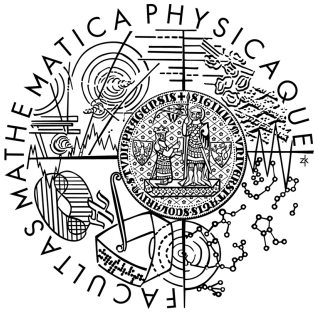
Charles University in Prague, Czech Republic
Faculty of Mathematics and Physics

MASTER THESIS



Szabolcs Gróf

Volume data fusions

Department of Software and Computer Science Education – KSVI

Supervisor: RNDr. Josef Pelikán

Study program: Computer Science
Field of Study: Software Systems

2009

I would like to give thanks to my supervisor, RNDr. Josef Pelikán, for the basic idea and the supervision of this thesis. Further, I would like to thank MUDr. Martin Horák for providing DICOM medical images for developing and testing the practical part of my thesis and also for giving an inspiring lecture about CT and NMR, which was very helpful while studying the background of medical image acquisition.

I hereby certify that I wrote the thesis by myself, using only the referenced sources. I agree with lending the thesis.

In Prague, July 29, 2009 Szabolcs Gróf

Contents

List of Figures

Název práce: Fúze objemových dat
Autor: Szabolcs Gróf
Katedra (ústav): Kabinet software a výuky informatiky – KSVI
Vedoucí diplomové práce: RNDr. Josef Pelikán
e-mail vedoucího: Josef.Pelikan@mff.cuni.cz

Abstrakt: V předložené práci studujeme metody fúze objemových dat. Diplomová práce začíná vysvětlením základních fyzikálních principů vybraných metod pořízení medicínských dat - Počítačová Tomografie, Magnetická Rezonance a Pozitronová Emisní Tomografie. Toto následuje základní přehled o organizaci a struktuře zpracovávaných dat. V další části se práce zabývá teoretickým a matematickým pozadím registrace obrazu. To zahrnuje vysvětlení různých typů transformací, různé druhy interpolace používané při transformaci, kritérium vzájemné informace pro přizpůsobení obrazů a procesu optimalizace. Následuje diskuse o vizualizačních metodách fúze, kde je vysvětleno 13 metod, přičemž vzorové grafické výstupy jsou poskytovány. Další kapitola se zabývá detaily implementace praktické části práce, tu jsou vysvětlené dvě hlavní součásti programu: registrační filtr komponent a komponent vizualizace fúze. UML diagramy jsou uvedeny a účel a funkce třídy z těchto prvků jsou podrobně popsány. Práci uzavírá uživatelská příručka programu a závěr, který shrne popsané metody a diskustuje o možných vylepšeních.

Klíčová slova: objemová data, fúze, registrace, vzájemná informace

Title: Volume data fusions
Author: Szabolcs Gróf
Department: Department of Software and Computer Science Education – KSVI
Supervisor: RNDr. Josef Pelikán
supervisor's e-mail address: Josef.Pelikan@mff.cuni.cz

Abstract: In the present work we study the methods of volumetric data fusion. The thesis starts by explaining the basic physical principles of selected medical image acquisition methods - Computed Tomography, Magnetic Resonance Imaging and Positron Emission Tomography. This is followed by a basic overview of the organization and format of the processed data. In the next part, the thesis deals with the theoretical and mathematical background of image registration. This includes the explanation of different types of transformations, the different types of interpolation used by the transformation, the mutual information criterion for the alignment of images and the process of optimization. The discussion of fusion visualization methods comes next where 13 methods are explained and example image outputs are provided. The next chapter describes the implementation details of the practical part of the thesis by explaining the two main components of the program: the registration filter component and the fusion visualization viewer component. UML diagrams are shown and the purpose and function of the classes of these components are described in detail. The thesis is closed by the user manual of the demonstrating program and a conclusion of the described methods, which also discusses possible future improvements.

Keywords: volume data, fusion, registration, mutual information

Chapter 1

Introduction

Computers play more and more important roles in the field of medicine since there is a great amount of data, which is acquired from special medical hardware, to be manipulated and calculated. One of these fields is medical imaging. Scientists have invented several ways to retrieve special images of the body, depending on the tissue of the given part. Some of these methods include *Positron Emission Tomography*, *Computed Tomography* and *Magnetic Resonance Imaging*.

However, all of these imaging methods have their strengths and their weaknesses, depending on the principle of their function. Some of them work best on those body parts that collect biologically active molecules, which are introduced into the body (like *PET*), some of them show tissues that are more solid than the others (like *CT*) and some of them show tissues that contain more water than the others (like *MRI*).

It is also important, that *MRI* has several operating modes, each of which emphasizes a different kind of tissue.

Furthermore, there is also the common task of acquiring images of a patient at different times, and then following the changes in the state of the given body part.

These above mentioned reasons call for a *fusion* method that would merge the information from all the methods, settings and times, and show all the tissues and details that these images present. The most important imaging methods in this case are those that provide 3D volumetric data.

Before discussing the processed data, the problem and solution of registration and the visualization of fusion itself, a very brief description of the functioning and the advantages and disadvantages of these image acquiring methods will be presented.

1.1 Computed Tomography

Computed tomography acquires images of the body in slices. The object is processed in the transversal plane, which means axial scans. This results in slices that are perpendicular to the long axis of the body.

The principle of this technology is using X-rays that are shot from several points on a circle, and then the absorption of these rays is detected on the other side. The slices are then reconstructed from the amount of absorption at each point on the circle using a computer.

Figure 1.1: CT acquiring device

The degree of absorption of the X-rays is measured in HU units, which depends on the density of the tissue through which the ray was shot. 4096 different shades are distinguished. For example -1000 is air, -100 is fat, 0 is water, +1000 is bone and +3095 is metal.

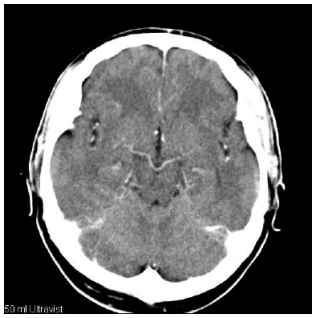


Figure 1.2: CT acquired image

1.2 Magnetic Resonance Imaging

Magnetic Resonance Imaging uses magnetism for acquiring images. A typical MRI scanner works with a static magnetic field, a variable magnetic field and radio-frequency pulses.

Figure 1.3: Modern 3 tesla clinical MRI scanner

MRI uses hydrogen atoms, which have one single proton. With a strong magnetic field,

the rotational magnetic axis of hydrogen atoms are arranged in one direction. Using radio-frequency pulses, the axis of the hydrogen atoms are shifted, the atom's proton is given energy, and after turning off the radio-frequency pulses, observation of how the protons return to the beginning state follows. The recovery of the longitudinal component of the magnetization of the proton is called T1 relaxation with T1 time constant. The loss of the transversal component of magnetization is called T2 relaxation with T2 time constant. When the radio pulse is turned off, an oscillating magnetic field is produced by the transverse component, and it induces a small current in the receiver coil, which is called free induction decay (FID). Ideally this FID would decay exponentially with time constant T2, but because of inhomogeneities between different spatial locations across the body, there is a destructive interference, and this FID is shortened. The time constant for the real FID is called T2* relaxation time, which is always shorter than T2.

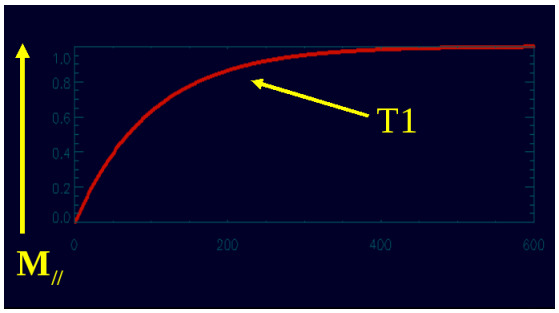


Figure 1.4: T1 relaxation time

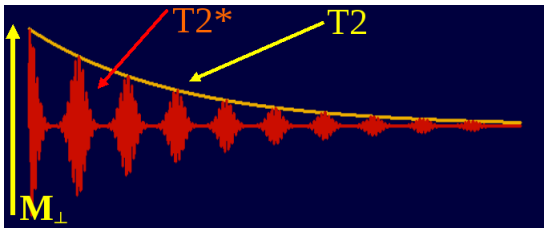


Figure 1.5: T2 and T2* relaxation times

Different types of tissues have different types of T1, T2 and T2* relaxation times. The most hydrogen atoms in the human body are contained by water, so MRI is a good method for those parts of the body that contain fair amount of water. Measuring these relaxation times at different points of the body gives us a good image of the included tissues. There are several other, more advanced, factors to be measured and visualized, but to explain all of them is beyond the scope of this thesis.

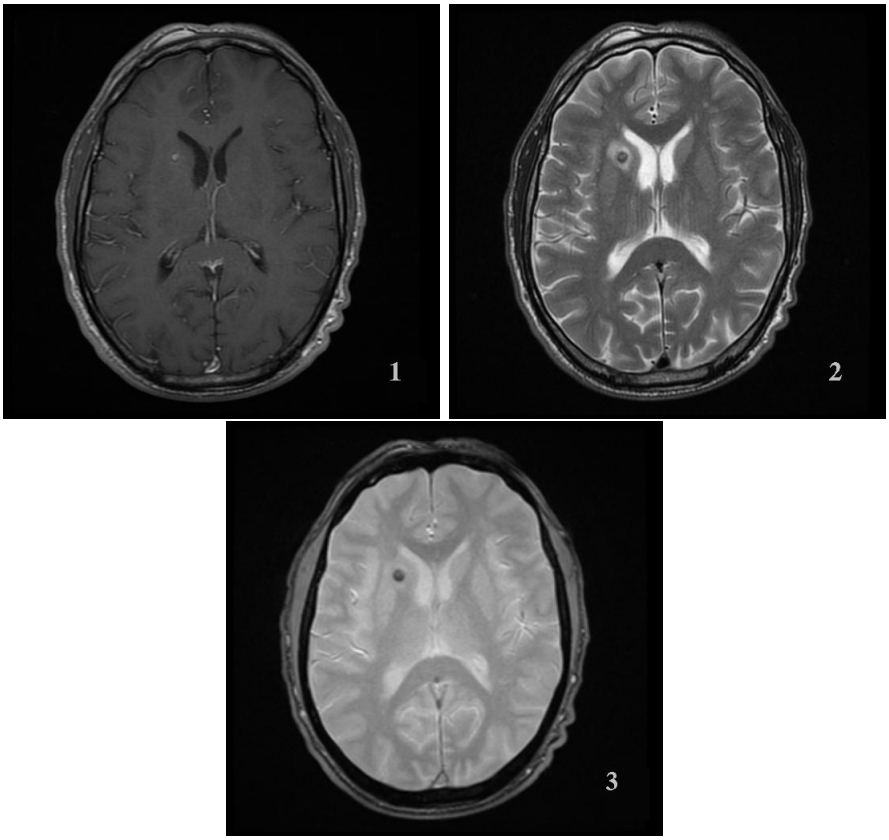


Figure 1.6: T1 weighted (1), T2 weighted (2) and T2* weighted (3) MR images

1.3 Positron Emission Tomography

PET uses radioactive tracer molecules, which contain a positron emitting radioactive atom. After some amount of it is injected into the patient, it distributes in the organs. This radioactive atom emits positrons, which combine with electrons shortly after. The positron-electron pairs emit two photons in opposite directions. Each of these pair of photons are then simultaneously detected, and the image is calculated.

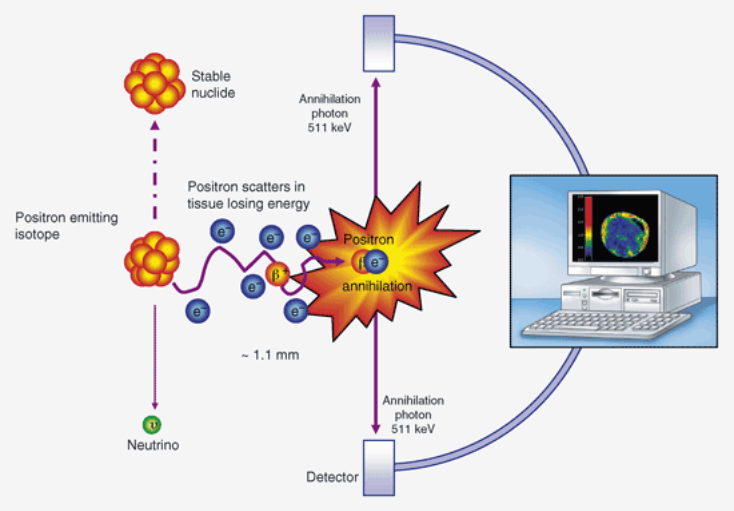


Figure 1.7: PET image acquisition schema

After the detection of the photon pairs, a list of coincidence events are received. Each of these events define a line in space, along which the emission occurred. These lines are used by a computer to calculate the PET image.

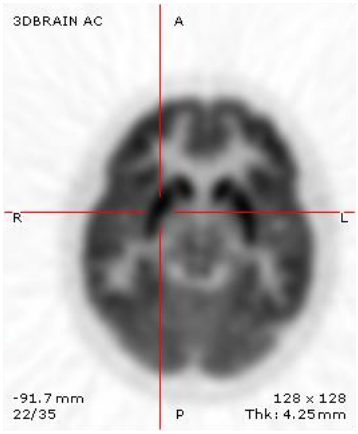


Figure 1.8: PET image

Chapter 2

Processed data

2.1 Logical organization

In this thesis, we limit the definition of volume data to static 3D volume data. This means, that at each point, given by 3 coordinates, there will be a value. Formally, static 3D volume data is a function $f : \mathbb{R}^3 \to \mathbb{R}^n$. In most cases $n = 1$, and we visualize scalar values, but there are times, when we need to visualize data with higher dimensions. Volume data fusions try to solve exactly that problem.

The data is acquired from a hardware, like those mentioned in the introduction. The firmware of those devices arranges the data into *slices*. Each slice is represented by a matrix of the values at each point of the slice.

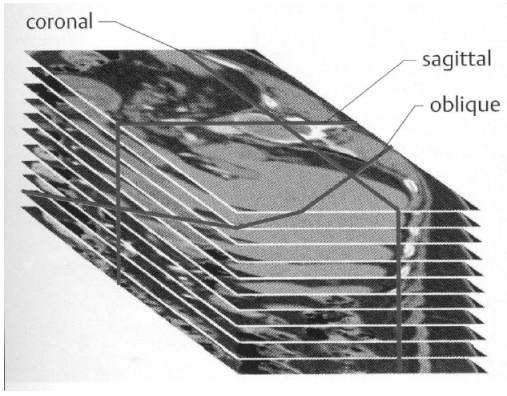


Figure 2.1: Slices and planes

Since we talk about 3D images, the basic units of these images are not pixels but voxels. Each of the voxels have a width, height and depth. In our case, we talk about uniform sampling, so the voxels have the same width and the same height, and as the space between the slices is also uniform, the depth of the voxels is the same as well.

The coordinate system of the volumetric image is set up in the following way:

- Coordinate `x` is parallel with the horizontal plane of the slices (coronal plane)
- Coordinate `y` is parallel with the vertical plane of the slices (sagittal plane)

- Coordinate `z` is the index of the slice (transverse plane)

In practice, the number of the pixel in the horizontal and the vertical directions are the same.

2.2 Data format - DICOM

As computers started playing more and more important roles in medicine, and became the main tool to store data on, a common and standardized format became necessary. In the 90s, the solution came as the DICOM format (Digital Imaging and Communications in Medicine). This standard defines not only the format of the medical data, but also the communication between the client parts of medical software and the DICOM data storage server. A DICOM file is an object file, its internal filetype is in most cases 16 bit JPEG. Each DICOM image has a unique identifier.

The data is stored in files with .dcm extension. The files are divided into pairs of key and value. There are several types of files, each having its own set of tags. When communicating with the server, a query on these tags is made.

Chapter 3

Registration

Registration is the process of bringing the input images into alignment, which means that we find the transformation that transforms them into the same coordinate system.

3.1 Classification of registration methods

Registration methods can be classified according to several criteria that are further discussed in [3], here only a summary of the article is presented.

Dimensionality

There are three types of dimensionalities between the images to be registered to each other:

- 2D to 2D
- 2D to 3D
- 3D to 3D

Since this thesis is about the fusion of volume data, we will focus mainly on 3D to 3D registration.

A further classification according to dimensionality is whether we are talking about spatial registration methods, which are used for images where all the dimensions are spatial ones and no time is involved, or about registration of time series, where monitoring of organ growth, tumor growth or healing periodically in time is emphasized.

Nature of registration basis

- *Extrinsic* - these methods rely on artificial objects - markers - that are attached to the subject, objects that can be easily identified in the picture without any complex

vision algorithm. According to the position of these objects in the images, the transformation can easily be calculated without any iterative optimization. The drawback is, however, that these objects has to be present before acquiring the data, otherwise these algorithms will not work. Most effective are invasive markers, which causes futher difficulties.

- *Intrinsic* - these methods rely only on the acquired image, no artificial objects are required. They can rely on selected points in the picture (called *landmarks*, which can be selected manually or algorithmically), segmented regions (*segmentation based*) or directly on the measured values of the image (voxel property based).

- *Non-image based* - this is the least frequent method group because it relies on the acquisition devices' relative position, which means that the devices have to lie at exact positions in a pre-defined coordinate system. This is usually impossible, however, modern ultrasound scanners, that are hand-held, can be placed at a pre-defined location while the patient is in a CT or MRI scanner. This is, however, a very hard prerequisite, which makes the use of these algorithms very rare.

Nature of transformation

- *Rigid* - only rotation and translation are allowed.

- *Affine* - parallel lines are mapped onto parallel lines.

- *Projective* - lines are mapped onto lines.

- *Curved* or *elastic* - lines are mapped onto curves.

Domain of transformation

The transformation is called *global* if it involves the whole image.
The transformation is called *local* if it involves only certain part or parts of the image.

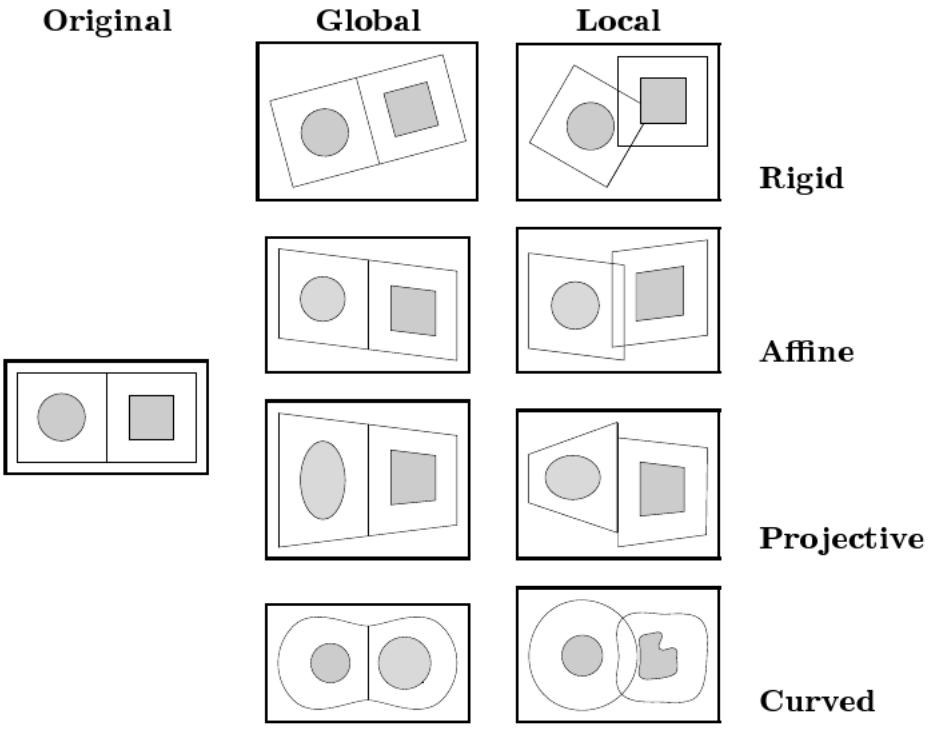


Figure 3.1: Transformation types, taken from [3]

Typically rigid and affine transformations are global while curved transformations are local. Projective transformations are rare in medical imaging since it does not have a typical physical basis so most often it is used as a simplification of curved transformations.

