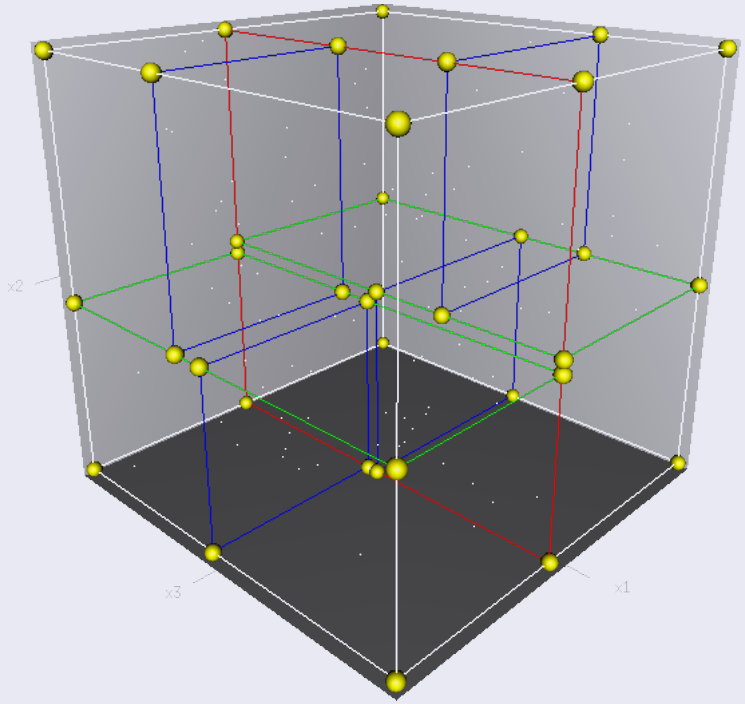


Left: Symmetric data - Right: Kernel

- Using **real data FFT** assuming domain is **symmetric on borders**
- Introduced errors just depend on the local data
- **Kernel** has to be **initialized for only $\frac{1}{8}$ of the data domain**
- Reduces the computation from $>$ 20 minutes to a few seconds

Handling Particles with KD-Trees

KD-Tree

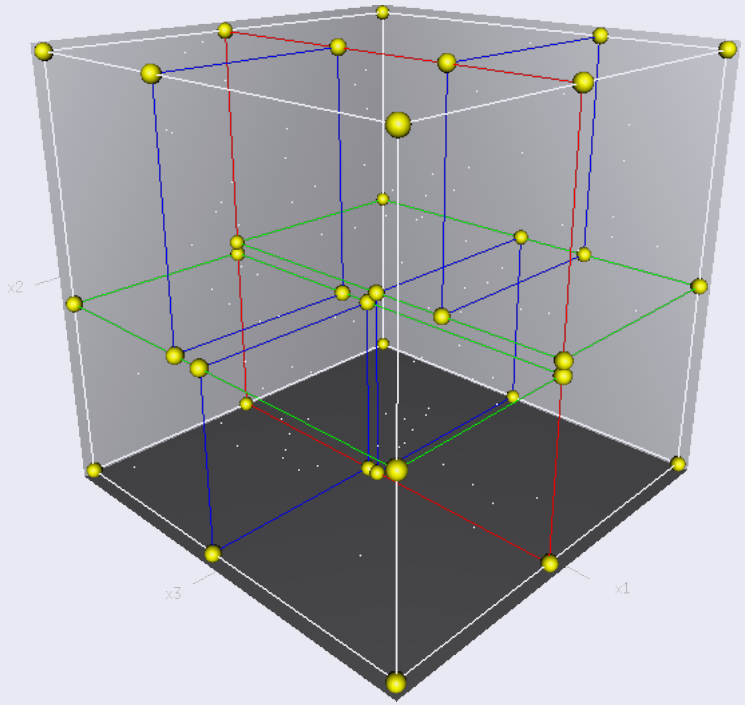


http://en.wikipedia.org/wiki/Image:3dtree.png

- KD Trees store arbitrary points using a tree like structure
- Efficient operations to **find points within a given radius**
- $\Rightarrow$ Graph can be constructed within a second

Handling Particles with KD-Trees

KD-Tree



http://en.wikipedia.org/wiki/Image:3dtree.png

- KD Trees store arbitrary points using a tree like structure
- Efficient operations to **find points within a given radius**
- $\Rightarrow$ Graph can be constructed within a second

Restricting Transformation Matrices

- $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ matching possibilities
- **Discretization** of points introduce **error** ϵ_1
- **Small anatomical differences** of points introduce **errors** ϵ_2
  - Error in matrix after construction becomes **less for far distant points**
  - Use only nodes with a **distance of at least** δ to create a **better conditioned problem**
- $\Rightarrow$ avoids the computation of the transformation matrix for many points
- Omitting **impossible node combinations** and **nodes producing a bad conditioned problem** decreases computation time to a few seconds

