Master Thesis Computer Sience

3D Blood-Vessel Analysis and Visualization

Authors: Tsjipke Wijbenga
Gijs de Vries

Supervisors: Dr. M. H. F. Wilkinson
Dr. M. A. Westenberg

RuG

Contents

Chapter 1

Introduction

Innovative scanning technologies such as computed X-ray tomography (CT), magnetic resonance imaging (MRI) or positron emission tomography (PET) empower radiologists to obtain 3D information of the inner human. This information is represented by a set of 2D gray scale images stacked upon each other. Analyzing those image sequences for diagnosis and therapy purposes using 2D image processing systems is a hard and time-consuming task, especially in the case of bloodvessels or other filamentous structures, which are the subject of this report. Not only structures which are not interesting for the observer are present in the image; thin elongated structures (like bloodvessels) which are perpendicular to the viewing direction of those 2D "slices" are hardly visible. This indicates the need for alternative ways of visualizing data with such structures. An example which shows the advantage of 3D imaging (a volume) over 2D imaging (a slice) is given in figure 1.1.

Virtual reality applications offer the possibility to visualize the data in a more intuitive way. By looking at a 3D image, physicians can recognize topological coherency in a much faster and more natural way. The benefits of virtual reality has been shown in many applications ranging from architectural design to flight simulations, but there is a difference between these and medical applications. Virtual reality mostly uses (textured) polygons to visualize a virtual environment. Data from CAD systems, like in architectural design, already consists of polygons. Applications that use natural data that is converted to polygon data depend heavily on simplification algorithms to keep polygon count within a reasonable range. By simplifying data, information can be distorted or lost for the sake of reasonable frame rates. This must not be done in medical applications because a diagnosis can depend on these small details. Current advances in constructing high performance computers and implementing 3D algorithms in hardware will overcome these limitations, but even constructing a inefficient polygon set from medical data has some difficulties. Data obtained from MRI, CT or PET consists, as earlier noted, of a cube of scalar values. Only a 'brightness' is known. If one wants to create a triangle mesh (or other graphical primitives) out of it, more information is needed.

Enhancement of curvi-linear, dendritic or other filamentous details has many applications in medical image analysis. In this thesis, we study one of those medical applications: the extraction and visualization of blood vessels in 3D angiography datasets. We explain in chapter 2 the concept of filtering, and show that blood vessels can be extracted more easily using filtering. Also a data structure (*Max-tree*) is described, which is excellently suitable for fast filtering. Chapter 3 is about segmentation of the vessels (dividing the voxels in two classes, vessel voxels and non-vessel voxels). This can be useful for quantitative analysis. Several segmentation methods are described, and a few are implemented. Also, new variants of exsisting strategies are implemented, and the results are discussed. We extended an existing segmentation method which has not been used before in this context (Rapid Automatic Threshold Selection), and developed it further. Those modi-

fications and extensions gave results which were promising enough to lead to a submitted publication for ICIP 2003 (International Conference on Image Processing). In chapter 4 we will give a quick survey on different visualisation techniques and combine direct volume rendering techniques with the Maxtree. We show that with the Maxtree, if combined with the right visualization method, filtering and visualization is possible in a speed at which interactively working is possible. A discussion of the results and conclusions are given in chapter 5.

We would like to thank everybody who helped us and inspired us during the research, especially our supervisors Michael Wilkinson and Michel Westenberg.

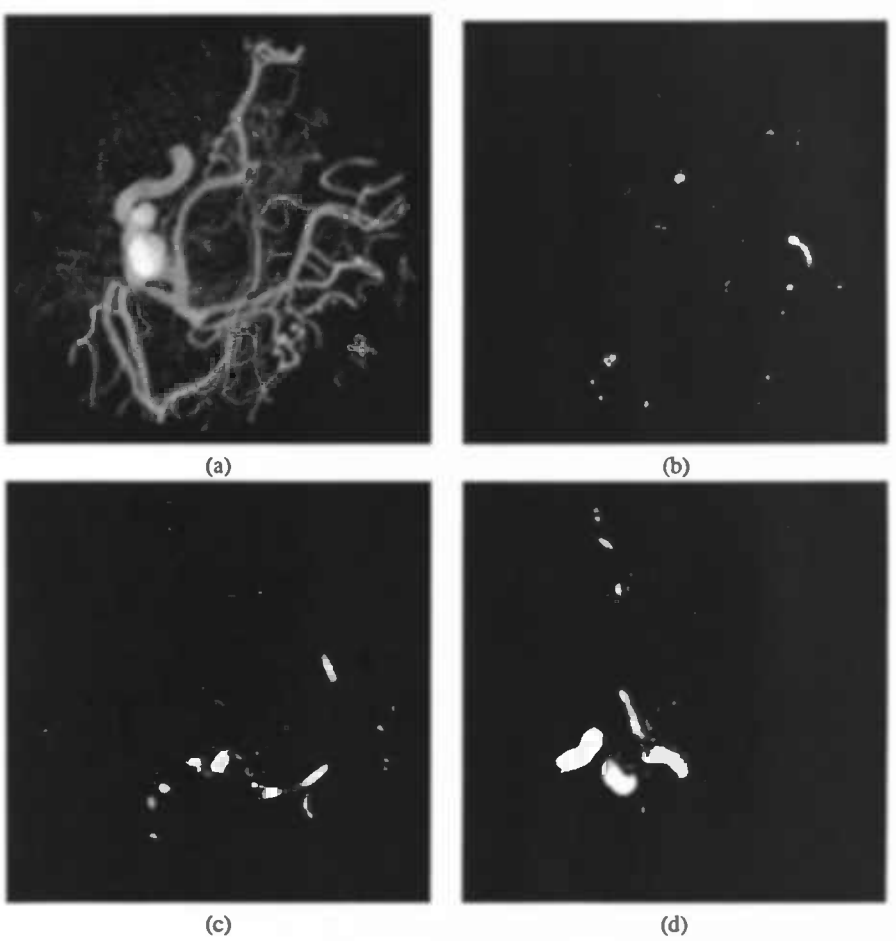


Figure 1.1: The difference between a slice and a volume. (a) A volume rendered 256^3 dataset, (b) a XY slice from the same dataset with $z = 128$, (c) a XZ slice with $y = 128$, and (d) a YZ slice with $x = 128$.

Chapter 2

Filtering

In this chapter a short theoretical introduction to morphological operators and connected filters is given. In section 2.1 we explain the concepts of connected operators on binary and grey scale images. We introduce size distributions (granulometries) in section 2.2. In 2.3 a suitable data structure for attribute filtering, the *Max-tree*, is introduced.

Filtering is a well-known task in image analysis. Filters are used for several purposes, like removing noise from an image, edge-detection, image decompositions or image segmentation.

A special class of operators for filtering images is the class of morphological operators. These are based on non-linear calculus, and they are often used when shape and speed are of importance. We will discuss such operators in detail later. Our purpose now is to discuss the type of filters we will need to extract or enhance certain details in an angiographic image.

In some applications, it is important that an image can be simplified, by removing noise for example, while the contour information is preserved. A type of operators that has this property is the class of connected operators.

2.1 Connected Filters

Mathematical morphology is a set-theoretical approach to binary images. Each foreground pixel (or group of connected foreground pixels) can be treated as a member of a set. In this way, set operations such as union or intersection can easily be applied to an image. All kinds of operators can be constructed using more simple operators.

For example, the *dilation* combines two sets using vector addition. The resulting image M of a dilation of the image X with structuring element B is the point set of all possible vector additions of pairs of elements, one from each of the sets X and B:

$$X \oplus B = \{p \in M : p = x + b, x \in X \text{ and } b \in B\} \tag{2.1}$$

Erosion is the dual operator of dilation and combines two sets using vector subtraction:

$$X \ominus B = \{p \in M : p + b \in X, \text{ for every } b \in B\} \tag{2.2}$$

We now can construct two new operators by combining erosion and dilation: the *opening* is defined as:

$$X \circ B = (X \ominus B) \oplus B \tag{2.3}$$

The *closing* of an image X by structuring element B is defined as

$$X \bullet B = (X \oplus B) \ominus B \tag{2.4}$$

Now we will give an informal definition of a connected component, since connected operators act on connected components of an image: The connected component of a set is a maximal set of points that may be connected by a path included in the set [22]. In the case of digital imaging, connectivity depends on the choice of which pixels are adjacent. There are several cases of connectivity, the most common are 4- and 8-connectivity in the 2D-case and 6-, 18-, or 26-connectivity in the 3D-case.