Interaction

There are three levels of interaction that are distinguished for image registration. *Automatic* methods are those when the user supplies only the image data to the algorithm, and it calculates the transformation on its own without any further input from the user. *Interactive* methods are those when the user does the registration by supplying the transformation parameters beside the image data, and the software only visualizes the result. *Semi-automatic* are those when the algorithm does the registration, but the user either initializes the algorithm or gives feedback to the algorithm by accepting or rejecting each result it gives.

Optimization

Finding the required parameters of the registration can be done in two ways:

- Computation
- Search

Computing the parameters directly is possible only in exceptional cases. In case of searching, the problem of finding the parameters is usually formulated as a mathematical function that needs to be optimized with methods like the well-known simplex method, the Levenberg-Marquardt method or the Powell method. This optimization can take very long, so several improvements have been developed, like multi-resolution approaches when the optimization is first performed on a low-resolution image, and then at a higher-resolution image the registration parameters that are to be optimized can be limited.

Modalities

1. Monomodal - the images to be registered belong to the same modality
    - CT
    - MR
    - PET
    - SPECT
    - X-ray
2. Multimodal - the images to be registered belong to different modalities
    - CT - MR
    - CT - PET
    - CT - SPECT
    - PET - MR
    - SPECT - MR
    - TMS - MR
    - X-ray - CT
    - X-ray - MR
3. Modality to model - only one image is involved, the second input is a model
    - CT
    - MR
    - SPECT
    - X-ray
4. Patient to modality - only one image is involved, the second input is the patient himself/herself
    - CT
    - MR
    - PET
    - X-ray

Subject

- Intrasubject - all the images to be registered to each other are acquired from the same patient.
- Intersubject - the images to be registered to each other are acquired from different patients.
- Atlas - the images to be registered to each other are acquired from one patient and the rest by summarizing data from an image information database containing many patients.

Object

1. Head
    - Brain or skull
    - Eye
    - Dental
2. Thorax
    - Entire
    - Cardiac
    - Breast
3. Abdomen
    - General
    - Kidney
    - Liver
4. Pelvis and perineum
5. Limbs
    - General
    - Femur
    - Humerus
    - Hand
6. Spine and vertebrae

3.2 Registration validation

The validation of a registration means a lot more than just accuracy. maintz-viergever presents a list of requirements that are expected from registration:

- Precision - typical systematic error of the output when the input is idealized
- Accuracy - actual error of the output at a specific image
- Robustness/stability - small variations in the input should cause small variations in the output
- Reliability - expected behavior for reasonable input
- Resource requirements - material and effort amount should be reasonable
- Algorithm complexity - the algorithm should registration within the time constraints of clinical environment
- Assumption verification - assumptions on reality should be verified in practice
- Clinical use - should satisfy some kind of clinical need

Fulfilling all of these requirements is almost impossible for one application, but effort should be made to satisfy as many of them and at as high level as possible.

3.3 Registration for volume data fusions

Since the goal of this thesis is to provide methods of fusion visualization, a fairly simple set of criteria was chosen for the registration part. The methods used for registration in this thesis were greatly inspired by [6].

- The dimensionality of registration is 3D to 3D. Spatial registration and time series registration is also possible.
- Since we do not have any prior knowledge nor can we have any prior requests concerning the image acquisition, the registration basis can only be intrinsic.
- Rigid transformations will be considered only.
- There is an option for either an interactive way of providing transformation parameters or an automatic way of registering images, repectively a semi-automatic registration by first providing manual transformation parameters, and then executing automatic fusion that starts with the given parameters.
- For optimization, the Powell algorithm will be used.
- A wide variety of multimodal and monomodal registrations are supported.
- Intrasubject, intersubject and atlas registrations are possible.
- Because of the restriction of rigid body transform, images of rigid body parts (like head) are supported only.

3.3.1 Rigid body transformation - mathematical basics

For describing the mathematics of rigid transformation, first we need to define the term *homogeneous coordinates*, which makes it possible to express the rigid transformation with matrices. Homogeneous coordinates of a point $[X, Y, Z]$ in space are $[x, y, z, w]$ for which holds that

$$X = \frac{x}{w}, Y = \frac{y}{w}, Z = \frac{z}{w}, w \neq 0$$

Using the above definition of a point $P = [x, y, z, w]$, we can linear transform it into another point $P^{'} = [x^{'}, y^{'}, z^{'}, w^{'}]$ using transformation 4-by-4 matrix M in the following way:

$$P^{'} = M * P$$

meaning

$$\begin{bmatrix} x^{'} \\ y^{'} \\ z^{'} \\ w^{'} \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

It is very important that transformations can be combined by applying their transformation matrices one-by-one to the original point. If the combined transformation is to be applied more than once, the included elementary transformations' matrices should be multiplied by each other and the resulting matrix applied to the points instead of applying the matrices separately each time. This way, the number of multiplications are reduced, and the received results are equivalent in both ways, since matrix multiplications (if possible) are associative.

$$(((P * M1) * M2) * M3) = P * (M1 * M2 * M3)$$

As it was already stated before, rigid body transform only allows rotations and translations to take place. A list of elementary transform matrices used for this purpose follows:

- Rotation around axis x by α degrees

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

- Rotation around axis y by β degrees

$$\mathrm{Rot}_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation around axis z by γ degrees

$$\mathrm{Rot}_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Translation by vector (T_x, T_y, T_z)

$$\mathrm{Trans}(T_x, T_y, Tz) = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The inverse of these transformations can be calculated easily:

- $\mathrm{Rot}_x(\alpha)^{-1} = \mathrm{Rot}_x(-\alpha)$
- $\mathrm{Rot}_y(\beta)^{-1} = \mathrm{Rot}_y(-\beta)$
- $\mathrm{Rot}_z(\gamma)^{-1} = \mathrm{Rot}_z(-\gamma)$
- $\mathrm{Trans}(T_x, T_y, T_z)^{-1} = \mathrm{Trans}(-T_x, -T_y, -T_z)$

3.3.2 Interpolation

Most of the times when we apply some kind of transformation to a point in space, tha resulting point will not lie in the integer domain of the coordinate system of the reference image. This means that *interpolation* becomes necessary so that we would have a value to further manipulate or observe.

Let the voxels be $V_{0,0,0}, V_{0,0,1}, ..., V_{1,1,1}$ that lie in the neighborhood of the transformed $Y = M * X$ point (X is the original point, M is the transformation marix and Y is the resulting point) while $V_{i,j,k}$ means the voxel lying at $[\lfloor Y_x \rfloor + i, \lfloor Y_y \rfloor + j, \lfloor Y_z \rfloor + k]$ where $i, j, k \in 0, 1$.

Nearest Neighbor interpolation

Let $\|.\|$ mark the norm of a vector in space. The value of the transformed point will be that of the neighboring voxels, which lies closest to it. Mathematically, this criteria can be formulated as follows:

$$f(M * X) = f(V_{i,j,k=0,1} | min(\|V_{i,j,k} - M * X\|))$$

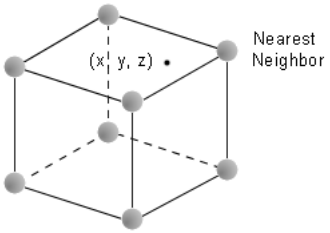


Figure 3.2: Nearest Neighbor interpolation, taken from "www.hyperzoid.com"

Trilinear interpolation

Trilinear interpolation is nothing more than applying linear interpolation for each of the three dimensions.

Linear interpolation of the value of point x that lies between x_0 and x_1 is calculated in the following way:

$$f(x) = f(x_0) + (f(x_1 - f(x_0) * \frac{x - x_0}{x_1 - x_0}$$

In our case, the neighboring voxels' distances equal to 1, so $x_1 - x_0 = 1$, which means that the equation can be simplified:

$$f(x) = f(x_0) * (x_1 - x) + f(x_1) * (x - x_0)$$

Bilinear interpolation we get if after applying linear interpolation for the point's first coordinate for both possible second coordinates, we apply linear interpolation for the two resulting points as well. The result can be calculated with the following formula:

$$\begin{aligned} f(x,y) = f(x_0,y_0) * \frac{(x_1 - x)(y_1 - y)}{(x_1 - x_0)(y_1 - y_0)} \\ + f(x_1,y_0) * \frac{(x - x_0)(y_1 - y)}{(x_1 - x_0)(y_1 - y_0)} \\ + f(x_0,y_1) * \frac{(x_1 - x)(y - y_0)}{(x_1 - x_0)(y_1 - y_0)} \\ + f(x_1,y_1) * \frac{(x - x_0)(y - y_0)}{(x_1 - x_0)(y_1 - y_0)} \end{aligned}$$

Trilinear interpolation basically interpolates the values that we receive by two bilinear interpolations in the planes above and below the given point by applying one final linear interpolation. Mathematically, it is formulated in a similar way to the others above, only with greater dimension, so the formula of calculating the result in our case - so in the case of unit distance voxels - will be presented:

$$\begin{aligned} V_{0,0,0} * (1 - i) * (1 - j) * (1 - k) + \\ V_{1,0,0} * i * (1 - j) * (1 - k) + \\ V_{0,1,0} * (1 - i) * j * (1 - k) + \end{aligned}$$

$$V_{0,0,1} * (1-i) * (1-j) * k +$$
$$V_{1,0,1} * i * (1-j) * k +$$
$$V_{0,1,1} * (1-i) * j * k +$$
$$V_{1,1,0} * i * j * (1-k) +$$
$$V_{1,1,1} * i * j * k$$

The next figure helps understand how this interpolation works. First, the values at the red squares are interpolated, which are then used to interpolate the values at the green triangles, which are finally used to interpolate the value at the blue dot.

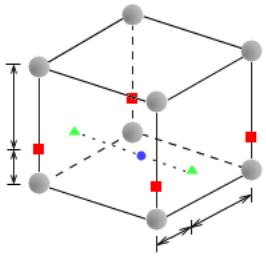


Figure 3.3: Trilinear interpolation, taken from "www.hyperzoid.com"

3.3.3 Criterion

The criterion is some kind of function with one or more parameters that evaluates into a real number. The goal is to find the extreme (minimum or maximum) of this criterion function.

Mutual Information

One of the most frequently used criterion for matching of medical images is *mutual information*. It is the measure of statistical dependence between two or more datasets. A very good description of this method is given in [4], here only a summary is presented.

Entropy To calculate and use mutual information, first we have to define a measurement of information. The *Hartley entropy* was the first definition, which considered a message to be n symbols with s possibilities for each symbol. This means that there can be s^n different messages. Hartley's definition wanted entropy to increase with message length. The value s^n would have the disadvantage of increasing exponentially, which was an unwanted property (a linearly proportional measure was needed), and it lead to the equation:

$$H = n * \log s = \log s^n$$

This definition says that the larger the number of possible messages, the more information one of them brings.

The problem with the Hartley entropy is that it assumes that all the messages are equally likely, which is not a realistic requirement. That is what the *Shannon entropy* was called to

cure. If there are m possible events, each of them occurring with probability $p_1, p_2, p_3, ..., p_m$ respectively, the Shannon entropy is defined as

$$H = \sum_i p_i * \log\frac{1}{p_i} = -\sum_i p_i * \log p_i$$

If we apply the Hartley entropy's assumption of equally probable events, we get

$$H = -\sum \frac{1}{s^n} * \log\frac{1}{s^n} = \sum \frac{1}{s^n}\log s^n = \log s^n$$

which is exactly the Hartley entropy.

According to this definition of Shannon entropy, the amount of information brought by an event ($\log\frac{1}{p_i}$) is reversely proportional to the probability of its occurrence. Weighting these informations by the probabilities and summing them together, we get that entropy can be interpreted as the *average information*.

Another intrepretation of entropy is the *measure of uncertainty*. From the equation, it is clear that the entropy is the highest if all the events are equally probable. This means that we are absolutely uncertain about which event will occur next.

Finally, a third interpretation of entropy is the *dispersion of probability distribution*. The distribution of probabilities with one sharp peak leads to a low entropy, whereas a distribution that is equally dispersed, leads to a high entropy.

Entropy can be calculated for images as well, in which case the events will be the grayscale values of the pixels, and the probabilities will be the percentage of appearance of each grayscale value in the picture.

Joint Entropy In the problem of registration, the *joint entropy* is an important term. It is calculated using the joint histogram of two (or more) images. The joint histogram contains the number of occurrences of each pairs of grayscale values divided by the total number of entries. The Shannon joint entropy is defined as

$$H_{joint} = -\sum_{i,j} p(i,j) * \log p(i,j)$$

The joint entropy gives the amount of dispersion of the joint histogram. The transformation that minimizes this dispersion registers the best the images. The following figure is taken from [4], and illustrates this property of the joint histogram and the joint entropy (which is written under each picture). The picture on the left shows the histogram of images that are registered correctly, the others show the situation when one of the images is rotated by 2, 5 and 10 degrees respectively.

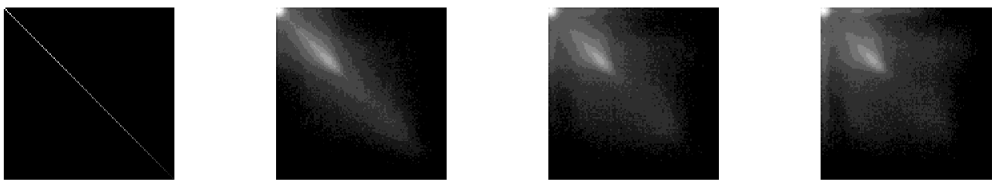


Figure 3.4: Joint histograms of registered and misregistered images

Mutual Information

Mutual Information has three well-known definitions. Each of them explains some of the properties of the term itself.

- $I(A,B) = H(B) - H(B|A)$ - where $H(B)$ is the Shannon entropy of image B and $H(B|A)$ is the conditional entropy. This interpretation says that mutual information is "the amount of uncertainty about image B minus the uncertainty about B when A is known." [4]
- $I(A,B) = H(A) + H(B) - H(A,B)$ - this definition says that mutual information maximization (in part) means the minimization of joint entropy.
- $I(A,B) = \sum_{a,b} p(a,b) * \log\frac{p(a,b)}{p(a)p(b)}$ - this definition measures the distance between the joint distribution of images ($p(a,b)$) and the joint distribution in case of independence between the images ($p(a)p(b)$). This value is the smallest when the images are in fact independent, which means misregistered in our case.

Here is a list of properties that apply to mutual information:

- $I(A,B) = I(B,A)$ - mutual information is symmetric
- $I(A,A) = H(A)$ - mutual information with the same image is the entropy of the image
- $I(A,B) \leq H(A)$
  $I(A,B) \leq H(B)$
- $I(A,B) \geq 0$
- $I(A,B) = 0$ if and only if A and B are independent - no knowledge is gained from one about the other.

Normalized Mutual Information (NMI)

The problem with mutual information is that its effectivity is reduced when the overlap is small between the two images since the number of samples is decreased as well. Furthermore, misregistration can actually increase mutual information when the areas of objects and the background even out and the entropies grow faster then the joint entropy. A normalized version of mutual information follows, which will be used as criterion for our registration algorithm:

$$NMI(A,B) = \frac{H(A) + H(B)}{H(A,B)}$$

3.3.4 Optimization

Powell's direction set method

Powell's method has been chosen as the optimization method for finding the maximum of mutual information between images. This method is well described in [5], which also provides an implementation of it that has been used as an external module for the implementation part of the thesis. Here is a shorter description of the method based on the above mentioned

text.

We have a function f with parameter $\mathbf{P}$, which is an N dimensional point in space. Our goal is to find the minimum (or maximum - it is irrelevant because minimum and maximum can be interchanged by multiplying the function with -1 or by calculating $\frac{1}{f}$) of this function, and give back the parameter's coordinates. This is done by selecting a vector $\mathbf{n}$, and minimizing the function along that vector, then selecting another one, and minimizing along that vector, and so on, until the optimum is found. For this procedure to work, a one dimensional sub-algorithm is required that can be defined as follows (from [5]):
"*linmin* : Given as inputs the vectors $\mathbf{P}$ and $\mathbf{n}$, and the function f, find the scalar λ that minimizes $f(\mathbf{P} + \lambda\mathbf{n})$. Replace $\mathbf{P}$ by $\mathbf{P} + \lambda\mathbf{n}$. Replace $\mathbf{n}$ by $\lambda\mathbf{n}$. Done."

The most simple way to use the above described procedure is to take the unit vectors as the *set of directions*: $\mathbf{e}_1, \mathbf{e}_2, ...\mathbf{e}_N$, and let the algorithm cycle through them, and optimize in these directions.

The problem with this approach is that the unit vectors can be very inefficient if the function value decreases much more in other directions while only little in these selected unit vector directions, which can be very often the case. This is the reason why we need a better set of directions.

This "better set" is found iteratively, updating the initial set of vectors, which remains the set of unit vectors. We need direction vectors, along each of which minimization does not spoil the value minimized in other directions. This set of vectors is called *conjugate directions*.

Ideally, the an algorithm would come up with N linearly independent, mutually conjugate vectors. Unfortunately, most of the time there is a tradeoff between these conditions for the vectors along which optimization occurs.

Powell discovered a method that returns mutually conjugate directions, which converges quadratically:
Initialize the set of directions $\mathbf{u}_i = \mathbf{e}_i$ for $i = 1, ..., N$, and then repeat the following steps until the function stops decreasing:

- Save starting position $\mathbf{P}_0$
- $\forall i = 1, ..., N$: find the minimum starting from $\mathbf{P}_{i-1}$ along direction $\mathbf{u}_i$, and call this point $\mathbf{P}_i$.
- $\forall i = 1, ..., N-1 : \mathbf{u}_i = \mathbf{u}_{i+1}$.
- $\mathbf{u}_N = \mathbf{P}_N - \mathbf{P}_0$.
- Find the minimum starting from $\mathbf{P}_N$ along direction $\mathbf{u}_N$ and call it $\mathbf{P}_0$.

The disadvantage of this method is that it discards one linearly independent vector from the set, and adds one that might be linearly dependent on the others. This means that after a number of iterations, the algorithm cannot check certain directions, which might result in returning an incorrect optimum.

There are numerous ways to fix this problem, like:

- After N or $N + 1$ iterations, reinitialize the direction set to unit vectors.
- Reset the direction set to calculated principal directions of the original matrix of direction set - singular value decomposition algorithm (see [5]).

- Abandon the property of quadratic convergence, and use heuristics.