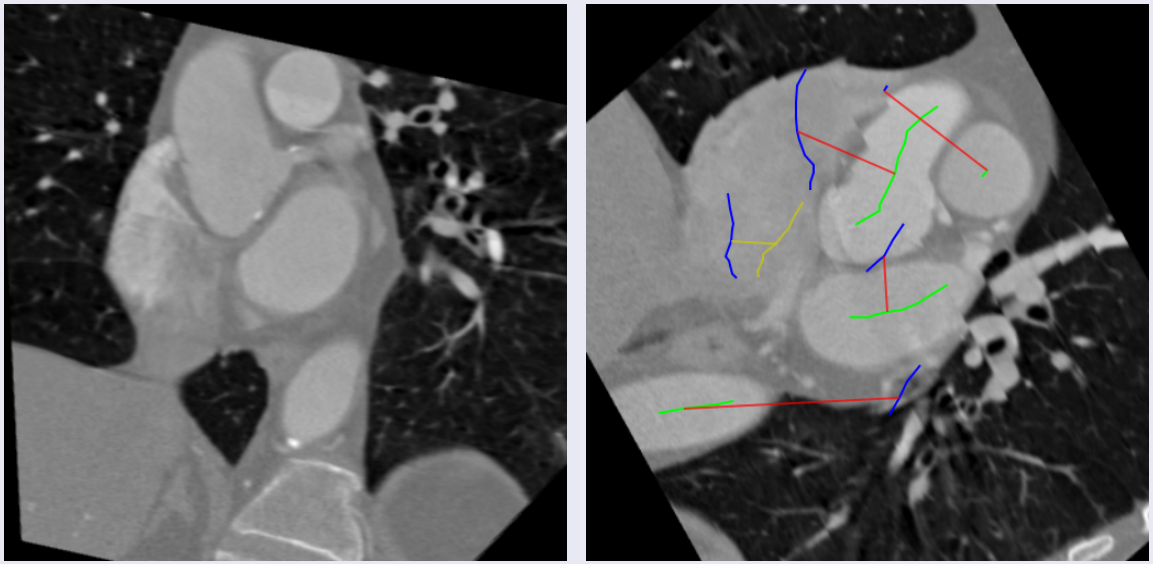
Restricting Transformation Matrices

- $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ matching possibilities
- **Discretization** of points introduce **error** ϵ_1
- **Small anatomical differences** of points introduce **errors** ϵ_2
  - Error in matrix after construction becomes **less for far distant points**
  - Use only nodes with a **distance of at least** δ to create a **better conditioned problem**
- $\Rightarrow$ avoids the computation of the transformation matrix for many points
- Omitting **impossible node combinations** and **nodes producing a bad conditioned problem** decreases computation time to a few seconds

Restricting Transformation Matrices

- $O\left(\left(\frac{N!}{(N-4)!}\right)^2\right)$ matching possibilities
- **Discretization** of points introduce **error** ϵ_1
- **Small anatomical differences** of points introduce **errors** ϵ_2
  - Error in matrix after construction becomes **less for far distant points**
  - Use only nodes with a **distance of at least** δ to create a **better conditioned problem**
- $\Rightarrow$ avoids the computation of the transformation matrix for many points
- Omitting **impossible node combinations** and **nodes producing a bad conditioned problem** decreases computation time to a few seconds

Results



Matching 2D CT slices using only midpoints of edges as graph matching nodes

3D CT heart blood segmentation and registration:

- Can be handled in an **efficient and fast way**
- Takes just a **few seconds** on recent quad-core-systems
- Using translation matrices offers **registration of invisible areas** (yellow line in right image)

Possible improvements

- Use **sparser representation** of graph using **extrapolation or spline curves**
- Using **interpolation** with spherical filter (aliased kernel) for more accurate sphere radii
- Graph construction: include possible omitted nodes at strip endings
- **Randomized/hierarchical matching points** selection (maybe using hints of graph)
- Matching graphs
  - Using **heuristics from graphs** for matching
  - Comparing **edge slopes**
- Use **matching positive abort** if computed **overall distance is below a certain value** (assuming this is the correct matching)

Thank you for your attention

Any questions?

References and Links

- CT Datasets: Nuklearmed. Klinik der TU Muenchen, Germany
- FFTW: http://www.fftw.org/
- KD-Tree: http://libkdtree.alioth.debian.org/
- DICOM-Toolkit: http://dicom.offis.de/dcmtk.php.de