We now can define a connected operator for sets:

Definition 2.1 *An operator ψ on sets is said to be connected when for any set A, the symmetrical difference $A\Delta\psi A$ is exclusively composed of connected components of A or its complement A^C.*

Even when interpreting the definition intuitively, it becomes clear that connected operators cannot change edges, nor introduce new edges. We see that a connected component from the original is removed completely, or kept at its original location.

Definition 2.2 *A partition of a space E is a set of connected components $\{A_i\}$ which are disjoint $(A_i \cap A_j = \emptyset, i \neq j)$ and the union of which is the entire space $(\cup A_i = E)$. Each A_i is called a partition class.*

If any pair of points in the partition class A_i also belong to an unique partition class B_j we call the partition $\{A_i\}$ *finer* than $\{B_j\}$.

Furthermore, we call a partition of a binary image an *associated partition* if this partition is exactly made of all the connected components in the image and their complements.

In terms of associated partitions it is possible to define connected operators in an alternative way:

Definition 2.3 *An operator ψ on sets is connected, iff. for each family of sets A, the partition associated with $\psi(A)$ is less fine than the partition associated with A.*

Before we can define connected operators for functions (grey-level images), we first have to introduce the grey-level counterpart of connected components, the so-called *flat zones*. Consider a grey-scale image f with domain M. We define the

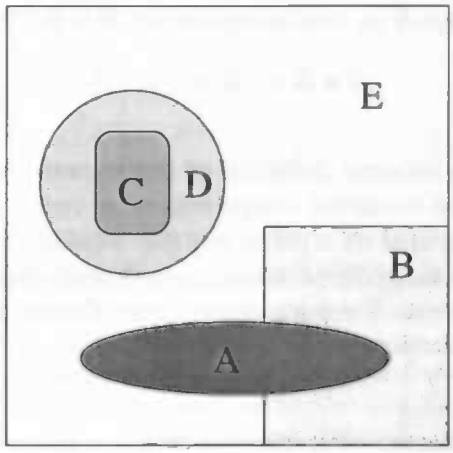


Figure 2.1: Example of an image and its flat zones. All flat zones in the image are labeled by a letter (A-E).

flat zone L_h as a connected component of the set of pixels $\{p \in M | f(p) = h\}$. Less formally, each group of adjacent pixels with the same grey-value forms a flat zone. Note that a single pixel also can be a flat zone (figure 2.1).

Salembier and Serra show that by "stacking" a connected operator for sets can be extended to a connected operator for functions. For example, the volume opening removes - in case of a binary image - all connected components with a volume smaller than a certain threshold λ. When performing a volume opening on a grey-scale image we also need a threshold. In this case all flat zones with a volume smaller than λ are removed.

Consider a binary image $T_h(f)$ which is obtained by thresholding the input f at grey-level h. Let Γ_λ^V be the binary volume opening with scale parameter λ. The grey-scale volume opening γ_λ^V is now given by:

$$(\gamma_\lambda^V(f))(x) = \sup\{h | x \in \Gamma_\lambda^V(T_h(f))\} \tag{2.5}$$

Finally, we can define grey-level connected operators.

Definition 2.4 *An operator Ψ on grey-level functions is connected, if, for any function f, the partition of flat zones of f is finer than the partition of flat zones of $\Psi(f)$.*

2.2 Granulometries

It is possible that we want to extract details at a particular scale. In order to do this, there are (ordered) sets of openings called *size distributions* or *granulometries*. In [30] the following definition is given:

Definition 2.5 *A binary size distribution or granulometry is a set of operators* $\{\alpha_r\}$ *with* r *from some totally ordered set* Γ*, with the following three properties:*

- $\alpha_r(X) \subset X$
- $X \subset Y \Rightarrow \alpha_r(X) \subset \alpha_r(Y)$
- $\alpha_r(\alpha_s(X)) = \alpha_{\max(r,s)}(X)$

for all $r, s \in \gamma$.

The second and the third property define α_r as anti-extensive and increasing, respectively.

An example of use of a size distribution is extracting all details within a certain scale range. Let $r < s$. To create an image g with all the details from image f within scale range $[r..s)$ we get:

$$g = \gamma_r(f) - \gamma_s(f) \equiv \gamma_r(f) - \gamma_s(\gamma_r(f)) \tag{2.6}$$

Now let a scaling X_λ of set X by a scalar factor $\lambda \in \mathbf{R}$ be defined as

$$X_\lambda = \{\in \mathbf{R}^n | \lambda^{-1}x \in X\} \tag{2.7}$$

and a scaling f_λ of a grey-scale image f be defined as

$$f_\lambda(x) = f(\lambda^{-1}x) \forall \lambda^{-1}x \in \mathbf{M} \tag{2.8}$$

An operator σ is said to be *scale invariant* if

$$\phi(X_\lambda) = (\phi(X))_\lambda \text{ or } \phi(f_\lambda) = (\phi(f))_\lambda \tag{2.9}$$

We also can define *shape distributions*. They consist of operators which are scale invariant but not necessarily increasing. In [30] the following definition is given:

Definition 2.6 *A binary shape distribution is a set of operators* $\{\beta_r\}$ *with* r *from some totally ordered set* Γ*, with the following three properties:*

- $\beta_r(X) \subset X$
- $\beta_r(X_\lambda) = (\beta_r(X))_\lambda$
- $\beta_r(\beta_s(X)) = \beta_{\max(r,s)}(X)$

for all $r, s \in \gamma$ *and* $\lambda > 0$.

These operators are not openings since they do not explicitly have the increasingness property. We can define a *grey-scale shape distribution* in a similar way.

A way to provide a shape distribution is by means of *attribute thinnings*, which are based on *connected openings*. A binary connected opening yields the connected component of X containing x if $x \in X$, and $\emptyset$ otherwise. This is extended to the *trivial thinning*, which uses a non-increasing criterion T to decide if a connected set has to be accepted or rejected. The attribute thinning is nothing more than performing a trivial thinning on all connected components of an image. Urbach and Wilkinson [30] show that attribute thinnings are shape-only granulometries or shape distributions (assuming the condition that the attribute is scale invariant holds).

It has been shown that attribute thinnings are useful for shape decomposition. If we define a criterion which is able to measure shape in a way, and we use that as the criterion T, we easily can extract details of a particular shape. This is very useful in the case of vessel extraction, for the reason that there are blood vessels in many different sizes, but the all share the shape property that they are filamentous.

2.3 Maxtree and filtering criteria

An efficient implementation of attribute filters relies on computing both the hierarchy of connected components in the data set, and some attribute for each component to use as a filter criterion.

Salembier *et al.* developed an efficient and versatile data structure to deal with the processing steps involved in anti-extensive connected operators, called the *Max-Tree* [21]. They restrict the use of this data structure to the case of anti-extensive operators ($\forall X, \{\psi(X)\} \subset X$). Extensive operators can be computed by inverting the source image, applying the Maxtree filtering and inverting the result.

2.3.1 Maxtree representation

Let $M \subseteq \mathbb{R}^n$ be some image domain ($n = 2$ for images and $n = 3$ for volumes) and $f : M \rightarrow \mathbb{R}$ the grey scale image under study. Implicitly we assume the

existence of some neighborhood graph (i.e. grid) on M.

A peak component P_h^k is a connected component of $T_h(M)$. The number of connected components is finite so can be enumerated. The superscript of P is nth component at level h. This way, every peak component can be uniquely identified. The Max-Tree is a tree representation of an image. Every node C_k^h in a Maxtree is derived from P_h^k using the following equation:

$$C_k^h = \{x \in P_k^h | f(x) = h\}. \tag{2.10}$$

The tree structure is defined as follows. A node a is a child of node b if $F_a \subset F_b$ and $h_a > h_b$. The root node contains M since every data point has a value h greater than or equal to the minimum.