[5] describes and implements the last one of the listed solutions, and this implementation was used for the practical part of this thesis, so that will be explained here. The basic idea is that after taking $\mathbf{P}_N - \mathbf{P}_0$ as a new direction vector, we throw away from the set the direction that gave the *largest decrease* of f. Although that direction was the best of the previous ones, it is probably the major component of the newly added direction.

However, there are occasions when not adding any new direction is the best thing to do. If we define

$$f_0 \equiv f(\mathbf{P}_0), f_N \equiv f(\mathbf{P}_N), f_E \equiv f(2\mathbf{P}_N - \mathbf{P}_0)$$

and we further define δf as the magnitude of the largest decrease along a single direction of the current set, we can set two conditions when not to add/discard any of the current directions:

1. $f_E \geq f_0$ - this means that the average direction $\mathbf{P}_N - \mathbf{P}_0$ will not bring any more significant decrease.

2. $(f_0 - 2f_N + f_E)[(f_0 - f_N) - \delta f]^2 \geq (f_0 - f_E)^2 \delta f$ - the decrease along the new average direction was not significantly influenced by a single direction, or we are very near to the bottom of its minimum.

Chapter 4

Fusion

We have arrived to the main goal of the thesis: volume data fusions. In this section, several fusion methods will be introduced each having its strengths and weaknesses. As it was stated in the beginning, methods should be developed for the fusion of images with different modalities or settings to gain information from all of them as well as for greater number of images acquired at different times to monitor changes. There are several things to decide for each fusion method:

- Use RGB channels or grayscale values
- Visualize all the datasets or a summary of them
- Should the values be calculated by selecting one value out of the images for each pixel, or summarize somehow all the values
- Should only one summary be shown at a time, or should we try to show more of them
- Only the dataset images should be shown, or should we take into consideration the gradients between the images
- We contribute more information to higher values, to lower values, or we do not have any preference about them in the images
- Only the input pixels at the given position are considered for the calculation of an output pixel value or other surrounding pixel values as well.

Keeping these questions in mind, 13 methods have been developed for the fusion of volumetric data. All of the possible answers to these questions have been realized in one or more of the methods described below.

4.1 RGB fusion

This method

- uses RGB channels,

- visualizes either all the datasets, or the first 3 of them, if there are more then that present,
- summarizes all the values by creating an RGB colored pixel of each of them,
- shows one summary at a time,
- does not take into consideration the gradients between images,
- does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

Its mechanism is very simple: it gets as an input 3 datasets (or fewer), modifies their brightness and contrast, and each one of them is assigned to one channel of Red, Green or Blue. The final image is shown as a regular RGB image.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

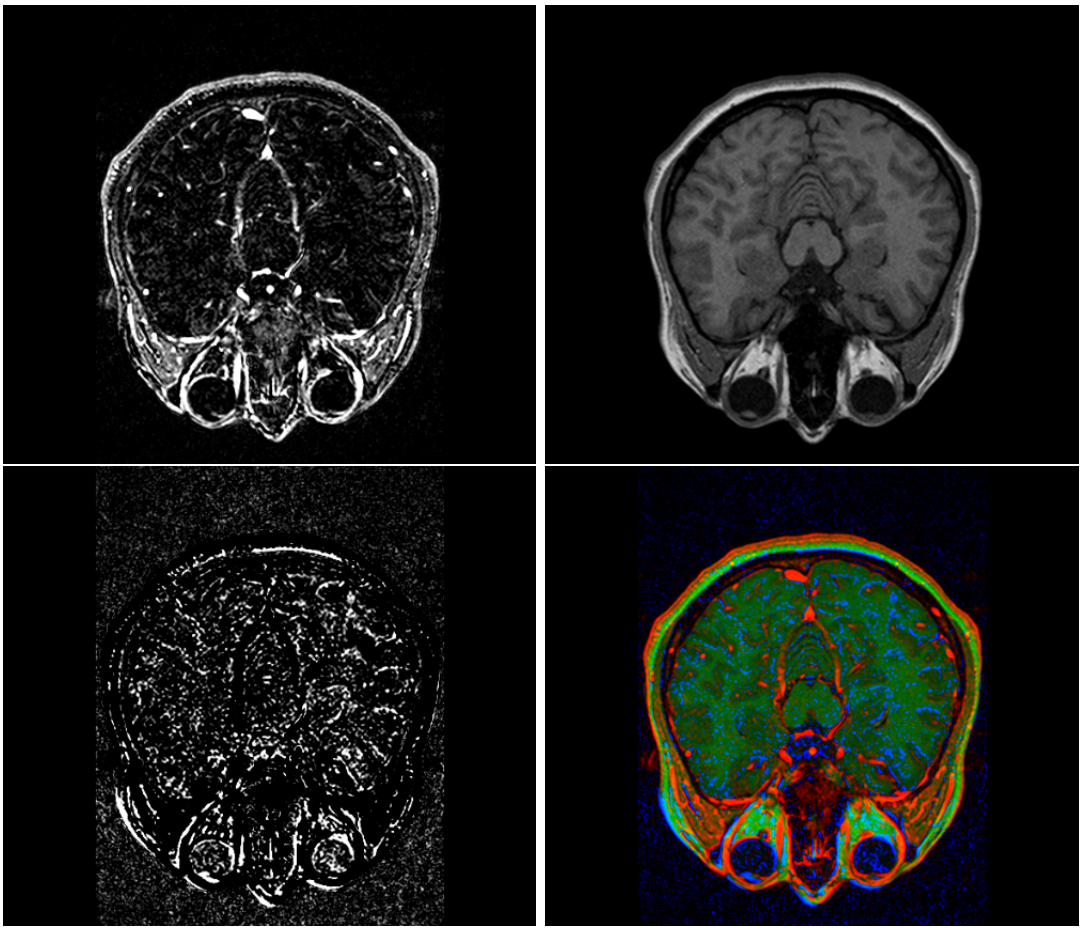


Figure 4.1: RGB image fusion

4.2 Multichannel RGB fusion

This method

- uses RGB channels,
- visualizes all the datasets,
- summarizes all the values by creating an RGB colored pixel by summarizing them,
- shows one summary at a time,
- does not take into consideration the gradients between images,
- does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

This method selects in the color spectrum into N equidistant points (where N is predefined - see the figure below for $N = 12$). It gets as input N datasets (or fewer), modifies their brightness and contrast, and each one of them is assigned to one color of the selected colors of the spectrum. The final image is then created by summing up these colors as RGB values and dividing each channel (Red, Green and Blue) with the number of inputs. The result is again a regular RGB image.

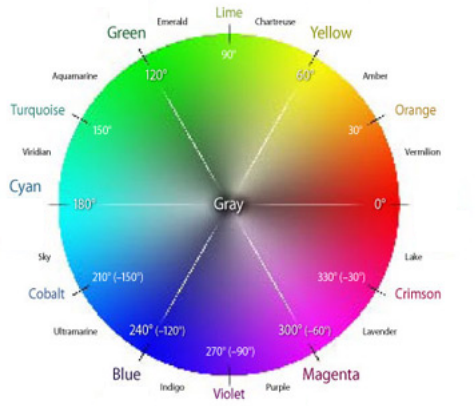


Figure 4.2: 12 equidistant points selected in the Color Spectrum Circle

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion. Not all the 12 channels have been used, so different datasets can be seen as red, orange and yellow, the rest of the channels are not used.

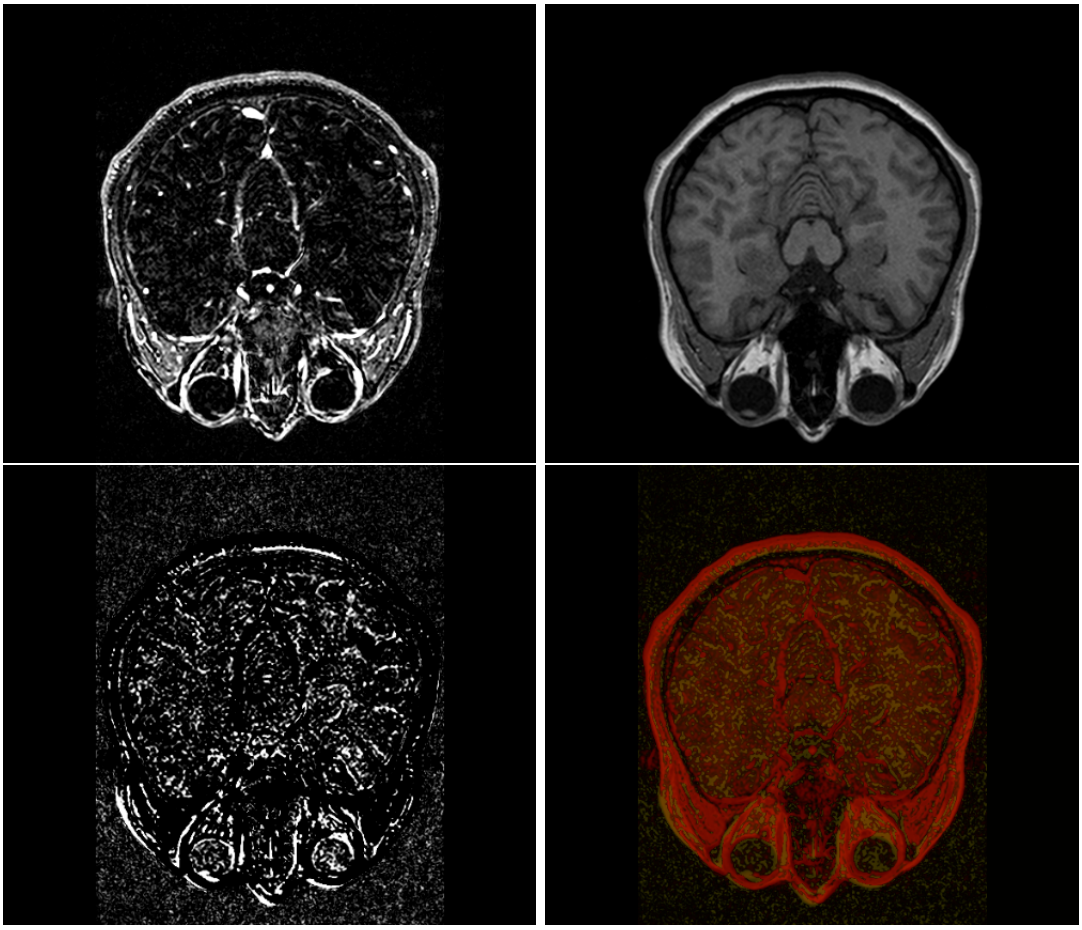


Figure 4.3: Multichannel RGB image fusion

This method works best for several images (like time series) because it shows a lot of channels. The disadvantage is that if N is too great, the channels interfere with each other too much, and it becomes hard to distinguish between them.

4.3 Maximum Intensity fusion

This method

- uses grayscale values,
- visualizes only a summary of the datasets at each pixel,
- selects one value out at each pixel for visualization,
- shows one summary at a time,
- does not take into consideration the gradients between images,
- contributes information to higher value pixels,

- only the input pixels at the given position are tekin into consideration for the given output pixel.

This method works like Maximum Intensity Projection. It selects the highest valued pixel out of the pixels of datasets at the given point, and shows that particular value at that position.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

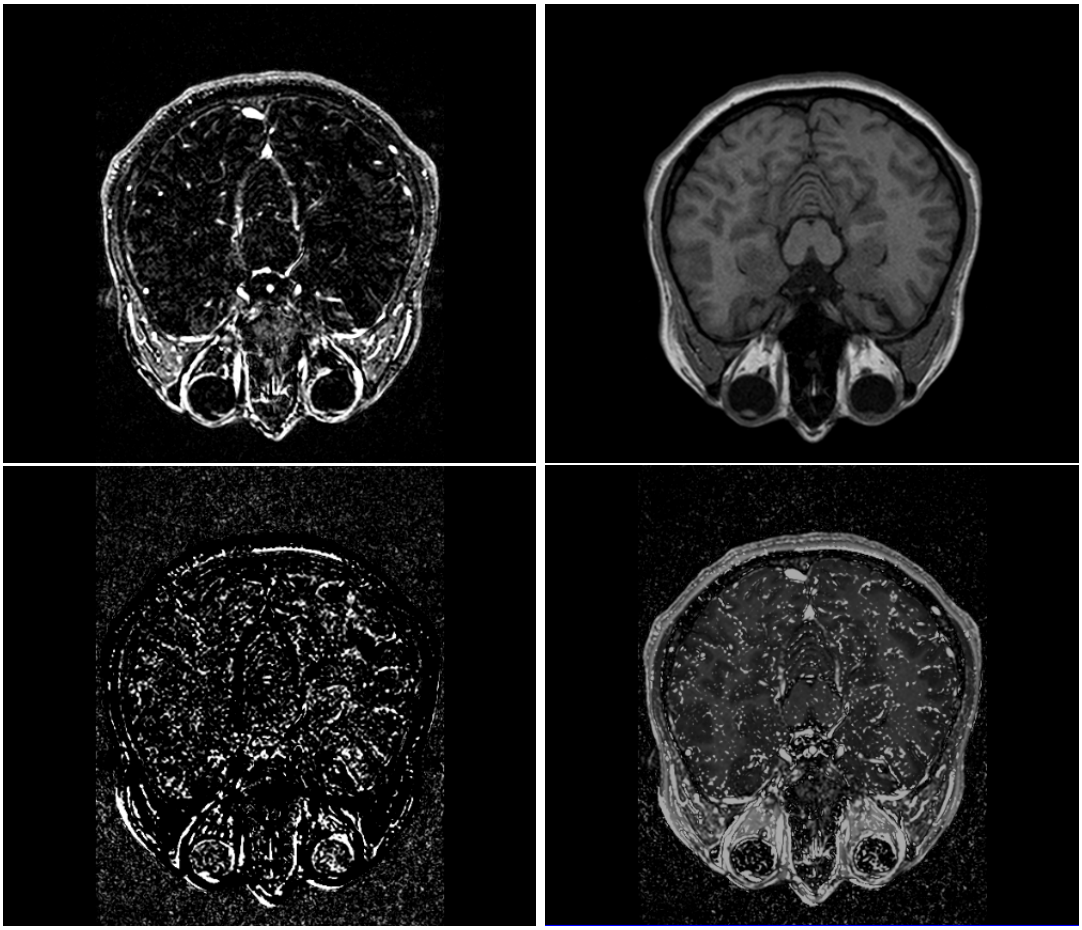


Figure 4.4: Maximum Intensity Fusion

4.4 Minimum Intensity fusion

This method

- uses grayscale values,
- visualizes only a summary of the datasets at each pixel,
- selects one value out at each pixel for visualization,
- shows one summary at a time,

- does not take into consideration the gradients between images,
- contributes information to lower value pixels,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

This method selects the lowest valued pixel out of the pixels of datasets at the given point, and shows that particular value at that position.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

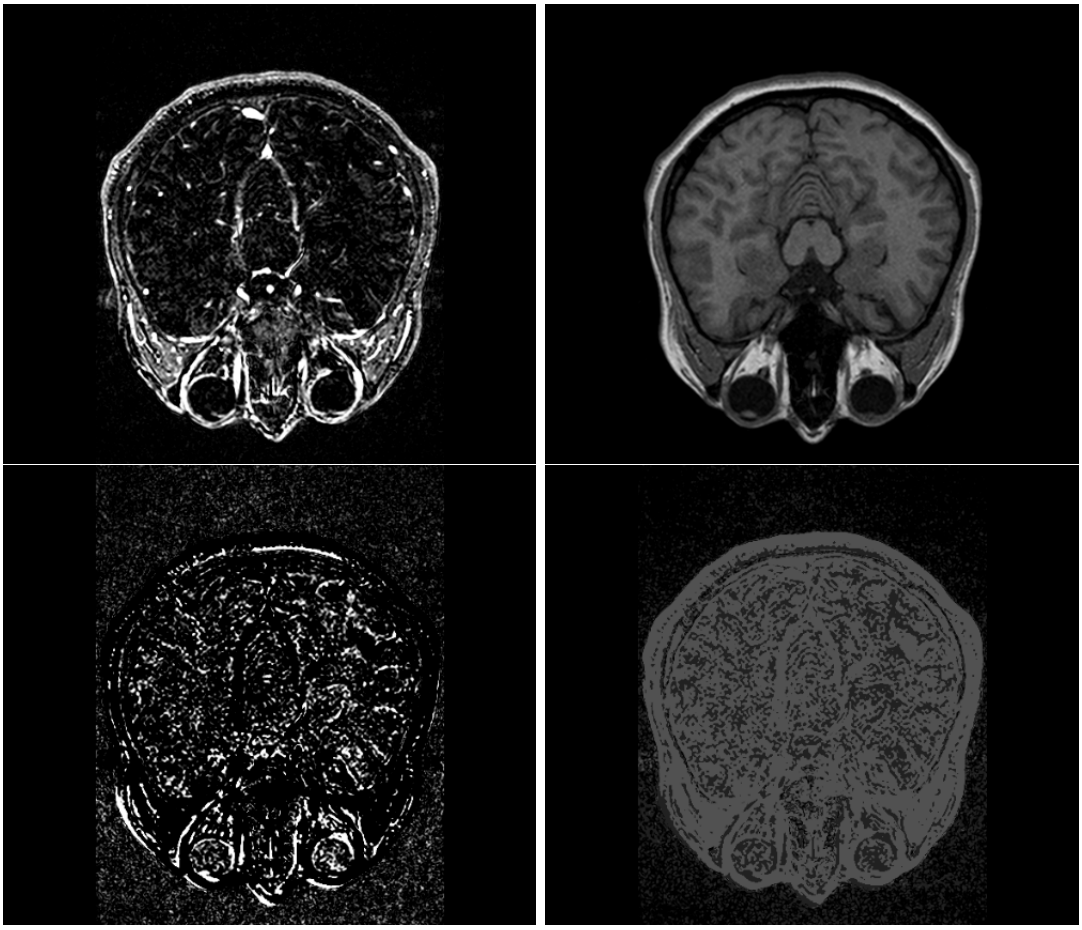


Figure 4.5: Minimum Intensity Fusion

4.5 Average Intensity fusion

This method

- uses grayscale values,
- visualizes only a summary of the datasets at each pixel,
- summarizes the values of all the datasets for visualization,

- shows one summary at a time,
- does not take into consideration the gradients between images,
- does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

This type of fusion calculates the arithmetical average of the pixel values of datasets at the given point, and shows the resulting average value at that position.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