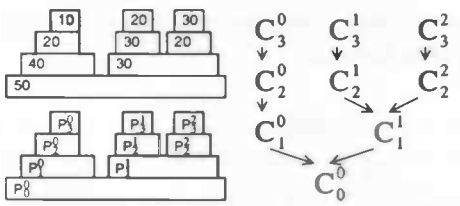


Figure 2.2: The peak components of a dataset (bottom left), the corresponding attributes (top left) and the Maxtree (right)

A graphical representation of this can be seen in the bottom left of figure 2.2. Every block is a peak component and the subscript of P denotes h. The right of figure 2.2 shows the corresponding Maxtree.

2.3.2 Attributes

Having ordered the voxels in a tree structure is one thing, to be able to do something useful with it is another. A certain labeling of the nodes is necessary. These labels are called attributes. Any attribute can be used, however, for computational efficiency incremental calculation, i.e. easily updated voxel by voxel, is nice. Some examples are:

- Volume
- Variables extracted from bounding shapes like area or roughness.
- Any moment based approach.
- Perimeter

More of these attributes are discussed by Urbach[29]. Since we focus on blood vessels, which are elongated structures, we need a method to calculate the "vesselness" of an object. Moment of inertia is a criterion that comes close to the desired result. For a given volume the moment of inertia is minimal for a sphere and increases rapidly as the object becomes more elongated.

For a set of voxels F the moment of inertia is defined as:

$$I(F) = \frac{V(F)}{4} + \sum_{x \in F} (x - \bar{x})^2. \tag{2.11}$$

in which $V(F)$ is the volume of F. The first term denotes the moment of inertia of individual voxels. In the case of a 3D shape the moment of inertia scales with the size to the power of five. It is by itself not scale invariant. Volume scales to the power of three. By combining these two we can calculate a shape-dependent but scale-invariant factor S defined as:

$$S(F) = \frac{I(F)}{V(F)^{\frac{5}{3}}}. \tag{2.12}$$

S is minimal for a sphere shape and maximal (in case of a connected component) for a single straight line.

2.3.3 Filter algorithms

Many of the classical attributes which are used for filtering, as well as their resulting operators, are increasing. For an operator ψ this means: $\forall x \leq y, \psi(x) \leq \psi(y)$. If the attribute $\mathcal{M}$ is increasing, that is, when for an attribute yields that $A \subseteq B \Rightarrow \mathcal{M}(A) \leq \mathcal{M}(B)$, the *criterion sequence* (obtained by scanning successively all ancestor nodes of a maximum going down to the root node) also is increasing. Defining the level h where the attribute is higher than λ_f is simple. All nodes for which $\mathcal{M}(C_h^k) < \lambda_f$ are removed, and the corresponding pixels are moved to the first ancestor node such that $\mathcal{M}(C_h^k) \geq \lambda_f$.

In the case of non-increasing attributes the decision rules are less simple, due to the criterion sequence fluctuating around λ_f. These strategies can be divided into two classes:

Pruning strategies Which remove all descendants of a node when this node is removed.

Non-pruning strategies in which the parent pointers of children of a node are updated to point at the oldest "surviving" ancestor of the node.

Salembier describes four different rules to filter the tree: the pruning strategies *Min*, *Max* and *Viterbi* and the non-pruning *Direct* strategy. In addition Wilkinson and Urbach[30] introduced another non-pruning strategy, called the *Subtractive* decision. The strategies are as follows:

Min decision Node C_h^k is preserved when $\mathcal{M}(C_h^k) \geq \lambda_f$ and if all its ancestors are also preserved.

Max decision Node C_h^k is removed when $\mathcal{M}(C_h^k) < \lambda_f$ and if all its descendants are also removed.

Viterbi The removal and preservation of nodes is considered as an optimization problem. For each leaf node the path with the lowest cost to the root is taken, where a cost is assigned each transition. We will not look into this method.

Direct decision - Node C_h^k is removed when $\mathcal{M}(C_h^k) < \lambda_f$. Its ancestors and descendants are unaffected.

Subtractive decision Same as the direct decision, but the grey level of descendants is lowered by the difference between the grey level of the current node and the grey level of its first ancestor which meets the criterion.

The top left of figure 2.2 shows an example of an attribute labeling of a Maxtree. Figure 2.3 shows the resulting Maxtrees using the different strategies. An example of filtered images can be seen in figure 4.2 on page 71.

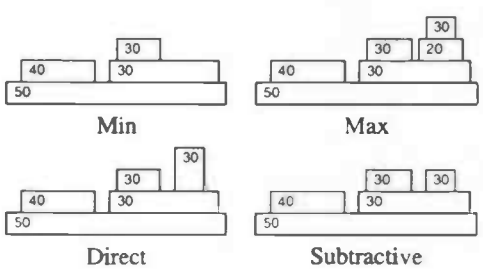


Figure 2.3: The result after filtering the signal in fig. 2.2 with four different decision rules, using $\lambda_f = 25$ as the attribute threshold.

Chapter 3

Segmentation

In chapter 3 we discuss the concept of segmentation. We begin with a short explanation of what segmentation is. In section 3.1 we will consider the core segmentation method of this report, Robust Automatic Threshold Selection (RATS). Then, a classification of segmentations is described in section 3.2. Some measures for the quality of segmentations are introduced in section 3.3, after which we will give some examples of segmentation techniques proposed in literature (section 3.4). We finish this chapter in section 3.5 with a thorough description of the segmentation methods, modifactions and additions we implemented, and show, compare and discuss the results.

After the filtering a choice has to be made which pixel belongs to which object. A binary dataset has to be created with the same dimensions of the original dataset. In this new dataset the foreground pixels (with a true value) mark the position of the objects of interest.

Segmentation can be done by thresholding which is one of the most basic operators in image analysis. If a pixel value is above the threshold it will be true, otherwise it will be false. Determining the threshold value can be done manually, but we will use an automatic threshold operation.

3.1 Robust automatic threshold selection

We will use a 3D conversion of Robust Automatic Threshold Selection (RATS) which was first developed by Kittler et al. [9]. The original 2D RATS uses image statistics to determine the threshold value. Let $p(x, y)$ be the grey level of the pixel at position (x, y). Now we can define the gradient operator $e(x, y)$ as:

$$e(x,y) = \max \left\{ \begin{array}{l} |p(x+1,y) - p(x-1,y)|, \\ |p(x,y+1) - p(x,y-1)| \end{array} \right\} \tag{3.1}$$

This gradient operator $e(x, y)$ is used in the computation of two important image statistics. Kittler et al. showed that for an area A, the optimum threshold can be determined by evaluating the statistic:

$$T = \frac{\sum_A e(x,y)p(x,y)}{\sum_A e(x,y)} \tag{3.2}$$

The other statistic can help telling us how likely it is that the area A contains an

edge:

$$C = \sum_{A} e(x, y) \tag{3.3}$$

To prevent that also all noise will be considered as edges, the threshold T is only used when the value for C is far enough above the value which should be expected in the case of flat noise. Because C is a part of T, it costs no additional computations to determine the value of C. When noise causes a bias in T, a modified form of the gradient can reduce this. The modified form $w(x, y)$ is defined as:

$$w(x, y) = \max \begin{cases} e(x, y) & \text{if } e(x, y) > \lambda_n \cdot \sigma \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

in which σ is the expected noise (per pixel) and λ_n is a multiplication factor limiting the sensitivity of the method to both noise-related and object-related gradients.

Determination of a single global threshold for an image is, especially in the case of a non-uniformly illuminated image, unsatisfactory. It is better to divide the image into smaller square parts and determine independent thresholds. RATS will produce an optimal threshold when the number of background and foreground pixels in the computation is equal. A decreasing area size makes this more probable, so RATS is excellently appropriate for local thresholding. In [9] a quadtree is proposed for local thresholding. The image is subdivided in a quadtree of a certain number of levels (figure 3.1). For the lowest levels the statistics C and T are computed, and if the edge criterion C is large enough, local threshold T is used, otherwise the parent threshold (if large enough) is used.

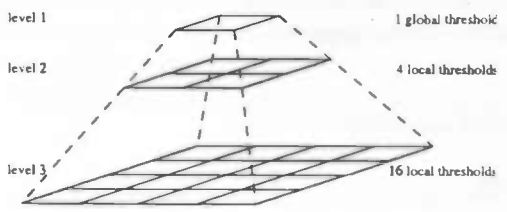


Figure 3.1: The subdivision of the image in subimages or quadtree is the data structure used for local thresholding.

Instead of the standard gradient operator, also other edge detectors can be used. Wilkinson [32] investigated several edge detectors and found that the quadratic Sobel filter is the overall winner in the context of RATS. How this operator is used will become clear later in section 3.5.1.

The conversion to 3D is straightforward: we only have to use a 3D version of the gradient operator instead of a 2D one.

The 3D dataset will be subdivided into an octree of cubes. At the highest level of the hierarchy lies the entire dataset, which is subdivided into eight "child" cubes, each of which in turn are divided into eight, etc. down to subdivisions with sizes in the orders of those of the objects of interest. If a cube at the lowest level cannot be assigned a threshold, its "parent" is consulted recursively if necessary to the highest level.

There are a few reasons which made us decide to implement RATS in 3D. One of them is that, as far as we know, RATS has never been applied to threshold 3D images. Furthermore, despite the fact that several methods have been used for thresholding angiograms, RATS has not. This, and the simplicity and speed of RATS, lead us to try it in this application.

3.2 Other methods

Most applications need some form of user interaction to define the threshold value. We investigated other methods of automatic thresholding.

Besides RATS, several other approaches for (semi-)automatic segmentation have been developed and proposed. We will give a short overview of segmentation methods (with respect to vascular or vessel segmentation), discuss how segmentations can be compared and how the quality can be measured. Finally we discuss some methods in more detail. In section 3.5.4 we decide which segmentation will be implemented.

Aylward et al. [2] present a short overview of existing methods for vessel modeling or segmentation and divide the approaches into three types:

- Centerline based modeling
- Spatial filtering
- Voxel Labeling

3.2.1 Centerline extraction

Methods based on *centerline extraction*, use the knowledge that centerlines of vessels often appear brightest. The width of the vessel is then determined by a response function. Aylward [2] performs a multi-scale traversal of the centerline

from an initial point on a vessel, and from this centerline the width of the vessel is estimated. Niessen and Frangi [7] also use a centerline method but that work depends on thresholds and uses a pair of points to define a centerline and therefore is more difficult to automate.

3.2.2 Spatial Filtering

Spatial filtering methods for vessel segmentation include anisotropic diffusion, matched filtering, morphological operations and level-set evolution. Often multi-scale filtering is used, which consists of convolving the images at multiple scales with Gaussian (orientation selective) filters, and analyzing the eigenvalues of the Hessian matrix at each voxel in the image to determine the local shape of the structures in the image. The Hessian matrix is given by:

$$H = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{3.5}$$

The partial second deratives of the image $I(x, y, z)$ are represented as I_{xx}, I_{xy} and so on. This matrix describes the second-order structure of local intensity variations around each point of a 3D image.

Results [7] show that these filters not always give maximum response on the vessel axis, although they should according to the theory.

Some methods use the output of the multi-scale filter to define a new image with enhanced vessels and suppressed noise and non-vessel-structures. This new image can be processed further, by thresholding or segmentation using e.g. an active contour method. Other methods use the obtained eigenvalues to determine the centerlines of vessels. Using response functions, the diameter at each voxel can be estimated, and with the centerlines and these diameters a surface model can be constructed.

Another (straightforward) approach is to rely on image contrast, applying an intensity threshold, followed by morphology analysis [16]. A non-uniform distribution of contrast agent however can cause significant variation in vessel intensity [35].

Other methods are based on three-dimensional thinnings. Masutani claims they fail to manage abnormal vessel structures such as aneurisms [16]. Another problem is that it is hard to keep the topology correct. Despite the fact that most thinnings are topology-preserving, this is hard to achieve in practice without manual correction. The main reason for this is low resolution of medical imaging modalities compared to the size of blood vessels. It is possible this causes partial volume effects

which connects vascular structures to other organs like bones in X-ray CT. When MRA (Magnetic Resonance Angiography) is used, flow-void (lack of signals in vessels), often causes disconnection of extracted vascular objects. More recently, deformable models have been developed to overcome this problem. In fact, they are useful for larger structures such as brains and livers, since they assume global smoothness, which is not the case for vessels having bifurcation structures.

Other methods work with differential geometry. The MRA image is in this case treated as a hyper surface of 4D space whose extrema of curvature correspond to vessel centerlines. Also deformable model approaches have been applied. The final segmentation is reached by iteratively optimising an energy function on a initial boundary estimate. Lorigo's approach [14] is based on this.

3.2.3 Voxel labeling

Voxel labeling is often done by statistical pattern recognition. Skeletonization of voxel models is used to generate center-line models. Examples are statistical approaches in which Gaussian or Rician [3] intensity distributions are assumed for background and vessel intensities, and the expectation maximization (EM) algorithm is applied to find appropriate thresholds for classification. Also, thresholding is a simple form of voxel labeling.

3.3 Quality of segmentations

Segmentation quality is computed on the basis of two properties: the similarity and disparity between pixels.

All segmentation techniques are extremely sensitive to the selection of certain parameters and it is generally considered to be very difficult to design a segmentation algorithm which is independent of them. Quality of a segmentation can only be judged in comparison to human expectations or a ground truth.

The problem with measuring segmentation is that the "ideal" segmentation is most often not available in cases of medical data. Phantoms or artificial data provide a ground truth, but in case of medical data the ideal segmentation does not exist. Gerig et al. [8] have developed a tool, VALMET, which is able to read an original 3D-image to be segmented and one or more segmentations of that image. In order to see the differences between the segmentations, various metrics are calculated. Below is a summary of the metrics used.

Volumes: The most easily measurable are the volumes of the distinct segmented

structures. However, comparing volumes does not take into account regional differences and where they occur.

Volumetric overlap: The subject and the reference segmentation are compared voxel by voxel and false positives, false negatives, true positives and true negatives are calculated. Examples of measures are intersection of subject (S) and reference (R) divided by the union $(S \cap R)/(S \cup R)$ or intersection divided by reference $(S \cap R)/R$. Both measures yield 0 for perfect disagreement and 1 for total agreement. A disadvantage is that the overlap depends on the size and shape of the object and is related to image sampling.

Probabilistic distances between segmentations: When there is no *absolute ground truth* by manual segmentation, only a "fuzzy" segmentation is possible. The developed measure is derived from the normalised L_1-distance between two probability distributions:

$$POV(A,B) = 1 - \frac{\int |P_A - P_B|}{2 \int P_{AB}} \tag{3.6}$$

with P_A and P_B the probability distributions representing the fuzzy segmentations and P_{AB} the pooled joint probability distribution.

Maximum Surface Distance (Haussdorff distance): this metric defines the largest difference between two contours and is computationally very expensive. All points of one contour are compared to all points of the other contour. However, this metric is extremely sensitive to outliers, which can be a drawback.

Mean absolute surface distance: This metric represents the *average* distance between two surfaces. It integrates over under- and over-estimation of a contour and results in an L_1-norm with intuitive explanation. The mean absolute distance does not depend on the object size, but it needs existing surfaces as a prerequisite.

Interclass correlation coefficient: this is a common measure of reliability of segmentation. It calculates the ratio between the variance of a normally distributed population and the "population of measurements". For example, let σ_b^2 be the variance of the population and σ_0^2 the variance of the rater. The intraclass correlation is now defined as $\rho = \frac{\sigma_b^2}{\sigma_b^2 + \sigma_0^2}$

All these metrics are implemented in VALMET, which could be used to compare RATS with other segmentations.

Furthermore, Levine and Nazif [11] have designed an error measure concerning two-dimensional segmentation, which easily can be extended to the three-dimensional case. Essentially, this is the 2D-distance between two segmentation outputs. The larger the distance, the more the two segmentations differ.