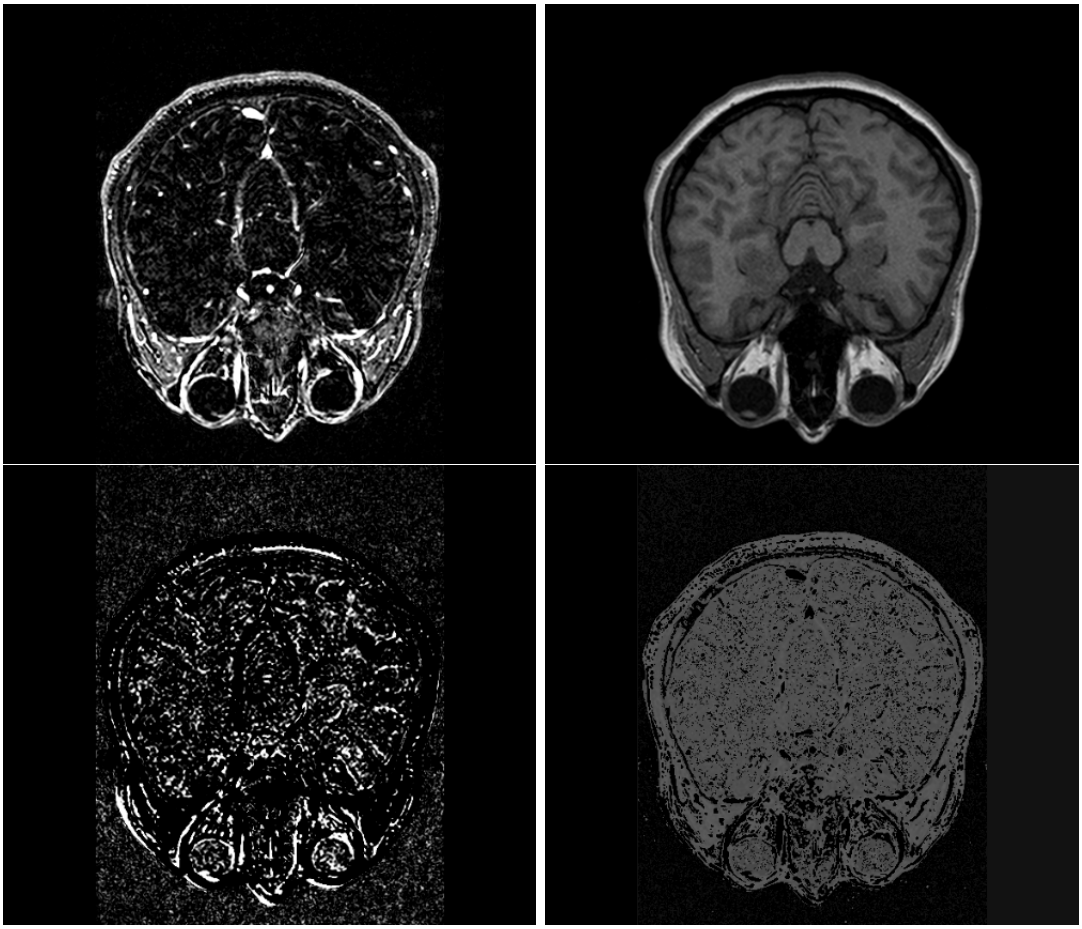


Figure 4.6: Average Intensity Fusion

4.6 Median Intensity fusion

This method

- uses grayscale values,
- visualizes only a summary of the datasets at each pixel,

- selects one value out at each pixel for visualization,
- shows one summary at a time,
- does not take into consideration the gradients between images,
- does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

This type of fusion selects the median out of the pixel values of datasets at the given point, and shows that particular value at that position.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

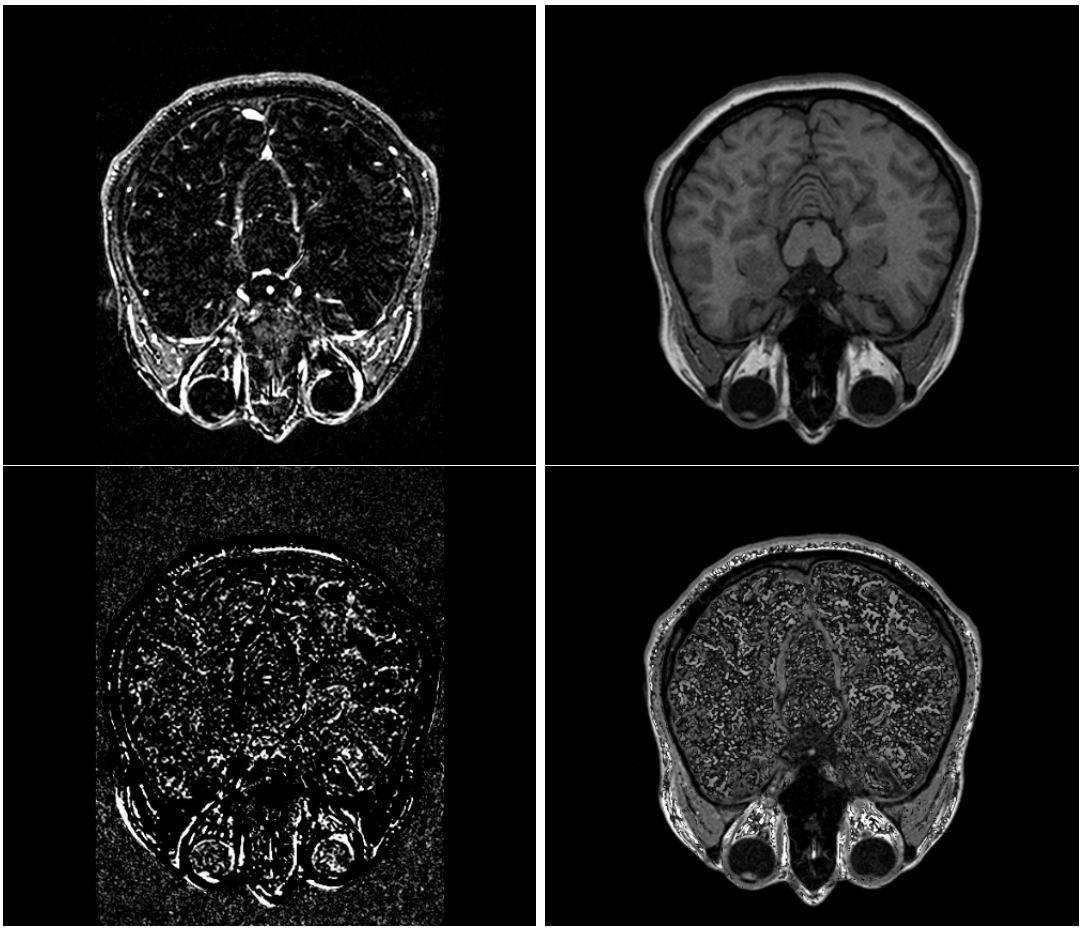


Figure 4.7: Median Intensity Fusion

4.7 RGB Maximum-Average-Minimum fusion

This method

- uses RGB channels,

- visualizes only a summary of the datasets at each pixel,
- summarizes all the values in one channel and selects one value at each pixel in two channels of RGB,
- shows three summaries at a time,
- does not take into consideration the gradients between images,
- one channel selects higher values, one channel selects lower values and one channel does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

The mechanism that this fusion uses is simply merging the results of Maximum, Average and Minimum Intensity fusions by assigning each one of them to one RGB channel (Red, Green, Blue respectively) after equalization.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

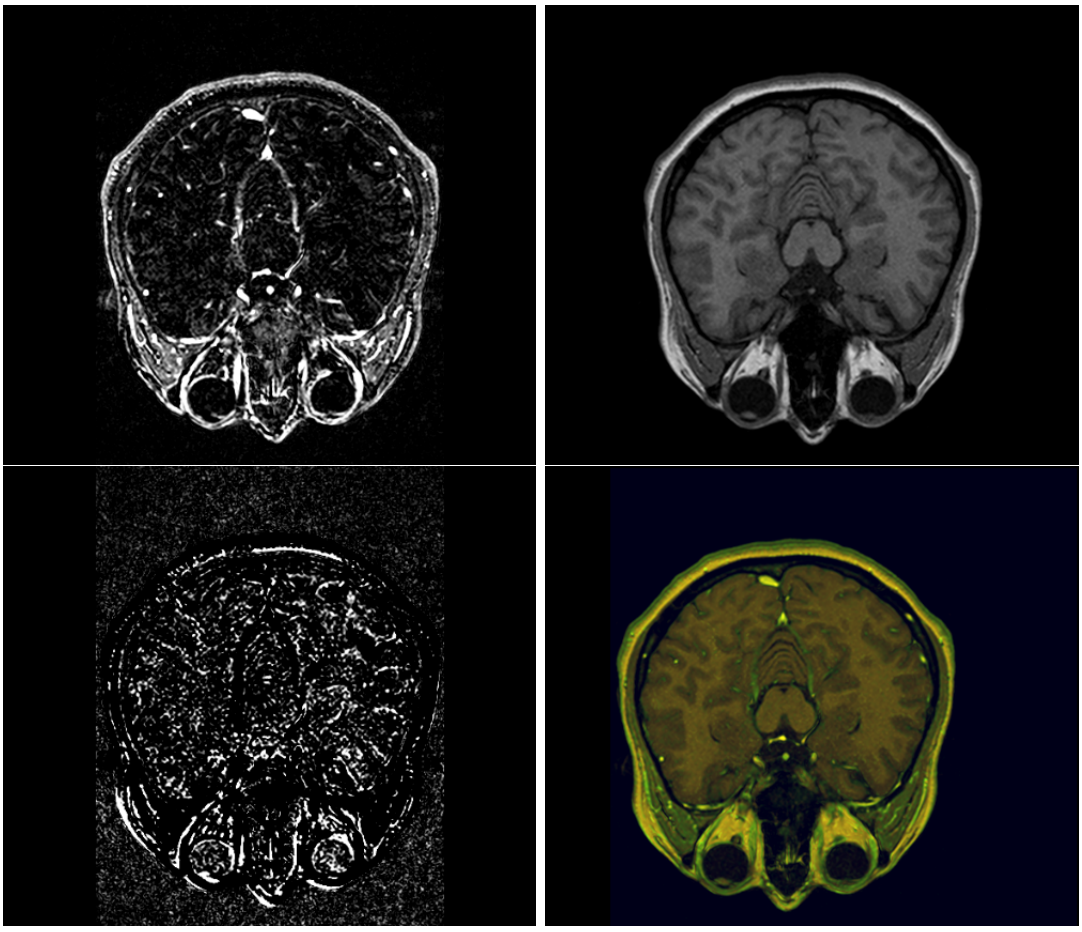


Figure 4.8: RGB Maximum-Average-Minimum fusion

4.8 RGB Maximum-Median-Minimum fusion

This method

- uses RGB channels,
- visualizes only a summary of the datasets at each pixel,
- selects one value at each pixel for each channel of RGB,
- shows three summaries at a time,
- does not take into consideration the gradients between images,
- one channel selects higher values, one channel selects lower values and one channel does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

The mechanism that this fusion uses is simply merging the results of Maximum, Median and Minimum Intensity fusions by assigning each one of them to one RGB channel (Red, Green, Blue respectively) after equalization.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

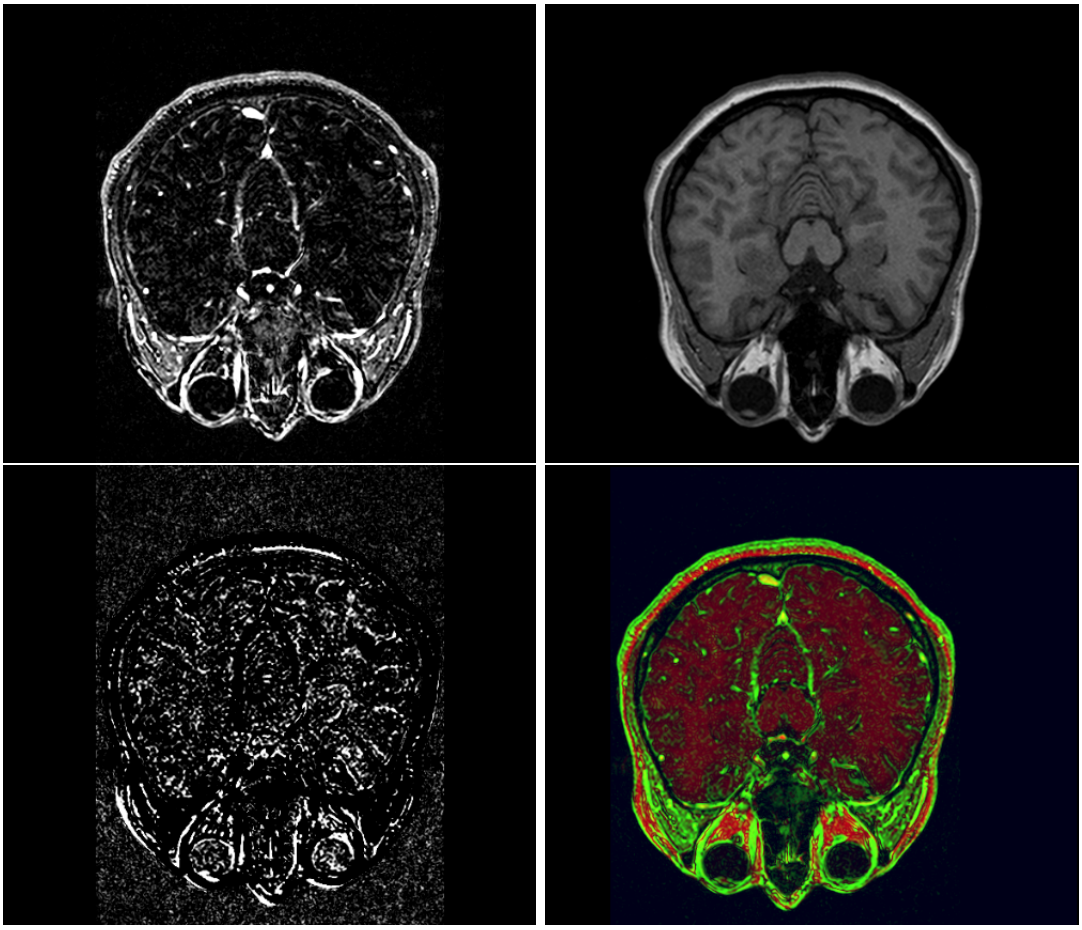


Figure 4.9: RGB Maximum-Median-Minimum fusion

4.9 Multichannel Gradient RGB fusion

This method

- uses RGB channels,
- visualizes all the datasets, but only their gradient,
- summarizes all the gradient values by creating an RGB colored pixel by summarizing them,
- shows one summary at a time,
- uses gradients between images,
- does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

This method selects in the color spectrum into N equidistant points (where N is predefined - see figure 4.2 for $N = 12$). It gets as input N datasets (or fewer), calculates the

gradient between them (decreasing the number of fused values by 1), modifies the gradient values' brightness and contrast, and each one of them is assigned to one color of the selected colors of the spectrum. The final image is then created by summing up these colors as RGB values and dividing each channel (Red, Green and Blue) with the number of inputs. The result is a regular RGB image.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion. Not all the 12 channels have been used, so different datasets can be seen as red, orange and yellow, the rest of the channels are not used.

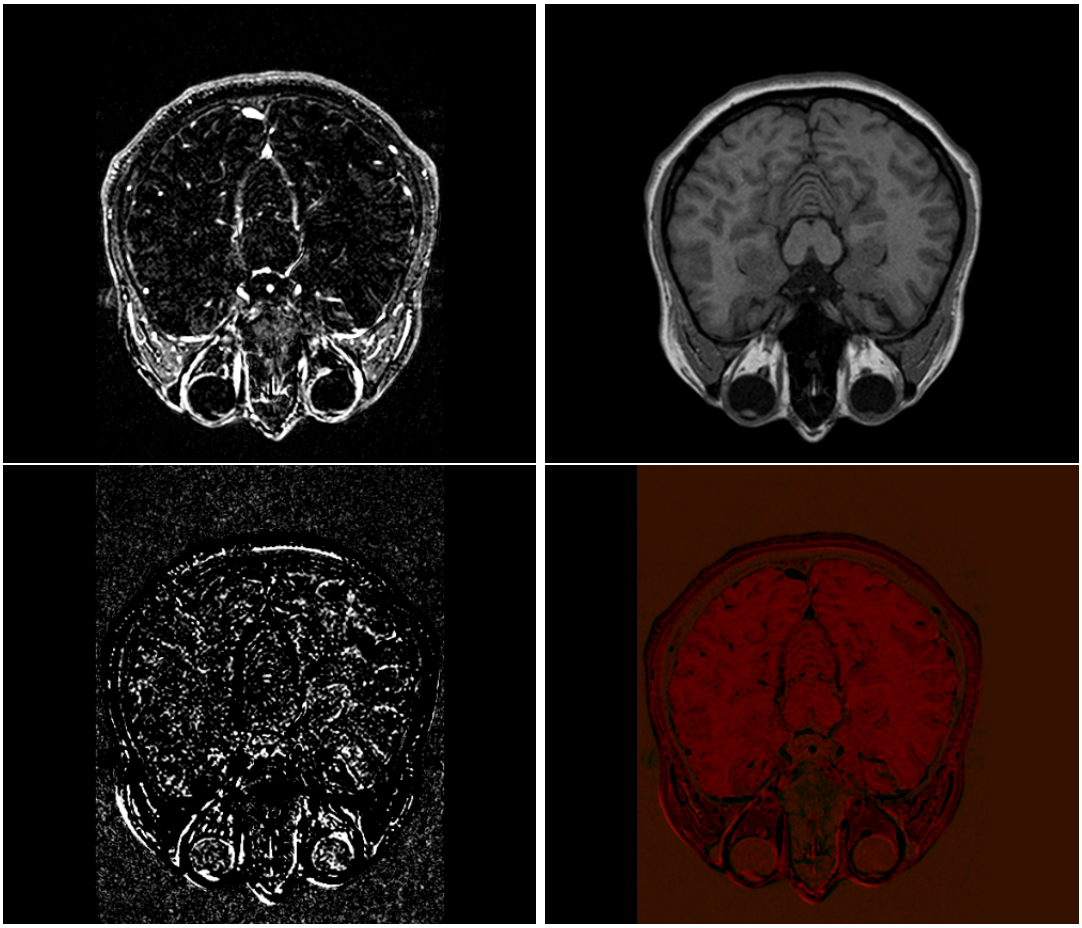


Figure 4.10: Multichannel Gradient image fusion

4.10 RGB Maximum-AverageGradient-Minimum fusion

This method

- uses RGB channels,
- visualizes only a summary of the datasets at each pixel,
- selects one value at each pixel for two channels and summarizes the pixels for one channel of RGB,

- shows three summaries at a time,
- shows the gradients between images in one of the channels,
- one channel selects higher values, one channel selects lower values and one channel does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

The mechanism that this fusion uses is simply merging the results of Maximum, Average Gradient and Minimum Intensity fusions by assigning each one of them to one RGB channel (Red, Green, Blue respectively) after equalization. Average Gradient fusion is similar to Average fusion, only it calculates the average of the calculated gradient values of the images and not the average of image values themselves.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

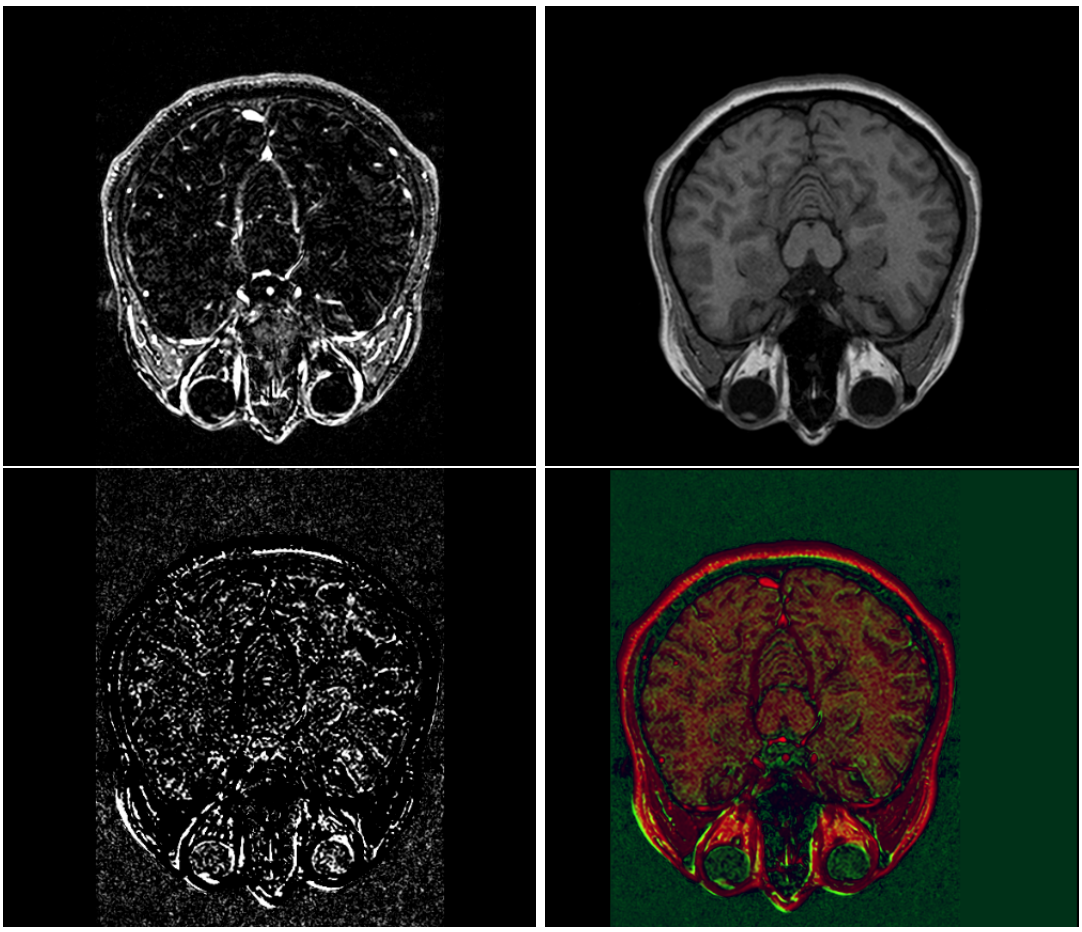


Figure 4.11: RGB Maximum-AverageGradient-Minimum fusion

4.11 RGB Maximum-MedianGradient-Minimum fusion

This method

- uses RGB channels,
- visualizes only a summary of the datasets at each pixel,
- selects one value at each pixel for each channel of RGB,
- shows three summaries at a time,
- shows the gradients between images in one of the channels,
- one channel selects higher values, one channel selects lower values and one channel does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

The mechanism that this fusion uses is simply merging the results of Maximum, Median Gradient and Minimum Intensity fusions by assigning each one of them to one RGB channel (Red, Green, Blue respectively) after equalization. Median Gradient fusion is similar to Median fusion, only it selects the median of the calculated gradient values of the images and not the median of the image values themselves.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

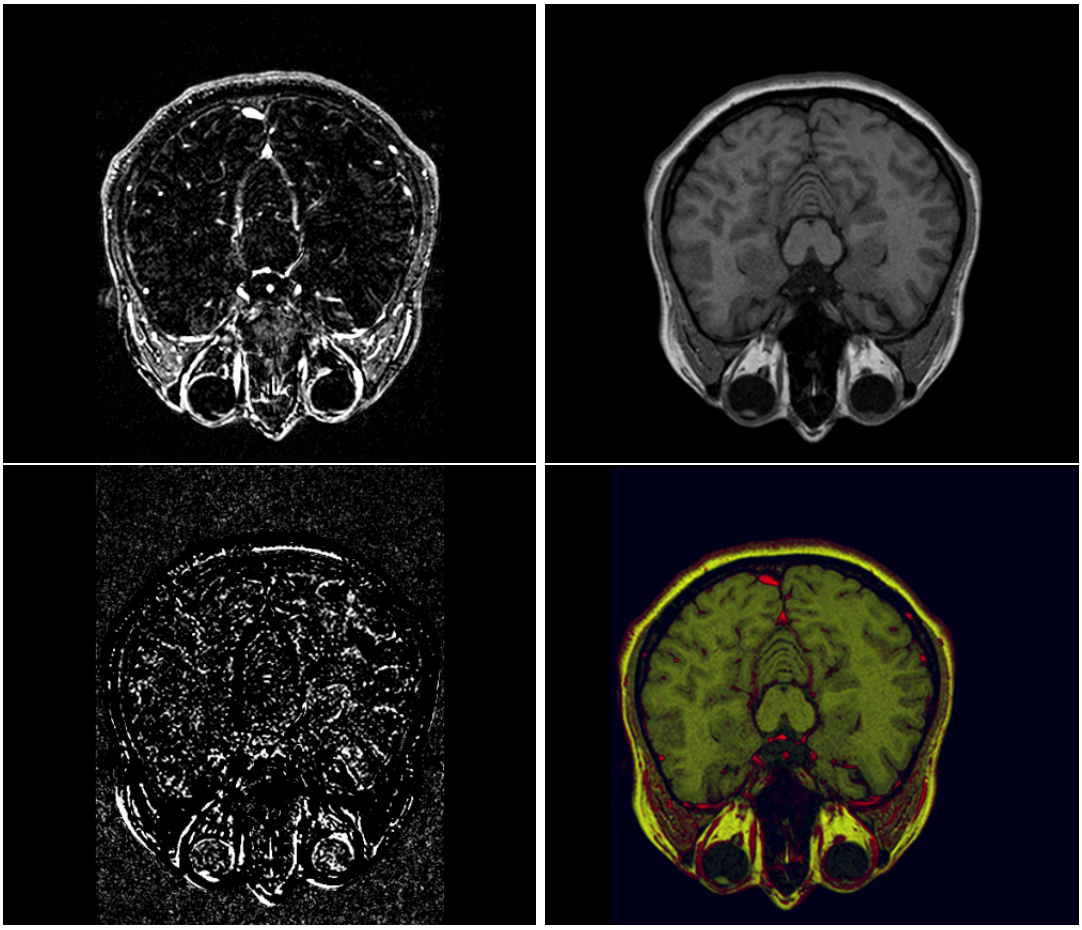


Figure 4.12: RGB Maximum-MedianGradient-Minimum fusion

4.12 RGB Maximum-MaximumGradient-Minimum fusion

This method

- uses RGB channels,
- visualizes only a summary of the datasets at each pixel,
- selects one value at each pixel for each channel of RGB,
- shows three summaries at a time,
- shows the gradients between images in one of the channels,
- one channel selects higher values, one channel selects lower values and one channel does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

The mechanism that this fusion uses is simply merging the results of Maximum, Maximum Gradient and Minimum Intensity fusions by assigning each one of them to one RGB channel (Red, Green, Blue respectively) after equalization. Maximum Gradient fusion is similar to Maximum fusion, only it selects the maximum of the calculated gradient values of the images and not the maximum of image values themselves.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

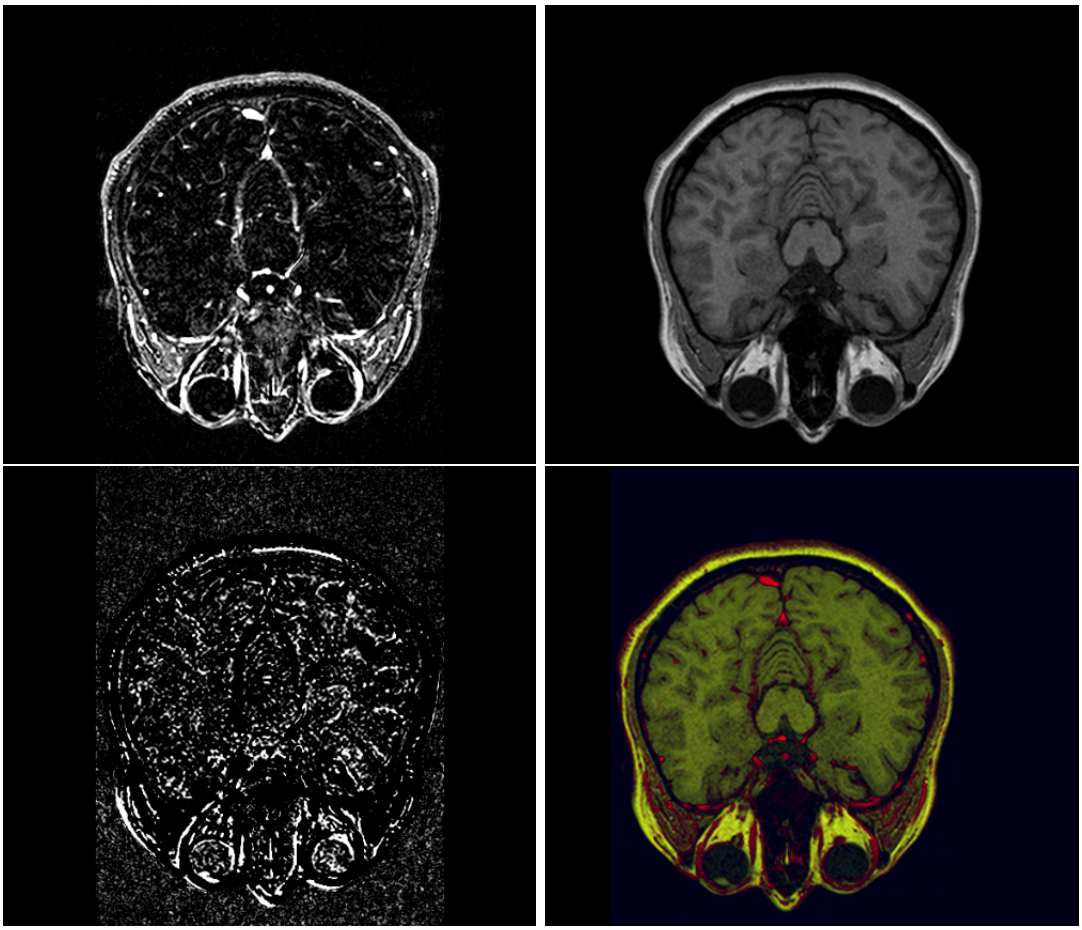


Figure 4.13: RGB Maximum-MaximumGradient-Minimum fusion

4.13 RGB Maximum-MinimumGradient-Minimum fusion

This method

- uses RGB channels,
- visualizes only a summary of the datasets at each pixel,
- selects one value at each pixel for each channel of RGB,
- shows three summaries at a time,

- shows the gradients between images in one of the channels,
- one channel selects higher values, one channel selects lower values and one channel does not have any preference about high and low values,
- only the input pixels at the given position are tekin into consideration for the given output pixel.

The mechanism that this fusion uses is simply merging the results of Maximum, Minimum Gradient and Minimum Intensity fusions by assigning each one of them to one RGB channel (Red, Green, Blue respectively) after equalization. Minimum Gradient fusion is similar to Minimum fusion, only it selects the minimum of the calculated gradient values of the images and not the minimum of image values themselves.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

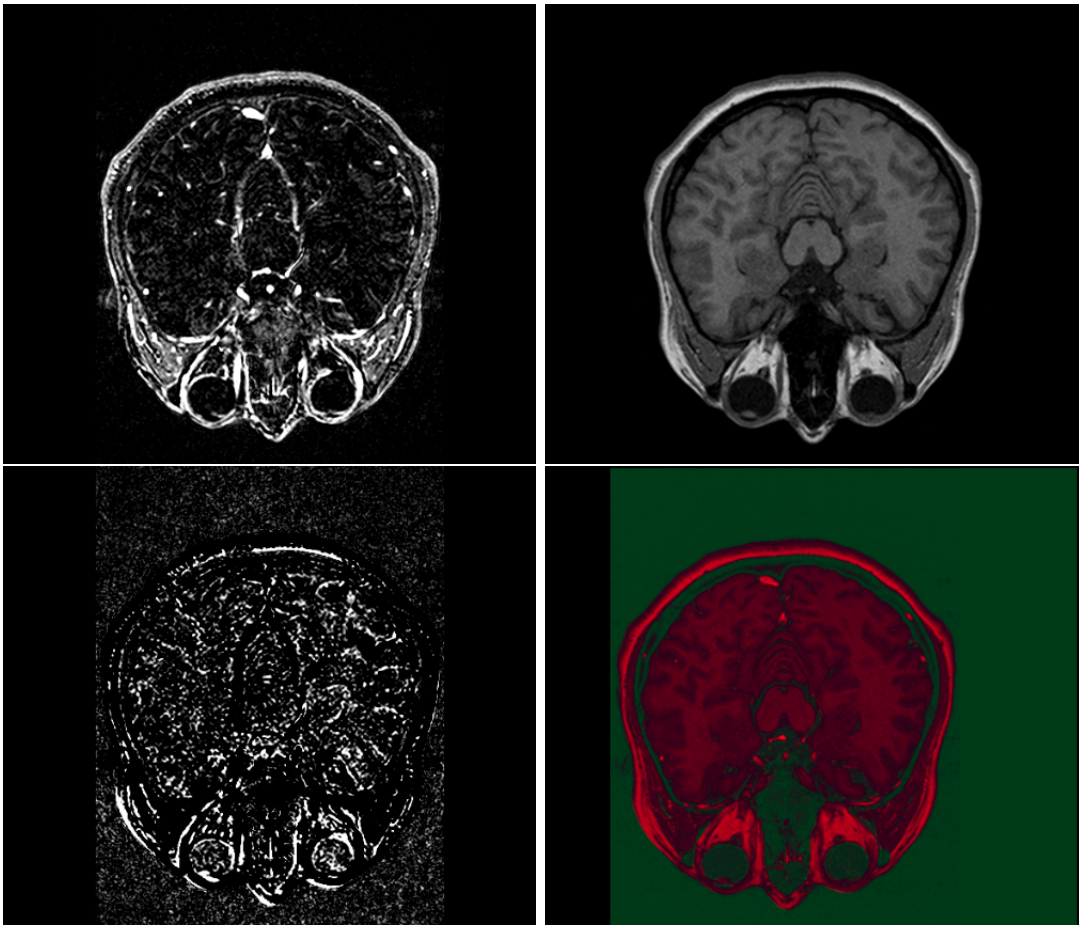


Figure 4.14: RGB Maximum-MinimumGradient-Minimum fusion

4.14 Sobel Operator fusion

This method

- uses grayscale values,
- visualizes only a summary of the datasets at each pixel,
- selects values from the datasets for visualization,
- shows one summary at a time,
- does not take into consideration the gradients between images,
- does not have any preference about high and low values,
- a one-pixel region of the source pixels is taken into consideration when calculating the output value.

This type of fusion applies the Sobel operator for each 3x3 region (boundary conditions are solved by selecting the first input value without calculation), and the resulting pixel will be the input pixel that gives the highest value after the operator application. Since the Sobel operator shows gradients - which means edges, - if the operator's outcome is high, there is probably an edge, which means a detail that should be shown.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

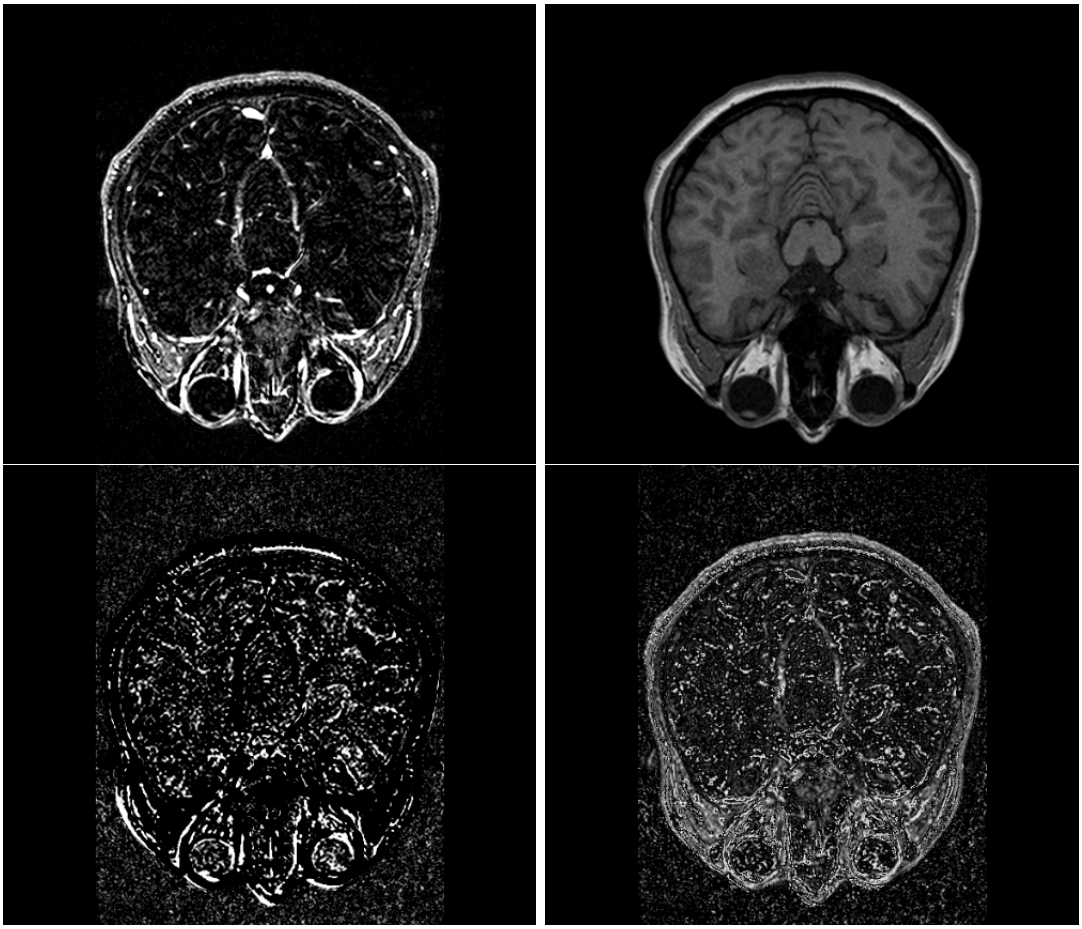


Figure 4.15: Sobel Operator Fusion

4.15 Standard Deviation fusion

This method

- uses grayscale values,
- visualizes only a summary of the datasets at each pixel,
- selects values from the datasets for visualization,
- shows one summary at a time,
- does not take into consideration the gradients between images,
- does not have any preference about high and low values,
- a region of the source pixels is taken into consideration when calculating the output value.

This type of fusion calculates the standard deviation between each pixel and the pixels that surround it (boundary conditions are solved by selecting the first input value, without calculation), and the resulting pixel will be the input pixel that gives the highest value of standard deviation. Since the standard deviation means that there are great changes in values in that region, if the operator's outcome is high, there are probably details that should be shown.

The next figure shows an example of this fusion, the first 3 pictures are the input pictures the fourth one is the image that we got out of fusion.