Since we deal with three-dimensional images, we generalize the measure given by Levine and Nazif to the 3D-case:

Consider two outputs, a reference segmentation output containing the set of N regions $\{R_1, R_2, ..., R_N\}$ having volumes $\{V_1, V_2, ..., V_N\}$ and a test segmentation output $\{\tau_1, \tau_2, ..., \tau_M\}$ having volumes $\{\theta_1, \theta_2, ..., \theta_M\}$.

For each region τ_j, find the region R_k, so that the volume of the intersection $\tau_j \cap R_k$ is maximal. We now can define the **under-merging error** as:

$$UM = \sum_{j=1}^{M} \frac{\{V_k - (\tau_j \cap R_k)\}(\tau_j \cap R_k)}{V_k} \tag{3.7}$$

The **over-merging error** is given by:

$$OM = \sum_{j=1}^{M} \{\theta_j - (\tau_j \cap R_k)\} \tag{3.8}$$

Both measures result in zero when the test and reference segmentations are identical, and the higher bound is equal to the volume of the image minus one. Because of this, the measures can be normalised and combined to compute a composite measure:

$$M_1 = \sqrt{\frac{UM^2}{V_I} + \frac{OM^2}{V_I}} \tag{3.9}$$

or

$$M_2 = \left|\frac{UM}{V_I}\right| + \left|\frac{OM}{V_I}\right| \tag{3.10}$$

where V_I is equal to the volume of the image.

3.4 Some examples of segmentation techniques

3.4.1 Vessel segmentation for visualisation of MRA with blood pool contrast agent

Young et al. [35] propose a front propagation approach. Vessels are only extended in one direction (their axis). This property might be useful to distinguish vessel voxels. Two types of vesselness response are considered:

- **Multi-scale vessel filter** - The eigenvalues of the Hessian matrix (3.5) reflect the degree of curvature. A response of a vesselness filter $F(\mathbf{p})$ can be defined using these eigenvalues. For details see [35]. The multi-scale response is defined as the maximum response over an appropriate scale range, using a scale space representation of the image. This filter does not need a priori knowledge of the orientation, but the accuracy of the radius-estimate is quite low, due to increasing complexity and memory requirements when the number of scales increases.

- **An adaptable cylinder model** - A cylinder-parametrization can be used to represent a short section of a vessel. Therefore, a vessel can be modelled as a sequence of those cylinders, where agreement between the model and the image can be measured by integrating the image gradient across the surface. An alternative response is based on model adaptation. A set of feature points along the surface normals are detected, and while cylinder parameters are updated new cylinders are placed on the surface to form a model.

The centerline of the vessels is now determined by selecting a seed point. The time field $T(x)$ is defined zero at the selected point, and infinity at all other (unselected) points. The propagation of the front proceeds as follows:

1. Unselected voxels bordering selected voxels are marked as *border*, their time values are updated according to: $|\nabla T|F = 1$. In this governing equation T is the time-field and F is the speed function (or vesselness response filter). Two speed functions have been proposed in the orginal article.

2. The border-voxel with the lowest time-value is moved to the selected region.

Since cross-sections of a vessel are not always circular, a deformable model is constructed, using the centerline and radius estimates. In this way, the segmentation can be refined. The vessel, represented as a triangle mesh, is optimized by minimizing an energy function which is composed of an external, image-related energy term ($E_{ext}(\mathbf{x})$) and an internal, shape-related term ($E_{int}(\mathbf{x})$). The (composite) energy function is now defined as

$$E(\mathbf{x}) := \alpha E_{int}(\mathbf{x}) + E_{ext}(\mathbf{x}) \tag{3.11}$$

Minimization is done by using the conjugate gradients method.

One of the drawbacks is border selection near bifurcations.

3.4.2 Vascular shape segmentation and structuring extraction using a shape-based region growing model

The method proposed by Masutani et al. [16] for vascular shape segmentation and structure extraction consists of two main parts:

- The initial shape is acquired by thresholding
- After that, region-growing in combination with mathematical morphology and local shape is performed.

First of all, a threshold is applied to the original image to obtain an initial binary shape. After this, a seed is chosen as a starting shape for iterative dilation.

The model iterates two processes, simple growing and growth-front smoothing. The former is bounded space dilation and the latter is bounded space closing, both with different kernels. A bounded space operation is a mathematical morphological operation, in which a constant boundary shape B is given which is never altered by any operation. The bounded space dilation is given by:

$$X \bullet_B K = X_N = (X_{N-1} \bullet K) \cap B \tag{3.12}$$

with X_N the shape at iteration N, K the growth-kernel and B representing the binary mask.

Other optional processes and growing conditions are mentioned as well, such as morphological size upper limitation and directional growing.

The unit of growth is a cluster, which is a compact set of voxels which belong to the same growth generation. C_0 denotes a seed voxel and C_N is a differential region after the N-th growth step. If voxels after bounded space dilation are disconnected, the regions must be distinguished as individual clusters. In this case we deal with bifurcation, and an alternative cluster growth is defined. While clusters are generated attributes of clusters are measured for latter use.

There is a short discussion in the article about the choice of kernel sizes and shapes. Basically, the kernel shapes for simple growing as well as for growth-front smoothing are spherical. They have a small kernel radius. However, directional kernels should be larger since small kernels make directional control difficult.

Finally, clusters are grouped based on their attributes. The clusters, and also their connection graph are obtained as an extracted shape. A branch is defined as a set of connected clusters, which have connectivity numbers smaller than three. With smaller kernels, smaller convex shapes are detected as bifurcation.

An important property is the controllability of the bifurcation detection by the size of the growth-front smoothing kernel.

3.4.3 Segmentation and visualisation of curvilinear structures in medical images

In this paper [23] a practical and general-purpose approach for enhancement and multi-scale integration of curvilinear structures in 3D medical images is described.

A line filter is used, based on the eigenvalues of the Hessian matrix, which is given by (3.5). The eigenvalues are ordered in magnitude, $\lambda_1 > \lambda_2 > \lambda_3$ and their corresponding eigenvectors are $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$. The ideal bright 3D line in the z-direction is given by

$$I(x, y, z) = e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{3.13}$$

In the ideal case $\lambda_1 \simeq 0, \lambda_2 \simeq \lambda_3 << 0$ Based on these conditions, $\sqrt{\lambda_2 \lambda_3}$ and $\min(-\lambda_2, -\lambda_3) = -\lambda_2$ were proposed as a measures of similarity to a line structure.

To prevent a high filter response for sheet-like structures (λ_2 small), and to let the filter response decrease when deviating from zero of λ_1, a similarity measure is used:

$$L = f(\lambda_1; \lambda_c) \times \lambda_c \tag{3.14}$$

with $f(\lambda_1; \lambda_c)$ a function decreasing with the deviation from zero of λ_1 and $\lambda_c = \min(-\lambda_2, -\lambda_3) = -\lambda_2$. f is chosen in such a way that it removes noise and other unimportant structures as well as it makes a fragmented curvilinear structure continuous.

In the case of multi-scale responses a trade-off has to be made between enhancing thin structures and increasing noise. A normalized response function is defined using a convolution with an isotropic Gaussian function. This works fine for large widths of lines, but with smaller widths the problem of high filter response for both small structures and noise components arises. Therefore another method of multi-scale integration is used:

$$\max_i \frac{1}{n_i} L_i(x, y, z) \tag{3.15}$$

where n_i is the standard deviation of noise amplitudes at scale i, and L_i is the line filter response at scale i. An estimation of n_i can be done using the region which includes typical structures which have to be removed and does not contain curvilinear structures of interest.

3.4.4 Highly automated segmentation of arterial and venous trees from three-dimensional MRA

Stefancik and Sonka propose a highly automated method for segmentation of arterial and venous trees [25]. Shortly, the algorithm consists of five steps:

1. Binary mask generation
2. Tree-structure generation
3. Optimal vessel path calculation
4. Vessel segment labeling
5. Conflict resolution

Binary mask generation - First of all, *a priori* knowledge is used to determine a grey-value at which the image can be thresholded. Knowing that vessel-structures occupy around 5% of the data set by volume, a 95% threshold is employed with a value derived from a grey level histogram. The binary mask which is obtained can still contain some artifacts. To remove these, a seeded region growth algorithm is needed, in which a seed point is chosen in the vessel structure and connected voxels above the threshold value are labelled.

Tree-structure generation - The vascular segments are stored in a tree, and each segment represents a section between two subsequent bifurcations. In the growth-front tree generation, a conditional bounded space dilation is used as in (3.12).

The dilation process grows from the seed point to all 26 neighbours for each iteration. If the growth-front reaches a bifurcation (and hence becomes discontinuous), this process is repeated for the bifurcated segments.

Optimal vessel path calculation - The resulting tree is quite complex and contains falsely detected bifurcations. This is the reason spatial information about vessel paths is needed. Skeletons are used to gain this information, using an dynamic programming path cost maximization. Cost elements, determined by a directional factor, voxel depth and penalty are calculated and placed in a cost matrix. From this, an optimal path is calculated to extract centerlines.

Vessel segment labeling - Since the artery or vein label is known for each seed point in the path search, the label is propagated along the paths. This may cause conflicting labels, which are solved by the conflict resolution step.

Conflict resolution - In case of conflicting labels a resolution function is computed. Absolute and relative distances of a voxel to both paths in the segment are calculated and a decision function is made. In this way, when a segment is labeled as belonging to both artery as vein, a decision can be made.

The segmentation is highly automated, just a few minutes of user interaction are needed. After thresholding seed points for the artery and the vein are chosen. Furthermore, for each vessel segment that needs to be separated and labeled, a point has to be selected at the end of the desired vessel section. Any mislabeled regions have to be corrected afterwards.

3.5 Experiment

In this section we take a closer look at the implementation of some of the ideas mentioned above.

We tested our programs with several datasets in `sff`-format. Since we will refer several times to those datasets we give a table with some characteristics of these datasets (table 3.1), including a *refname* which we use in the rest of the chapter for referring to these datasets. The characteristics are supposed to be obvious enough so we will not explain them in more detail.

3.5.1 3D Implementation of RATS

The existing version of `rats.c` is able to segment 2D images using the RATS-method. After compilation of the source code the usage of the program is trivial. An input image is given to the program, which converts it to a segmented output image. The file format used is `pgm`, which stands for *Portable Greyscale Map*.

In short, the program reads in an input image, a noise parameter and the number of levels of the quadtree that is used for thresholding. Quadtrees are filled with weights (the denominator of threshold statistic T) and sums (the numerator of threshold statistic T), which in turn are used for the threshold computation. Weights are computed using a quadratic 3×3 Sobel kernel. The choice for this kernel has already been explained in section 3.1. Thresholds for the leaf centroids are recursively computed and interpolated for all pixels, so the thresholded image can be obtained by applying that threshold.

The extension to 3D is quite trivial. The most important changes which have to be made are:

- using a 3D gradient operator instead of a 2D one
- using a octree instead of a quadtree
- modify routines for reading and writing images in 3D format

Refname	MRA1
Filename	`angio.sff`
Type	short
Dimensions	128 × 128 × 62
Total # of voxels	1015808
Total # non-0-voxels	26341
Total # 0-voxels	989467
Total # of greyvalues	202
Min greyvalue	0
Max greyvalue	254
Avg greyvalue	0.607936
Refname	MRA2
Filename	`angiolarge.sff`
Type	long
Dimensions	256 × 256 × 124
Total # of voxels	8126464
Total # non-0-voxels	6152597
Total # 0-voxels	1973867
Total # of greyvalues	1152
Min greyvalue	0
Max greyvalue	1322
Avg greyvalue	19.1226
Refname	CTA1
Filename	`vessels.sff`
Type	short
Dimensions	256 × 256 × 256
Total # of voxels	16777216
Total # non-0-voxels	168948
Total # 0-voxels	16608268
Total # of greyvalues	255
Min greyvalue	0
Max greyvalue	255
Avg greyvalue	0.504499

Table 3.1: Datasets used

Handling the 3D images will not be possible with the pgm-format, since this is only for 2D images. The datasets we usually work with are `sff`-files.

We will describe the most important changes made to the program.

A slight modification to the image data type has to be made. Not only does another dimension have to be added to the `Image2D`-type, we also change the `Pixel`-type from `unsigned char` to `unsigned short`.

Another important change is the use of the 3D Sobel operator instead of the 3×3 2D Sobel operator. The former operator applies a $3 \times 3 \times 3$ kernel to compute each of the partial deratives. The kernel used for the x direction is:

-1	-3	-1
-3	-6	-3
-1	-3	-1

$x-1$

0	0	0
0	0	0
0	0	0

x

1	3	1
3	6	3
1	3	1

$x+1$

The kernel used for the y direction is:

1	3	1
0	0	0
-1	-3	-1

$x-1$

3	6	3
0	0	0
-3	-6	-3

x

1	3	1
0	0	0
-1	-3	-1

$x+1$

And the kernel used for the z direction becomes:

-1	0	1
-3	0	3
-1	0	1

$x-1$

-3	0	3
-6	0	6
-3	0	3

x

-1	0	1
-3	0	3
-1	0	1

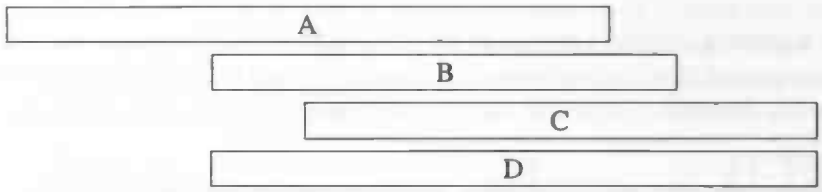
$x+1$

Using these kernels the gradients in x, y and z-direction can be computed. After that the square Sobel gradient is computed and normalized by dividing by 484 (22^2).

Another change worth mentioning is the extension from bilinear to trilinear interpolation. Since the thresholds are assigned to the center pixels, values of the other pixels have to be computed using trilinear interpolation. Trilinear interpolation is the name given to the process of linearly interpolating points within a box given values at the vertices of the box.

Consider an unit cube with vertex values denoted $V_{000}, V_{100}, V_{010}, ..., V_{111}$. When performing trilinear interpolation the value at position (x, y, z) within the cube will be denoted V_{xyz} and is given by:

$$
\begin{aligned}
V_{xyz} =& V_{000}(1-x)(1-y)(1-z) + V_{100}x(1-y)(1-z) + \\
& V_{010}(1-x)y(1-z) + V_{001}(1-x)(1-y)z + \\
& V_{101}x(1-y)z + V_{011}(1-x)yz + \\
& V_{110}xy(1-z) + V_{111}xyz
\end{aligned}
\tag{3.16}
$$



(3.17)

$$
\begin{aligned}
H_c = & \frac{1}{2n}\sum_{l=0}^{n}(-1)^l(n-l)^{p-2} \sum_{l_1+\cdots+l_p=l} \prod_{i=1}^{p}\binom{n_i}{l_i} \\
& \cdot [(n-l)-(n_i-l_i)]^{n_i-l_i} \cdot \left[(n-l)^2 - \sum_{j=1}^{p}(n_i-l_i)^2\right].
\end{aligned}
\tag{3.18}
$$

This way an interpolated value of every voxel in the volume can be computed using the known values at the center pixels.

Results

The RATS3D method is tested with several values for the number of levels in the statpyramid and the noise parameter η. For practical purposes, the maximum number of levels of the octree should be limited to 8. When this number exceeds 8 too much memory is needed. This is not very surprising because at 8 levels 2^{21} local thresholds are determined.

The noise parameter η also can be set. Pixels with gradients below $\lambda_n \cdot \eta$ are not used in the computation of the threshold (equation 3.2). The recommended value for λ_n in the 2D-case is 3.0 [32]. It appears that in the 3D case λ_n should be a bit higher. We did not obtain these values theoretically, but after trying different values for η and looking at the results, it visually becomes clear which values are to be preferred above other values (note that raising η has the same effect as raising λ_n). A λ_n which is too low yields a too noisy image; the segmented vessel-system is hardly visible due to all noise (foreground) pixels. A too high λ_n makes the program treat some small or thin parts of the vessel-system as noise, and thus removes too many. An example of this is given in figure 3.2.