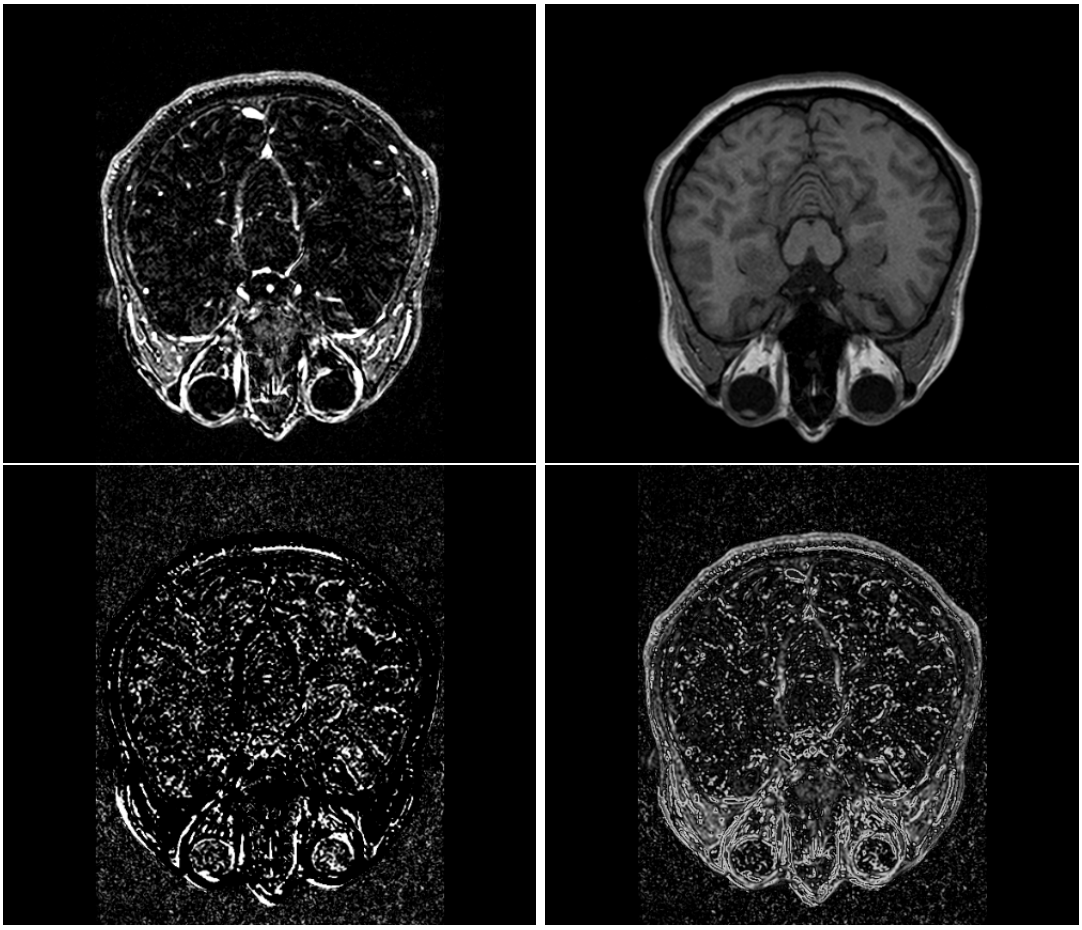


Figure 4.16: Standard Deviation Fusion

4.16 Discussion

There have been several fusion methods described in this chapter. This section will try to sum them up and discuss their properties, strengths and weaknesses.

Since the input medical images only consist of one intensity value, the use of RGB channels is a very good opportunity to visualize three datasets at once. This way we get all the information that is contained by each of the three datasets. While grayscale images let us show only one value at a position, which means we either select or summarize the values of the input pixels, thus has the effect of losing some information, the RGB channels let us show three values at a position. This way, we can easily see the differences between the details at each pixel, since the pixel will be close to gray if the values are similar, or close to red, green, blue, cyan, magenta or yellow if one (repectively two) of the RGB channel values is much higher than the rest of them.

This is what the RGB three channel fusion type is for. It is simple and very effective, it is used very often for example for PET-CT and PET-MRI fusions. While showing differences very well, this method's greatest weakness is that it can show only up to three channels while the differences between images is required most in case of time series where the number of

input images is a lot more than three, and we would like to see how the condition of a body part is changing in time. The multiple channel variant of RGB fusion tries to solve this problem. As described earlier, it is a generalized version of the common and simple RGB fusion. While the latter represents the input channel's values as values of red, green and blue, the former selects more colors from the spectrum to represent the input channels' values. This way the number of channels represented and visualized is only limited by the number of selected colors, so it can be used for time series with more than three inputs. However, the greatest disadvantage of this solution lies exactly in the principle of color selection: If we see a pixel that has a color close to green in case of a simple RGB fusion, we can be sure that the green channel input's intensity is much higher at that position than the other channels' input intensities. On the other hand, since the multiple color fusion method can use colors that are somewhere between the basic RGB colors, interferences can occur very easily. This means that if we see a pixel that has a color close to green, like in the above mentioned example, in case of a multiple colored fusion, it can mean that the green colored channel has a higher valued pixel at that position than the others, but it can also mean that the lime and turquoise colored channels have higher valued pixels there and the rest of the channels have lower. This means that special care has to be taken of how the colors are assigned to the input channels, so these kind of interferences would not cause completely misleading images.

Another approach to this problem is the multiple channel gradient fusion. While using the same principle of color selection like the multiple channel fusion, it only shows the differences between two neighboring channels instead of the values of the channels themselves. This helps visualize long time series, since these kind of inputs are usually very similar, there is only a limited number of changes between two neighboring inputs (like the difference between the size of a tumor at the moments of acquisition). By showing only these differences, which are most probably at different places - meaning that the interference will probably not be as significiant as in the case of multiple channel fusion, comparing the output of this fusion with the output of multiple channel fusion can help identify interferences in the latter and find out what the interfering colors were.

Although the use of colors is very good to see each value of each of the input channels, there are times when we don't need all the exact values, but only the information that they contain, and too many colored channels would only cause confusion. This is the case for example with images that show different objects with focusing the camera on them one-by-one (not very common in medical imaging because of the very nature of the acquisition methods) or with multi-modal images or images with different acquisition settings (these are, however, very often used in medical imaging).

If we consider the information contained by CT and MRI, we will see that CT shows solid tissues where the X-rays do not go through while MRI shows tissues with a lot of water in it. This means that solid tissues will not be shown in MRI (since they contain very little water) while water containing tissues will not be shown too well in CT images since these tissues let most of the X-rays through. However, it is very important to see how these tissues are located in the same coordinate space, so an effective fusion method that shows the information contained by these images is very important.

There are two types of methods in this thesis that deal with such a fusion while giving a grayscale value output: the output pixel's value is either a summary of the input pixels or the value of one of the input pixels itself (presumably the one that contains the most

information). There is only one summarizing method developed in this thesis, the averaging fusion method, and several input value selector methods because each of them consider a different measure of information of the pixels.

There is one major issue about these kind of fusion methods concerning medical images: since these methods of image acquisition focus on different parameters while acquiring pixel values, this can result in images that have brightnesses and contrasts that are very far from one another. This means that the histograms of the input images are not aligned, and whatever measure of information we select to be the criterion for output value, some of the input images might not be included just because they are brighter, darker, etc. than the others.

This problem is solved by adjusting the input images to have a predefined brightness and contrast, which alignes their histograms, therefore the pixel values of the images will be from similar intervals with the same overall average value, which is the consequence of the statistical interpretation of brightness and contrast as the image's pixel values' *expected value* and *standard deviation*. Figure 4.17 shows this method of histogram alignment.

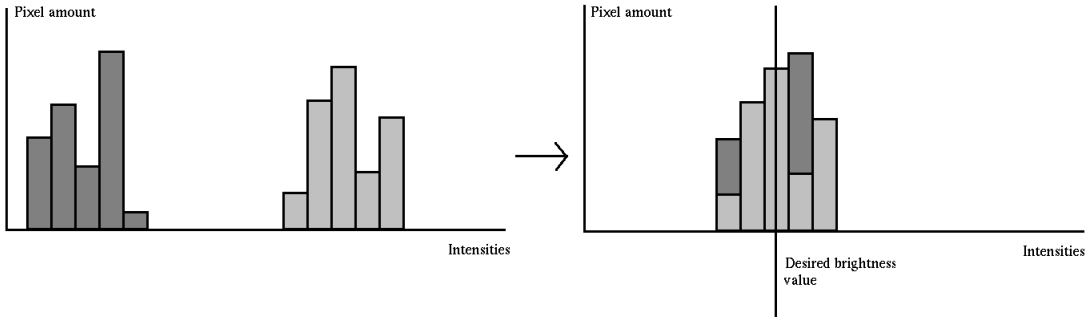


Figure 4.17: Histogram Alignment

It is very important to define how we measure information within an image since we would like these fusion methods to show the most information they can. This can vary from modularity to modularity, that is why several of these methods (respectively measurements) have been defined by the methods mentioned in the previous sections. They can be group into two categories:

1. Measures that take into consideration the values of only those pixels that lie at the given position of the input images:

   - maximum - the pixel with the highest value will be considered as the one with the most information for a given position. This is very useful for CT-MRI fusion where the tissues shown by each method are the ones with high values.
   - minimum - the pixel with the lowest value will be considered as the one with the most information for a given position. This is useful when PET or SPECT images are fused with a model that shows details as low intensity values since these methods have low value intensities where many positrons/photons come from (see figure 1.8).
   - median - the pixel that is the median among the pixel values will be considered as the one with the most information for a given position. This method does not

have specific target image types to use on, but it can produce good results when both high and low values contain information in an image (for example MR images with T1, T2 and T2* settings - see figure 1.6), and there are more than two of them.

2. Measures that consider also the pixels surrounding those particular ones, which lie at the requested position of the input images:

   - edge - the pixel's value that is the most probable to contain an edge will be selected to be the output value for the pixel at that particular position. This can be decided by applying an edge detector operator to the region around each pixel (like the Sobel operator), which means that convolution with an edge detector matrix is applied to each pixel and its surrounding region, and the pixel that comes out of this operator with the highest value will be the resulting output pixel. The operator can be selected according to needs, Sobel's operator has been tested for the purpose of this thesis.
   - standard deviation - the pixel's value is decided in the following way: for each of the pixels of the input channels, the standard deviation of the pixel values is calculated within a given range from the calculated position. The channel that comes out with the highest value for a certain position will provide the output value for that particular pixel. As it was mentioned before in this section, standard deviation can be interpreted as the contrast of an image or image region. This means that if the standard deviation for a particular region is higher for one input image than the others, than the contrast is much higher there, which means higher differences between pixel values. This implies the presence of details for that particular region of the image (as details are changes - and usually high value changes - in pixel values). Special care has to be taken of selecting the region's size because if it is two large, the outcome will consider far away pixels as well and could give high values to pixels that do not actually contain details but are located close to details.

A further feature that has been introduced in several of the fusion methods discribed in the previous sections, is showing more than one of the above mentioned summarizer/selector fusion method's output at once. This can be accomplished by using the RGB channels for visualization. However, this time they are not used for showing the different channels at once, but they are used to show different the results of different pixel summarizing fusion methods.

This is a very good way of seeing the differences between the information that is given by each of these selector methods. The most obvious (and probably the most useful) combination is the maximum and minimum selection because this way we can see the difference between the maximum and the minimum value distributions. Since RGB provides us the possibility of three input channels, it is possible to show another summary as the third input channel beside maximum and minimum selection.

Another interesting possibility is shown in several fusion methods that use gradient summarized images as the third channel. This means that the third input is not the summary of the input images but the summary of the differences betweeen neighboring images. This way the summary of time series can be summarized in a useful way by seeing the maximum values, the minimum values and the summary of the gradients of how the images changed.

The principles of gradient summarization methods are very similar to that of the simple

intensity summarization methods. Either all the gradient pixels are summarized (like averaging), or one of the input gradient values is selected (like maximum, minimum, median, etc.) to be the output. The measure of information of the selected pixel in the case of gradient selection methods can be considered the same as the measure in case of regular intensity selection, the measures will have the same advantages, disadvantages and uses as the ones discussed above.

It is also important to mention here, concerning the three channel summarized fusion types, that the possibility of changing the brightness and contrast of each of these summaries is very important because this way either one of the summaries can be emphisized or ignored according to the needs of the user.

This section tried to sum up and discuss the methods and results described and shown in the previous sections. Methods have been proposed for time series with little change and few channels, time series with little change and several channels, multimodal inputs that each contain different information that should be shown in the same picture and the possibility of showing the differences of up to three different summaries. None of the methods work best, they all have their purposes, and the only way we can get useful results out of them, if we consider which one to use for a certain kind of input image set.

Chapter 5

Conclusion

In this thesis, a variety of fusion visualization techniques have been designed and implemented. All of these methods are for slice viewing - a further improvement could be to design and implement fusion methods for 3D volume rendered images.

Some of the methods implemented are suitable for visualizing changes between several inputs, like in case of time series (for example the multi-channel and the multi-channel gradient fusion), some of the techniques are suitable to gather information from all the inputs, and visualize a summary of all the information, like in case of multimodal images (for example the intensity selector and intensity summarizer fusion types) or there are techniques that let us see the difference between several types of information (like maximum voxel value, minimum voxel value and average / median / gradient voxel value) by showing more than one summary at a time.

As discussed before, none of these methods is able to provide us all the information that the images contain, but according to the given purpose, the observing doctor should be able to find a method to at least partially fit his or her needs. Even if he or she does not, since the visualization part is modular, exchangeable and extreme care was taken to be reusable when developing further methods of fusion, it should be easy to implement new kinds of techniques according to the medical expert's needs.

Since most of the times it is a required preprocessing before fusion, registration of datasets have also been studied and implemented. Even though in the theoretical part of the thesis, a lot of possible methods and directions in this field have been mentioned, this was not the main goal of the thesis and only a simple technique has been selected to be implemented in the framework as part of this thesis.

The rigid body transform constraint is only suitable for few body parts (like the head), while other parts of the body (like thorax) makes a registration method that uses non-rigid body transform necessary. These type of methods do not have the above mentioned constraint, so they are much more effective, but much more complex as well, and implementing them would have been beyond the scopes of this thesis.

Another weakness of the implemented registration method is its speed. As the method tries to optimize the mutual information of the datasets, it has to perform the transformation of the image several times, which takes time.

Finally, there is also the possibility of local minimums in the registration criterion, so misregistrations can occur because of this reason as well.

There are possible further improvements concerning fusion as well. One of them would be to implement wavelet transform and inverse wavelet transform to detect the details in the

images. It is a widely used method for fusion, which is dealt with in many scientific articles and has several implementations, so it was not dealt with in this thesis. For a very good explanation of this method, see [2].

As more and more people are using, developing and adding components to the *MedV4D* project, hopefully some of these possible improvements will be implemented to improve the registration component of the framework, and possibly some further methods of fusion might be thought of and implemented also. This thesis gives a basic starting point to both of these fields of medical image processing within *MedV4D*, and hopefully will be of some use to medical experts as well as developers orienting in this direction.

Appendix A

CD contents

- `/Datasets` - data to test the application
- `/Documents`
  - `/MedV4D` - documentation of the MedV4D framework
  - `/Thesis` - LaTeX source files, generated pdf and image files
- `/Executables` - executable binaries of the application
- `/Sourcecodes` - source code of the MedV4D framework with the described components embedded and with the Volume Data Fusions application included in Applications

Appendix B

Implementation

The practical implementation of the ideas discussed in this theses are realized within the *MedV4D framework*. It is a set of components to help design and implement medical applications. A quick overview of the framework will follow now, for further information about this project, see [1].

The framework has three main components: Image filters, Graphical User Interface and DICOM backend. The image filters component uses only the *boost* [7] external library, otherwise it is written using only C++ standard libraries. It defines a datatype for images and also an effective pipeline for calculations. There are filter base classes that can be inherited from, so implementing filters is very easy. The registration and transformation filters benefit from this a lot.

The DICOM backend component is based on the *DCMTK* [8] toolkit, which implements the functions of the DICOM protocol.

The GUI environment, and especially the sliceviewer component is very important because the fusion itself is visualized by replacing a component in the sliceviewer. The GUI is programmed using the *Qt* [10] GUI framework, which besides it very easy and effective usage, makes it cross platform as well (Windows, Linux, FreeBSD, etc.). The sliceviewer itself is an *OpenGL* [9] component embedded into a Qt widget. This way, the sliceviewer itself can be realized using opengl functions, which is also cross-platform, and usually a key component of graphical oriented applications.

Now, the description of the two main components of the practical part of the thesis follows: the registration filter components and the fusion visualization sliceviewer components.

B.1 Transformation and registration

This part is realized as two filters, using the filter framework of *MedV4D*. First of all, the UML diagram of the design is shown to see the structure of this component (it does not show all the details and classes, only the ones that are new for these filters).

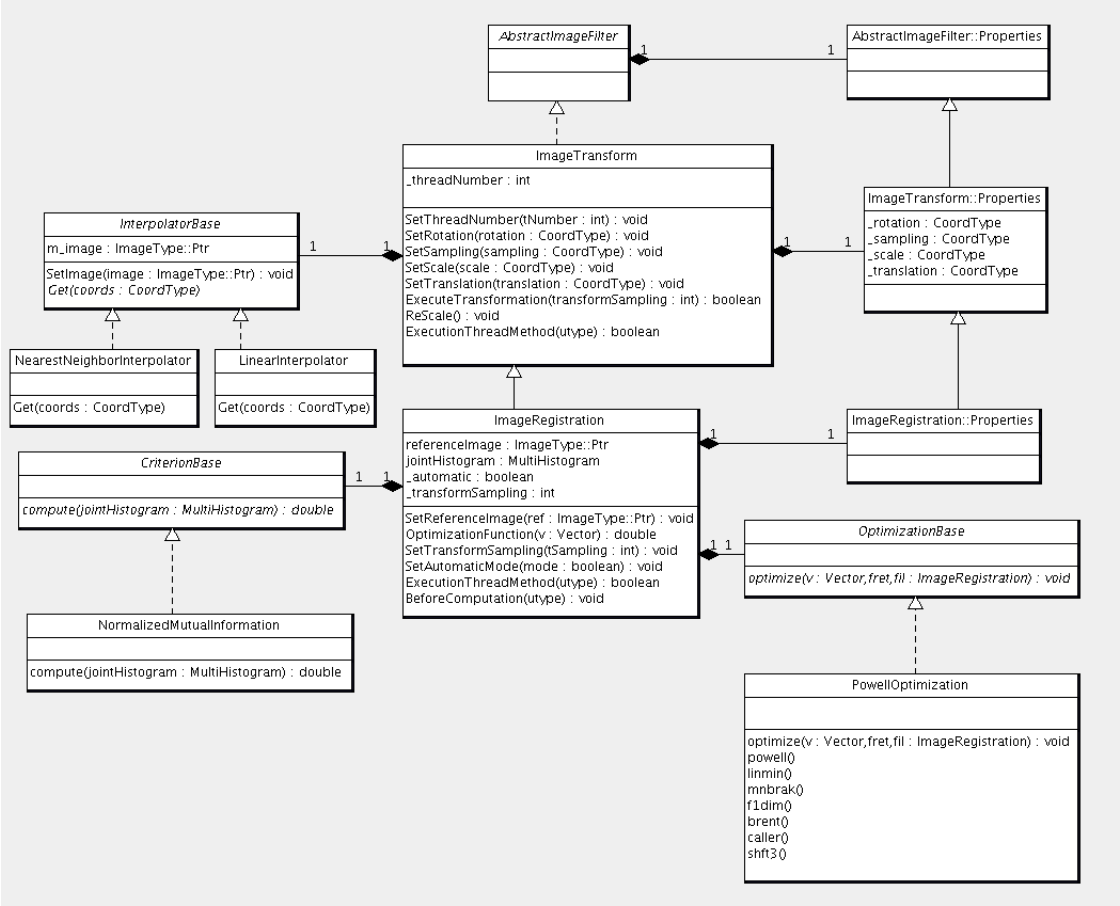


Figure B.1: Registration component UML diagram

B.1.1 ImageTransform::Properties

Just like all the filter classes in the *MedV4D* framework, the ImageTransform has a *Properties* structure that is derived from the *Properties* structure of its superclass. Most of the times, this structure is empty (except for the fields inherited) and is used by the pipeline mechanism, but in this case it holds some information as well. Its fields are the following:

- _rotation - stores the transformation's rotation in radians around each angle of the coordinate system.
- _sampling - stores the transformation's sampling, which means that the output image should be sampled in *_sampling* : 1 ratio to the input image.
- _scale - stores the transformation's scale, which means that the voxel distances of the input image will be multiplied by this value to get the output image's voxel distances for each dimension.
- _translation - stores the transformation's translation in each direction of the coordinate system.

This structure is useful for handing over to partially specialized functions that calculate the transformation and are not part of the ImageTransform class.

B.1.2 InterpolatorBase

Since interpolation is essential for effective image transformation, the ImageTransform filter, too, needs a component that realizes this function. This class is an abstract base for these interpolators. It needs an image to do the interpolation on, which can be set using the *SetImage* method.

The only pure virtual method that makes this class an abstract, is the *Get* method. This is the method that would calculate the interpolated value of the transformed voxel according to the input coordinates, and which is needed to be implemented in subclasses, according to the method that the subclass wishes to realize.

B.1.3 NearestNeighborInterpolator

As a subclass of InterpolatorBase, this class realizes the nearest neighbor interpolator method. It does this by implementing the base class' *Get* method. The whole implementation is very simple, just like the algorithm itself - the coordinates are rounded to the nearest integer value and the value of the voxel at the resulting position is returned.

Because of its simplicity, it is a fast method, but the borders between transformed voxels are visible in the resulting image, which disturbs the smoothness of the picture.

B.1.4 LinearInterpolator

Another subclass of InterpolatorBase that realizes the trilinear interpolator (and additionally, using partial specialization, can easily be modified to realize bilinear and simple linear transformation as well) method. Yet again, the *Get* method is implemented. Each of the 8 values around the requested point are stored, and the result is calculated by weighting them according to the point's relative position to them, and the results are summed up and returned.

Since this method requires a lot more calculation than the previous one, this class works slower than the above mentioned one, however, it gives a much smoother result.

B.1.5 ImageTransform

The ImageTransform class is one of the two central elements of the registration component. It is a *MedV4D* filter that gets one input image, and creates a transformed output image, according to the parameters held in Properties and the type of interpolation used. It has only one significant attribute, namely *_threadNumber*, which defines the number of threads that are executed simultaneously (each transforming one slice of a 3D image) during transformation. This makes the parallelization of the whole process not only possible, but also scalable by giving the opportunity of setting this number according to CPU and/or CPU-core count.

The ImageTransform contains the following methods:

- SetThreadNumber - it sets the _threadNumber attribute.
- SetRotation - it sets the _rotation attribute of Properties, thus the rotation of the transformation.

- SetSampling - it sets the _sampling attribute of Properties, thus the sampling of the transformation.
- SetScale - it sets the _scale attribute of Properties, thus the scale of the transformation.
- SetTranslation - it sets the _translation attribute of Properties, thus the translation of the transformation.
- ExecuteTransformation - this method sets the input of the interpolator (or throw an exception if there is no interpolator component attached), and transforms the image. Its parameter is the sampling rate, which means that how many voxels are to be left out of transformation (so left blank) after one transformed voxel in a row or a column. If this is 0, a complete transformation is to take place. This is useful when the registration is processed, which needs a lot of transformations to calculate, which can be speeded up by not transforming all the voxels.
  The image transformation starts a new thread for each slice and using the transformation matrix, which is calculated according to the Properties structure's attributes, transformations for position (0, 0, sliceNumber), (1, 0, sliceNumber) and (0, 1, sliceNumber) are calculated. The difference between them is used to get the rest of the positions without all the multiplications needed by the transformation itself by starting from the origo point and adding to it the multiplication of the differences and the coordinates. Finally the interpolation of the voxel value occurs.
- ReScale - this method only sets the voxel extents of the output image if needed.
- ExecutionThreadMethod - it does some checks at the beginning and then simply calls ReScale() and ExecuteTransformation(). This function is called by the pipeline, so it needs to be implemented.

B.1.6 ImageRegistration::Properties

As it has already been mentioned before, most of the filters' Properties structure is empty and only inherits the content of the baseclass' Properties structure. It is the same way with ImageRegistration filter's Properties, which only inherits the content of ImageTransform's Properties.

B.1.7 CriterionBase

The criterion is an element of the ImageRegistration filter. This contains is the function (as a method) that has to be optimized to get the images registered. This class is only an abstract class with nothing more than one pure virtual method (*compute*) that computes the criterion and returns a double value. The parameter of this method is the joint histogram of the two images that are to be registered.

B.1.8 NormalizedMutualInformation

It is an implementation of abstract class CriterionBase. This class implements the *compute* method to calculate the normalized mutual information value of the two images according

to the parameter joint histogram. There is nothing special in its implementation - it only calculates the NMI value according to the equation mentioned in section 3.3.3.

This component works fairly fast provided that the input joint histogram has a resolution of hundreds by hundreds.

B.1.9 OptimizationBase

This abstract class gives an interface to implement for optimization of the criterion function of the registration. Its only method is a pure virtual one, called *optimize*, which has parameters the vector of input values, reference to the return value and a pointer to the ImageRegistration filter it belongs to. This last one is needed because the function that we want to optimize is a method of the ImageRegistration filter, so the easiest way is to pass a pointer to the whole structure, and call its method when needed.

B.1.10 PowellOptimization

This class is a wrapper for Powell's algorithm implemented in (and taken from, as an external module) [5]. It implements the *optimize* method of the OptimizationBase class only to prepare the output parameters for the powell method module, and start the optimization.

The *caller* method only converts the vector of input parameters of the criterion function to a vector type used by the powell module, which is an own vector implementation.

All the rest of the methods of this class (powell(), linmin(), mnbrak(), f1dim(), brent() and shft3()) are part of the external powell implementation module and are discussed in detail in [5].

B.1.11 ImageRegistration

This filter is the one that does the registration itself. It is derived from the ImageTransformation class, and has several important attributes and methods.

The attributes are the following:

- referenceImage - this is a smart pointer to the reference image according to which the registration is made to the input image. It is very important that this pointer is set before the execution of the filter because without the reference image, there can be no registration done.

- jointHistogram - this is a MultiHistogram, which contains the joint histogram of the reference image and the input image. It is calculated iteratively after each step of transformation calculation.

- _automatic - this attribute is a boolean only saying whether an automatic or a manual registration is desired. If an automatic one is needed, the algorithm prepares the parameters and executes the powell optimization module to find a minimum of the normalized mutual information function. On the other hand, if a manual registration is selected, the image is simply transformed according to the parameters that are given to the filter. The automatic transformation always starts with the parameters calculated during the previous registration for a particular image regardless of whether it was a

manual or automatic registration. This means that if we first transform the image manually and then execute the automatic registration, we might get different result than executing the automatic registration immediately since from different starting points, the registration can get to different local minimums.

- _transformSampling - this attribute defines what kind of sampling is to be used for calculating during the iterations of registration - meaning that how many voxels in a row and a column are to be left out after calculating one during transformation and during histogram calculation. The lower this value is, the more accurate the algorithm will work. However, the higher this value is, the faster the algorithm will finish. This means that this parameter is kind of a measure of trade-off between quality and speed of registration. The algorithm is a multi-resolution one, which means that it starts from registering a lower resolution dataset, and then iteratively makes the result more and more accurate by increasing resolution to the threshold value given by this parameter.

There are also several methods that are contained by this filter:

- SetReferenceImage - this method simply sets the smart pointer of the reference image.
- OptimizationFunction - this is the method that is given to the optimization component as a function to optimize. It sets the parameters of transformation according to the parameters, transforms the image and calculates the joint histogram of the input image and the reference image. The criterion function is called at last, and the inverse of the result is returned (because the optimization component minimizes, and we need the maximum of the normalized mutual information).
- SetTransformationSampling - it simply sets the _transformSampling attribute.
- SetAutomaticMode - sets the _automatic attribute.
- ExecutionThreadMethod - this method is called by the pipeline, and it has to be implemented. Here it begins with rescaling the image to have the correct size compared to the reference image. Then, it checks whether automatic registration is required. If it is, the program sets the input parameters and executes the powell optimization. Otherwise, or when the optimization is finished, the method goes on to execute the final transformation according to the registered or the prior given parameters.
- BeforeComputation - this function calculates the scale and sampling of the output image according to the input and the reference images, and stores the value for the transformation part of the execution.

B.2 Fusion

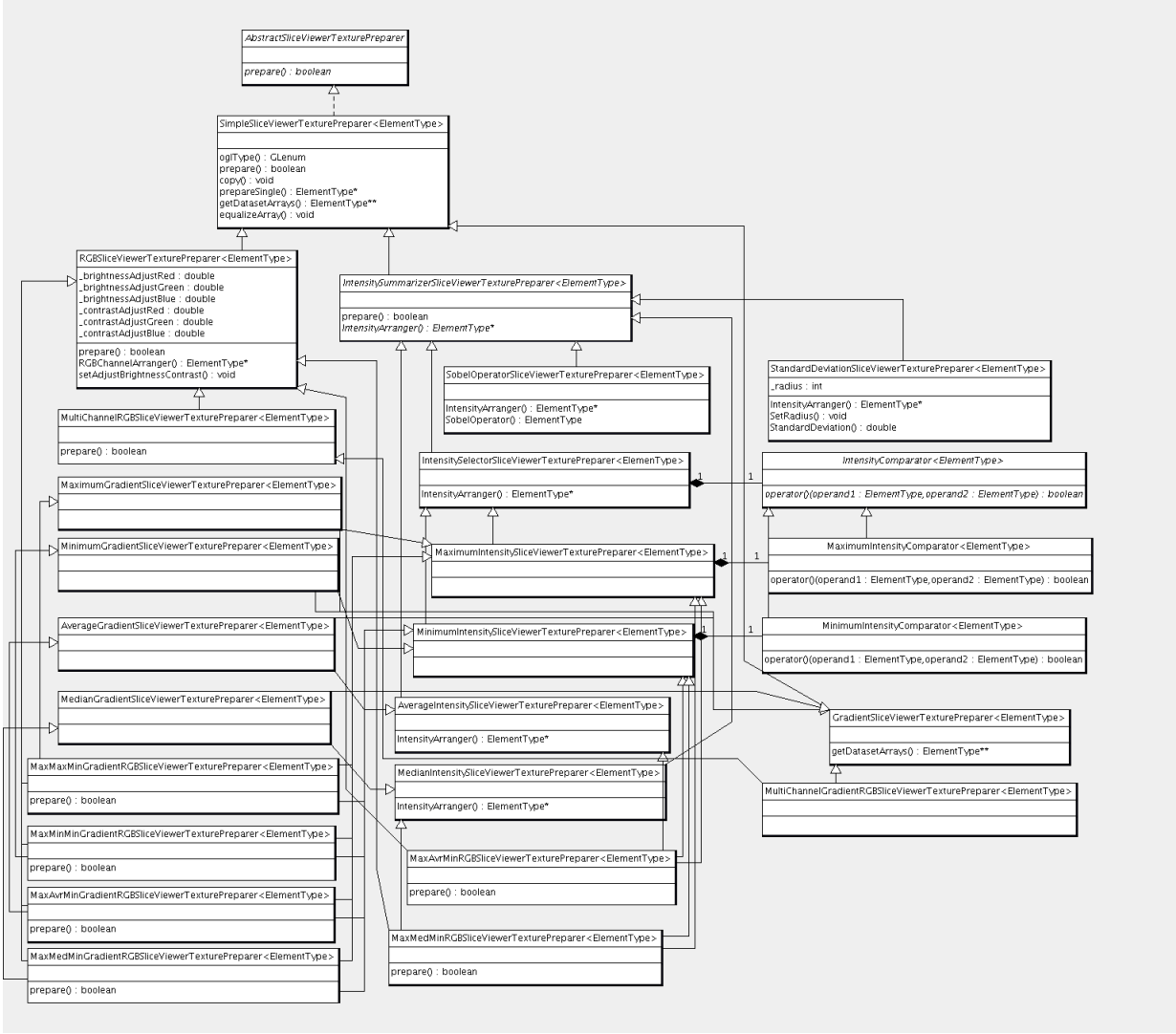


Figure B.2: Fusion component UML diagram

This component is responsible for visualizing the slices of the volumetric image. It needs to be connected to a SliceViewerWidget of the development framework, and the slices (or fusion of the slices) will be visualized according to which of the following elements is connected to the sliceviewer. There are numerous elements here, each of which has a common base class, AbstractSliceViewerTexturePreparer, which declares which method is needed by the sliceviewer and therefore needs to be implemented. The classes use advanced object-oriented programming techniques, such as multiple inheritance, code reusing and templates. The classes themselves are not too complicated, the functionality of this component is based on the class hierarchy, which is, however (as it can be seen from the UML diagram above - the parameters of methods have been left out to make it less difficult to understand), fairly complex. A description of each of the subcomponents of this component will follow.

B.2.1 AbstractSliceViewerTexturePreparer

This class is an abstract base of all the texture preparer classes. It declares what a texture preparer has to implement so that it could be connected to a sliceviewer widget. It has only a single method, called *prepare*, which ensures the preparation of the texture. This method has several parameters:

- inputPorts - it is the list of input ports to get the input datasets from.
- width - reference to set the width of the picture.
- height - reference to set the height of the picture.
- brightnessRate - the brightness value of the picture to set to.
- contrastRate - the contrast value of the picture to set to.
- so - the orientation of the slice in the coordinate system (xy, yz, zx).
- slice - the slice number to be shown.
- dimension - the dimension of the dataset.

The method should return a bool indicating whether the texture preparation has been successful or not. Usually this method passes some or all of these parameters to other methods in subclasses, so parameters with the same name mean these parameters and will not be explained again.

B.2.2 SimpleSliceViewerTexturePreparer

This is the basic implementation of the texture preparer. It is used for simple slice visualization, which means that the slice from the first input port is dragged, and its pixels are arranged in order to set them as OpenGL texture. To achieve this, several methods are defined:

- oglType - simply returns the OpenGL enumeration number of a given type according to the template parameter. It is important so that the texture binding function could use it without having to partially specialize the whole *prepare* function.
- pepare - the implementation of the method from base class. It simply gets the dataset array, modifies the brightness and contrast of the first element, and calls the OpenGL texture preparer function for that element.
- copy - copies the pixels of the dataset into a new array while reordering them to be correctly set up to visualize (parameters are the source pointer, destination pointer, the old and new heights and widths, the slice number and the strides) and fills its border up with black pixels if the slice does not have a size of $2^N * 2^N$ - this is needed for compatibility with older graphics cards.
- prepareSingle - drags the dataset out of the input port given as a parameter (the rest of the parameters are the same as the ones of *prepare* in section B.2.1) and reorders the pixels, using *copy*.

- getDatasetArrays - simply calls *prepareSingle* for all the input ports in the port list in the parameter (again, parameters are the same as the ones of *prepare* in section B.2.1), and sets one of an array of pointers at each one of them. This array is returned as a result.
- adjustArrayContrastBrightness - it sets the picture's pixel value, which is passed as a pointer to an array in the first parameter, to the given brightness and contrast rate (the rest of the parameters are like the ones in *prepare*), so that the picture would have the desired brightness and contrast.

B.2.3 RGBSliceViewerTexturePreparer

This class is derived from SimpleSliceViewerTexturePreparer and implements the *prepare* method in a slightly different way: it receives up to three arrays of datasets from *getDatasetArrays*, modifies all of them according to brightness and contrast settings, and finally calls its own *RGBChannelArranger* method, which organizes each one of them into one array of RGB channel values. Finally, the texture is prepared, but with an RGB setting in the parameters.

B.2.4 MultiChannelRGBSliceViewerTexturePreparer

It inherits from RGBSliceViewerTexturePreparer and reimplements only the *prepare* method. This class drags the array of all the input datasets from all the input ports, and gives each of the channels a different color from the spectrum circle, shown in figure 4.2. The values are summed up after weighted by the values of the pixels of each dataset and divided by the number of input datasets. The result is arranged using the superclass' *RGBChannelArranger* method.

B.2.5 IntensitySummarizerSliceViewerTexturePreparer

This is an abstract class, derived from SimpleSliceViewerTexturePreparer. It reimplements the *prepare* method, by dragging all the input datasets, and summarizing them by calling the *IntensityArranger* pure virtual method, which needs to be implemented in derived classes. The resulting array of grayscale pixels are then set as the texture for the given slice.

B.2.6 IntensitySelectorSliceViewerTexturePreparer

This class is derived from IntensitySummarizerSliceViewerTexturePreparer. It implements its pure virtual method, *IntensityArranger*. For each pixel, a special **IntensityComparator** class is used, which is an abstract base needed to be implemented. Using this class' () operator, the texture preparer goes through all of the datasets for each pixel, and sets the resulting pixel to the grayscale value to the value that is the "greatest" (meaning the last in the order) according to the comparator.

B.2.7 MaximumIntensitySliceViewerTexturePreparer and MinimumIntensitySliceViewerTexturePreparer

These two classes are derived from IntensitySelectorSliceViewerTexturePreparer, and reimplement none of the methods of their base, only implement the comparator interface to select

the maximum respectively the minimum of the datasets' values for each pixel.

B.2.8 AverageIntensitySliceViewerTexturePreparer

It is derived from IntensitySummarizerSliceViewerTexturePreparer, and it implements the base class' pure virtual method *IntensityArranger*. All it does is summarize the datasets' pixel values for each position, and divides them by the number of the datasets, thus calculating the arithmetic mean of datasets for each pixel.

B.2.9 MedianIntensitySliceViewerTexturePreparer

As yet another derived class of IntensitySummarizerSliceViewerTexturePreparer, this class implements the *IntensityArranger* method too. Unlike the average calculator texture preparer, this class sorts the values of the datasets for each pixel and selects the median of them as the output pixel.

B.2.10 SobelOperatorSliceViewerTexturePreparer

This is a derived class of IntensitySummarizerSliceViewerTexturePreparer that uses the Sobel Operator to detect edges. Edges are probably details, so if the Sobel Operator gives a high value for a pixel, there probably are details. This is why there is a method, called *SobelOperator*, which calculates this value for a given pixel, and the *IntensityArranger* method will select the pixel that gives the highest value after applying this operator.

B.2.11 StandardDeviationSliceViewerTexturePreparer

As another derived class of IntensitySummarizerSliceViewerTexturePreparer, which takes into consideration the input pixels' neighbors, this class also implements *IntensityArranger* and selects the value of the output pixel according to the standard deviation of the pixels that surround the input pixel at the given position. The *_radius* attribute (which can be set by calling *SetRadius*) defines the size of the radius of the given position, within which the pixels' values will be taken into consideration for the calculation of the standard deviation value (this calculation is implemented in the *StandardDeviation* method). The intensity of the pixel with the highest result of standard deviation value will be copied to be the output pixel intensity for that particular position.