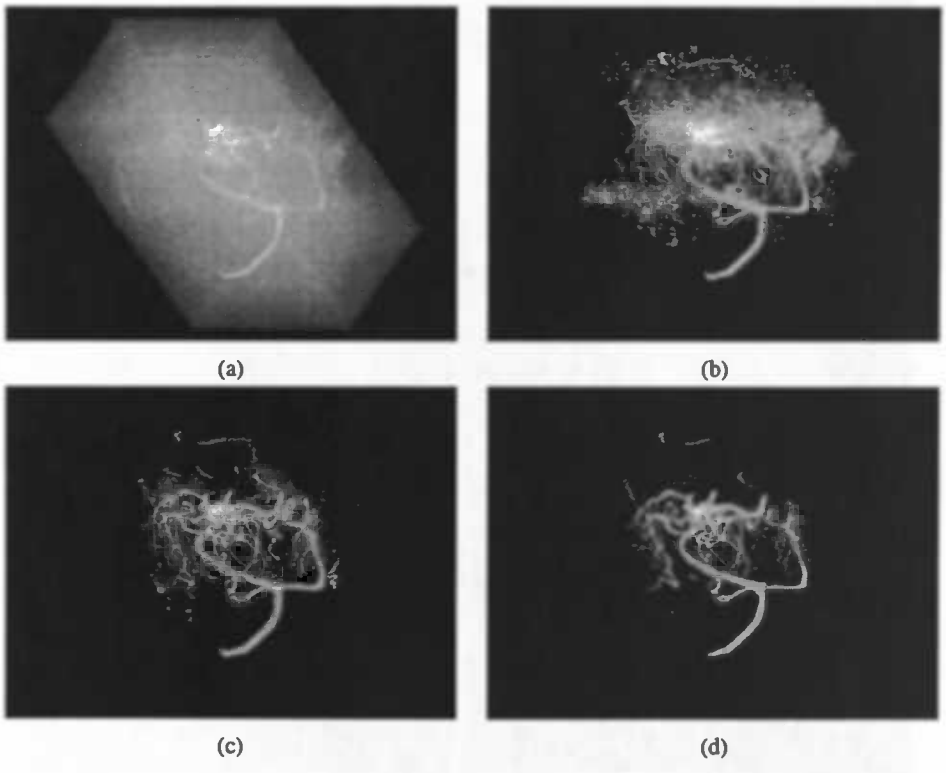


Figure 3.2: The effect of changing the noise parameter η: (a) the original volume; three segmentations with (b) $\eta = 10$, (c) $\eta = 25$ and (d) $\eta = 40$

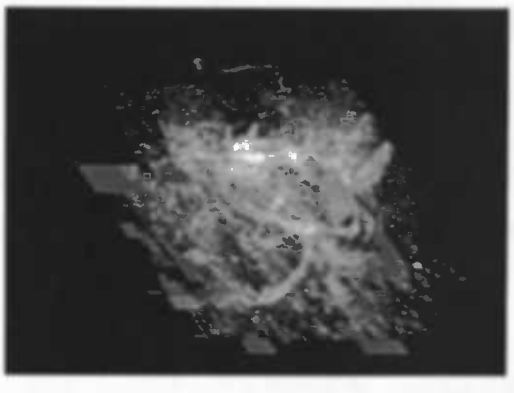


Figure 3.3: White block effects: a problem caused when the edge criterion C is lower than the expected noise level.

When segmenting datasets using relatively high levels in the octree, we see "white blocks" (figure 3.3). This means that a parent threshold is used (possibly) several times, from which we can conclude that the edge criterion C is lower than the expected noise level. To prevent this the value of C should be higher, which can be

established by raising η. However, an optimum value is not found yet. It is difficult to find a value for λ_n such that the noise is removed from an image sufficiently while thin structures are preserved. Since the blocks mainly occur in non-vessel areas, using `rats3d` on datasets filtered on vessel-like structures are likely obtain better results.

At this time, `rats3d` has been tested on 3 different datasets. When segmenting the smallest one, **MRA1**, best results are obtained using octrees with levels 2 till 4, and η between 6 and 20 (figure 3.4 (a)). In the larger version of **MRA1**, **MRA2**, octree levels of 5 and lower give acceptable results. The noise level η, on the other side, has to be raised to 20 and higher.

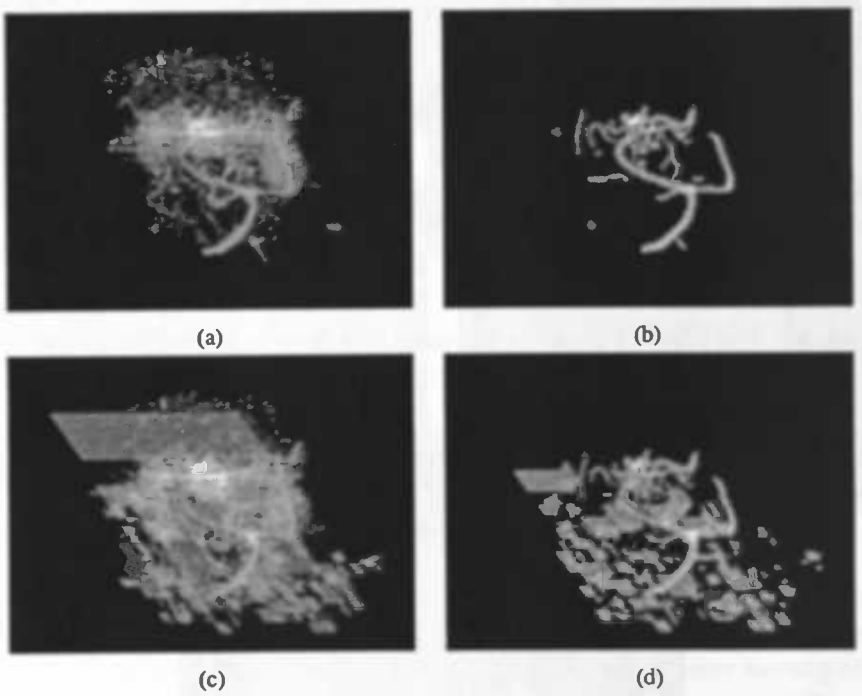


Figure 3.4: Examples of segmentations by standard RATS. (a) **MRA1**, $\eta = 5$, octree size=4. (b) Filtered version of **MRA1** ($\lambda_f = 2$), $\eta = 5$, octree size=4. (c) **MRA1**, $\eta = 5$, octree size=6. (d) Filtered version of **MRA1** ($\lambda_f = 2$), $\eta = 5$, octree size=6

Datasets which are preprocessed by attribute filtering give -not surprisingly- better results(figure 3.4 (b)). The rats-segmentations of the filtered images contain significantly less noise, but it has to be said that the white blocks-effect is still there (figure 3.4 (d)). The improvement of the results on larger volumes like **MRA2** is even bigger, which is visible in figures 3.5 and 3.6.