B.2.12 MaxAvrMinRGBSliceViewerTexturePreparer and MaxMedMinRGBSliceViewerTexturePreparer

These classes use the well-known object-oriented programming method of multiple inheritance, inheriting from classes MaximumIntensitySliceViewerTexturePreparer, AverageIntensitySliceViewerTexturePreparer/MedianIntensitySliceViewerTexturePreparer respectively, MinimumIntensitySliceViewerTexturePreparer and RGBSliceViewerTexturePreparer. They reimplement the *prepare* method in the following way: after getting the array of dataset pixels it applies MaximumIntensitySliceViewerTexturePreparer's, AverageIntensitySliceViewerTexturePreparer's/MedianIntensitySliceViewerTexturePreparer's and MinimumIntensitySliceViewerTexturePreparer's *IntensityArranger* methods, receiving three different array of

pixels. Finally, RGBSliceViewerTexturePreparer's *RGBChannelArranger* is used to arrange these pixel arrays into one single RGB pixel array and set that array as the texture to be rendered.

B.2.13 GradientSliceViewerTexturePreparer

This class inherits from SimpleSliceViewerTexturePreparer. It uses all of the methods implemented by the base class, the only reimplementation it makes, is the *getDatasetArrays* method. This method, after calling the same named method of the base class, it calculates the differences (this way approximating the gradients) between each pairs of dataset pixel arrays that are next to each other. Finally, the array of gradient pixel arrays (which is, of course, means the reduction of the number of the arrays by one) is returned instead of the datasets' original pixel arrays.

B.2.14 MultiChannelGradientRGBSliceViewerTexturePreparer

This class makes use of multiple inheritance. It inherits from MultiChannelRGBSliceViewerTexturePreparer and from GradientSliceViewerTexturePreparer, therefore instead of giving colors to and summarizing of the original dataset pixel values, this class visualizes the gradients of the dataset pixel arrays because the *prepare* method inherited from MultiChannelRGBSliceViewerTexturePreparer will call the *getDatasetArrays* method inherited from GradientSliceViewerTexturePreparer.

B.2.15 MaximumGradientSliceViewerTexturePreparer, MinimumGradientSliceViewerTexturePreparer, AverageGradientSliceViewerTexturePreparer, MedianGradientSliceViewerTexturePreparer

These classes, again, do not implement anything, only make use of inheritance. They inherit from classes GradientSliceViewerTexturePreparer and MaximumIntensitySliceViewerTexturePreparer / MinimumIntensitySliceViewerTexturePreparer / AverageIntensitySliceViewerTexturePreparer / MedianIntensitySliceViewerTexturePreparer respectively, so the *prepare* method inherited from the latter base classes will use the *getDatasetArrays* method of the former meaning that the summarizer function will summarize the maximum / minimum / average / median gradient respectively instead of summarizing the actual dataset pixel values.

B.2.16 MaxMaxMinGradientRGBSliceViewerTexturePreparer, MaxMinMinGradientRGBSliceViewerTexturePreparer, MaxAvrMinGradientRGBSliceViewerTexturePreparer, MaxMedMinGradientRGBSliceViewerTexturePreparer

These classes are very similar to MaxAvrMinRGBSliceViewerTexturePreparer and MaxMedMinRGBSliceViewerTexturePreparer. The only difference is, that for the green channel instead of using the average / median texture preparers, these texture preparers use the

classes MaximumGradientSliceViewerTexturePreparer / MinimumGradientSliceViewerTexturePreparer / AverageGradientSliceViewerTexturePreparer / MedianGradientSliceViewerTexturePreparer respectively. This means that the green channel of the prepared slice texture will show some kind of summary of the gradients of dataset pixel values instead of the summary of the pixel values themselves. The methodology is the same as before: multiple inheritance and using the *IntensityArranger* method of the three derived classes inheriting directly or indirectly from IntensitySummarizerSliceViewerTexturePreparer.

Appendix C

User Manual of the demonstrating program

Since this program is written using the *MedV4D* framework, many of the GUI features described here are common for this framework and are explained in more detail in [1].

C.1 Installation and Execution

The execution of the program is very simple: the executable binary (according to operating system) needs to be copied somewhere on the hard disk, and a configuration file, called *config.cfg*, needs to be created (or the default copied) to the same folder where the binary is. The configuration file sets several things including the location of the DICOM server and the client ID, so that the server would allow connections from our client. Now, the executable binary can be executed: *VolumeDataFusions*.

C.2 Main Window

When the program is executed, the main window is visible, which is shown in figure C.1. The toolbar buttons (open, load, save, screen layout, contrast/brightness, pan, zoom, stack, info overlay, probe tool, new point, new shape, clear point, clear shape, clear all, flip horizontal, flip vertical, slice orientation, rotate volume, replace viewer) and menu items are explained in [1] as previously mentioned, and they also have informative text visible in the status bar when the mouse cursor hovers over them.

The main window has a combo box that allows us to set which connection of the pipeline should be visible by the selected viewer. The program can process, register and show fusion of 12 image datasets at a time (this number can be set at compile time). Each of the inputs are registered and/or transformed before fusion, so altogether there are 24 items in the combo box: one for each input dataset and one for each registered/transformed dataset to select which one of them needs to be shown.

It is very important that in order for registration and fusion to work, the *Input #1* has to be set. It is always the reference image, according to which the other datasets are sampled and scaled, and - in case of automatic registration - matching is done. Without setting this

input, no fusion will be shown, and instead of registration, the input image will be simply copied to be the output as well.

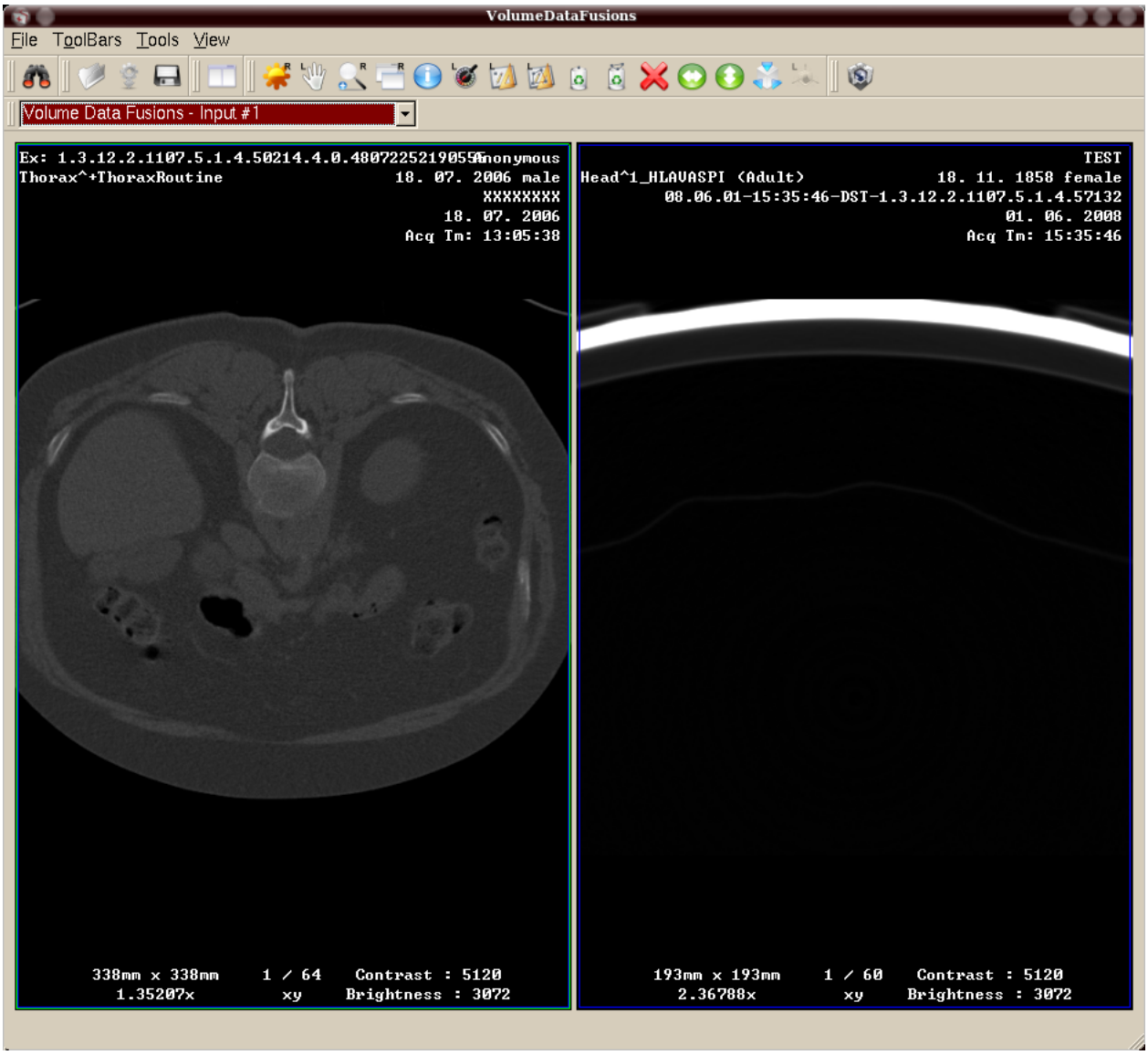


Figure C.1: Application Main Window

C.3 Data loading

Data loading is very similar to the default loading method of MedV4D, but since we have to know which out of the 12 input datasets we are about to load, we have to select the given number in the source combo box. When we want to load data, it does not matter whether we select the combo box item with input or registered label, the important is to select one with the number of the input that we wish to load.

After selecting the input number, we can proceed to the *Search button* either in the toolbar or in the *File* menu. In the upcoming study manager window C.2, we select whether we want one of the recently viewed images, a remote image from the DICOM server or an image

from our hard disk. After selecting the location of the dataset, we give search to get the study list. From the list we select the study we want to examine, and give view. The image is shown after loaded, or another window pops up (C.3) if there are multiple images inside one study to select the one we want to see.

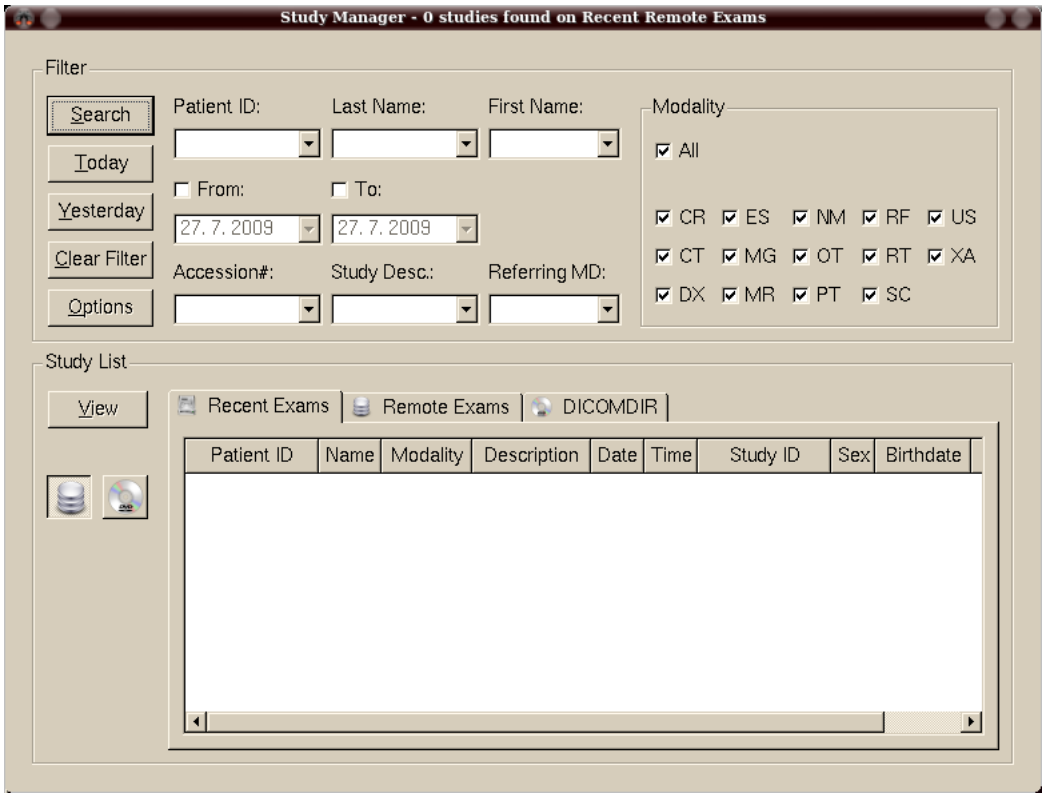


Figure C.2: Study Manager to select and filter studies

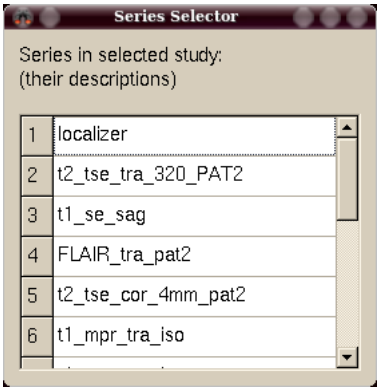


Figure C.3: Image selector in case of multiple images in a study

C.4 Settings window

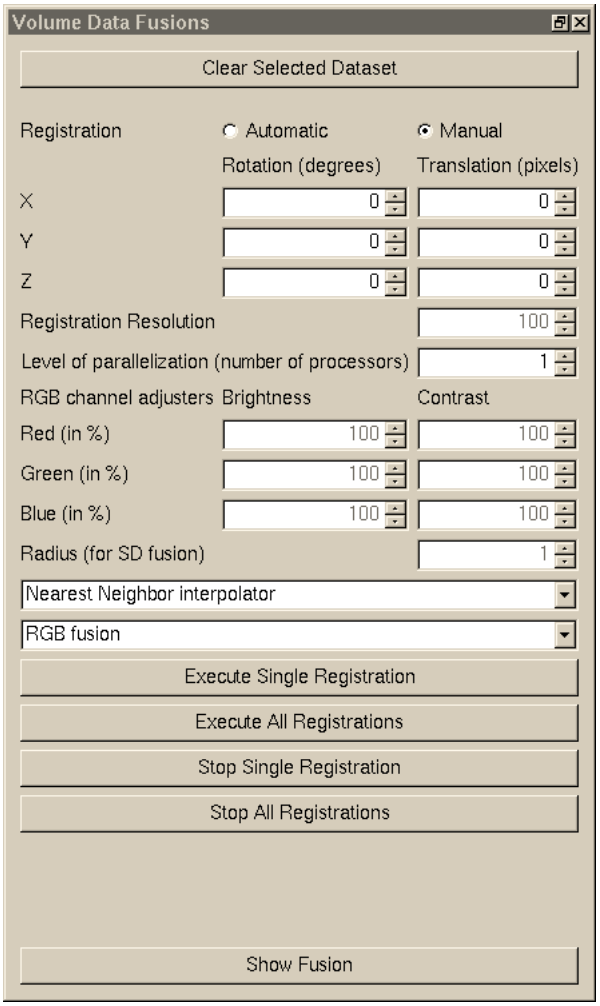


Figure C.4: Settings window

The Settings window controls the application itself. There are numerous elements in this window, each of which is explained in the following.

Clear Selected Dataset

This button clears the input dataset that is selected by the given viewer.

Registration

It is a radio button that selects whether the transformation of the image should be given manually, or it should be registered automatically by the software. The automatic registration uses the parameters of the previous registration of a particular input, which means that if we execute automatic registration in the beginning, the registration might find other lo-

cal minimum of the criterion function than if we provide a different starting point by first transforming the image manually. This way semi-automatic registration is possible beside automatic and manual ones.

Translation

The three spinboxes under this caption allow the user to set the translation of the manual transformation of the input image in each direction of the 3D space coordinate system. It is disabled when an automatic registration is requested.

Rotation

The three spinboxes under this caption allow the user to set the rotation of the manual transformation of the input image around each axis of the 3D space coordinate system. It is disabled when an automatic registration is requested.

Registration Resolution

In this spinbox the user can select the maximum number of samples per row, column and depth to be considered in the final iteration of calculation during the multiresolution registration process. This way, registration of a lower resolution image may occur, which results in a faster, but less accurate calculation. This field is disabled when a manual registration is requested.

Level of parallelization

Here, the parameter for parallel processing can be set. This is used during the transformation of the image dataset. It sets the number of slices processed at once. It is a good idea to set this parameter a bit higher than the number of processor(core)s present, but not too high since it would slow the calculation rather than speed it up.

RGB channel adjusters - Brightness

These parameters adjust the global brightness rate of the image for each of the RGB channels. These values are given in percentage and are absolute values, so 100

RGB channel adjusters - Contrast

These parameters adjust the global contrast rate of the image for each of the RGB channels. These values are given in percentage and are absolute values, so 100

Radius (for SD fusion)

This paramter is only available in case of a Standard Deviation fusion. It defines the calculated pixel's radius, within which the pixel values will be taken into consideration for standard deviation calculation.

Interpolator type

This field sets the interpolator type used for transformed voxels. There are two types of interpolators that can be set here: nearest neighbor or trilinear. The former is faster, but less smooth, the latter gives a smoother result, but is calculated slower.

Fusion type

Here the type of fusion can be set, which are discussed in chapter 4. Their functions and results are discussed in detail there, here only a list of available fusion types follows:

- RGB fusion
- Multiple colored fusion
- Maximum Intensity fusion
- Minimum Intensity fusion
- Average Intensity fusion
- Median Intensity fusion
- Maximum - Average - Minimum summarized RGB fusion
- Maximum - Median - Minimum summarized RGB fusion
- Multiple colored gradient fusion
- Maximum - Maximum gradient - Minimum summarized RGB fusion
- Maximum - Minimum gradient - Minimum summarized RGB fusion
- Maximum - Average gradient - Minimum summarized RGB fusion
- Maximum - Median gradient - Minimum summarized RGB fusion

Execute Single Registration

Execute the registration of input that is selected in the given viewer.

Execute All Registrations

Execute registration for all the input datasets.

Stop Single Registration

Stop the execution of registration for the selected input.

Stop All Registrations

Stop the execution of registration for all of the inputs.

Show fusion

Take the output of the registrations of the input datasets, and visualize them in the selected viewer, using the selected type of fusion.

Bibliography

[1] Gróf Sz., Klecanda V., Kolomazník J., Ulman A.: *MedV4D project documentation*, Software project, Charles University in Prague, Faculty of Mathematics and Physics, 2008.

[2] Hill P., Canagarajah N., Bull D.: *Image Fusion using Complex Wavelets*, The 13th British Machine Vision Conference, 2002, page 487-496.

[3] Maintz J. B. Antoine, Viergever Max A.: *A Survey of Medical Image Registration*, Medical Image Analysis, 1998, Volume 2, Number 1, page 1-36.

[4] Pluim Josien P. W., Maintz J. B. Antoine, Viergever Max A.: *Mutual information based registration of medical images: a survey*, IEEE Transactions on Medical Imaging. 2003, Volume 22, Issue 8, 986-1004.

[5] Press William H., Teukolsky Saul A., Vetterling William T., Flannery Brian P.: *Numberical Recipes in C: The Art of Scientific Computing*, 2. edition, Cambridge University Press, 1992.

[6] Tůma, T.: *Segmentace cév mozku v CT angiografii*, Bachelor thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2006.

[7] *Boost C++ libraries documentation*, http://www.boost.org/doc

[8] *DCMTK documentation*, http://dicom.offis.de/dcmtk

[9] *OpenGL documentation*, http://www.opengl.org

[10] *Qt documentation*, http://www.qtsoftware.com