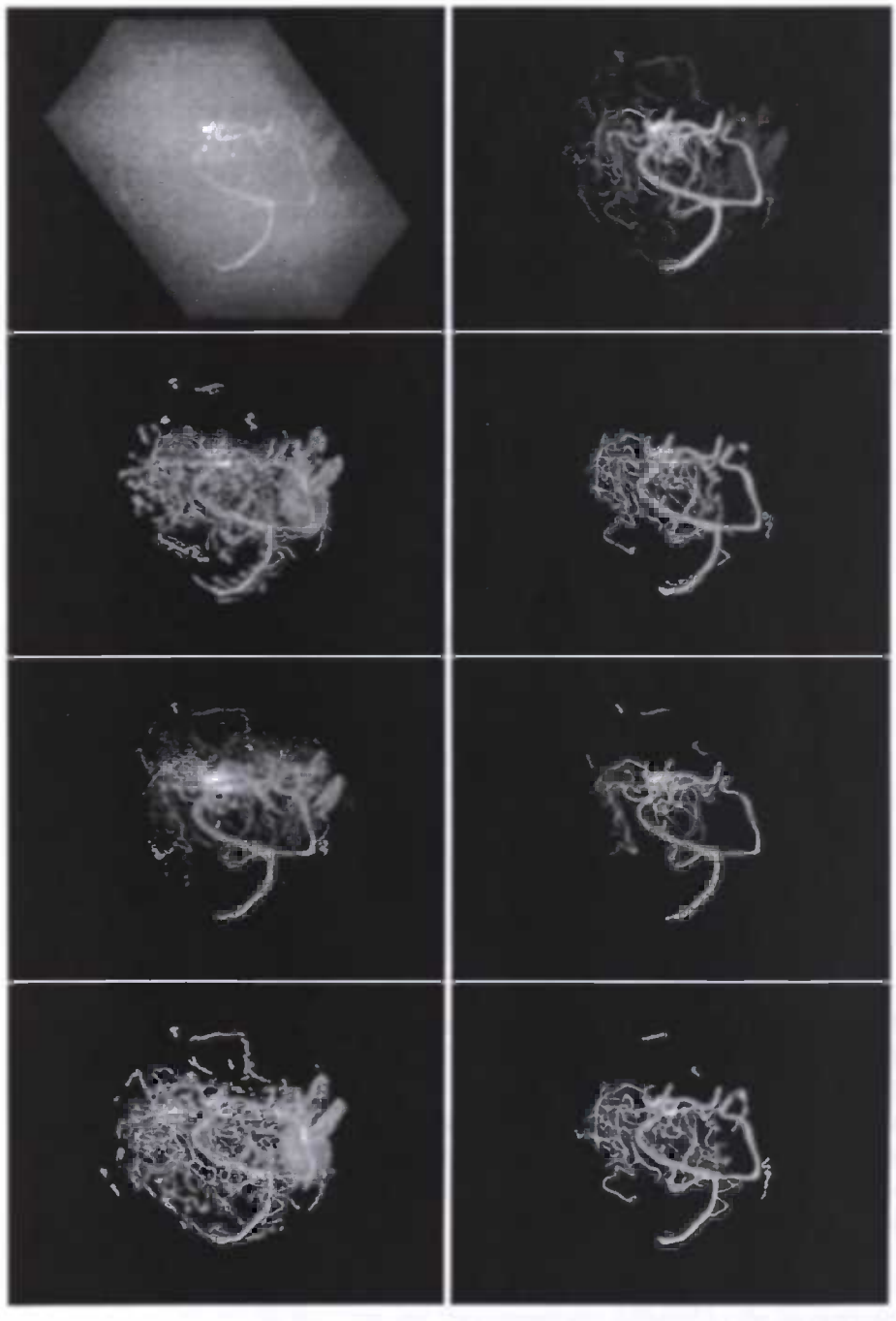


Figure 3.5: The importance of attribute filtering. In the top row left the original **MRA_2**, on the right another filtered version (λ=2). Then, from the second row till the last row segmentations of the original and both filtered versions. From top to bottom respectively moving cube RATS (N=3, η=24), normal RATS (*octreelevels*=3, η=32) and recursive Gaussian (σ=3, η=10)

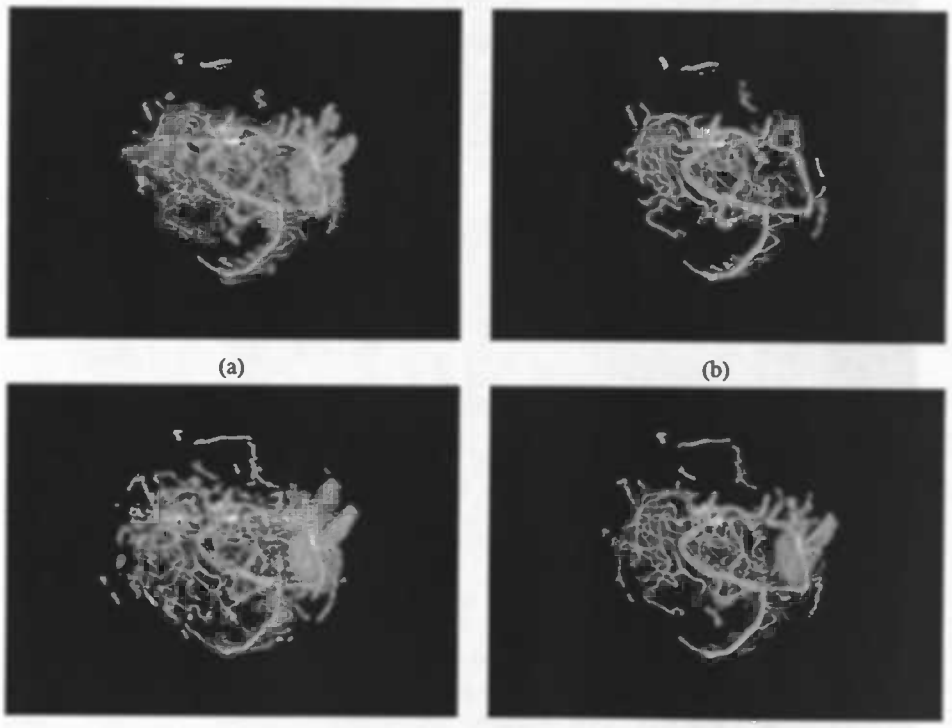


Figure 3.6: When segmenting filtered images η can be lowered. (a) moving cube segmentation of **MRA2**, $\eta = 24$, $N = 3$; (b) moving cube segmentation of filtered ($\lambda_f = 2$) version of **MRA2**, $\eta = 10$,$N = 3$; (c) recursive Gaussian moving cube segmentation of **MRA2**, $\eta = 10$, $\sigma = 2$;(d) recursive Gaussian moving cube segmentation of filtered ($\lambda_f = 2$) version of **MRA2**, $\eta = 4$, $\sigma = 2$; Despite the lower value for η used for segmenting the filtered volumes, the resulting segmentations contain less noise.

3.5.2 Moving Cube Rats

In this section a new approach to RATS is considered. One of the drawbacks of the current method is the recursive division of the volume in smaller, non-overlapping volumes. The reason this is done, is to make the subvolumes in which an edge is sought sufficiently small. If no edge is found in a subvolume, its parent (sub-) volume is analyzed.

In three dimensions the number of leaves in the octree increases very rapidly with the number of levels in the tree, and hence uses a large amount of memory. Another disadvantage is that for all voxels in a subregion the same voxels are used for computation of the statistic for the center voxel.

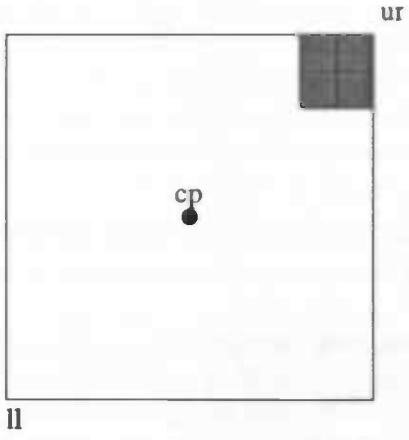


Figure 3.7: Drawback of RATS: a few pixels near ur affect the threshold of *cp*, which in turn has consequences for computation of the threshold of the whole area.

To show that this can be a drawback we give an example for the 2D case: Consider a square subregion in which almost all pixels are background, except for a 2 $\times$ 2 block of foreground pixels in the upper right corner (ur) (figure 3.7). In this upper right region, the Sobel value will be higher. However, this influences the computation of the threshold in the "edgy" region exactly as much as in the non-edgy area in the lower left corner (ll). All pixels in the square subregion (a leaf of the quadtree) contribute equally to the threshold which is assigned to the center point (*cp*) of this subregion. Since interpolation is performed, the thresholds of the pixels ll and ur will not be the same, but they are equally affected by the foreground pixels near ur. It would make more sense when the computation of the threshold at ur is considerably more strongly affected.

For this reason, another, new approach is applied to RATS. Instead of recursively dividing the volume in sub-volumes, we will use a kind of *moving window*-approach.

In 2D this means that a window W is moved along every pixel in dataset, and that at each pixel p the pixels within the window are used to compute the statistic for that pixel p (figure 3.8). In 3D the idea is the same, only a cube is used instead of a window. Now every voxel has his own "private" cube of surrounding voxels to compute the threshold as accurately as possible.

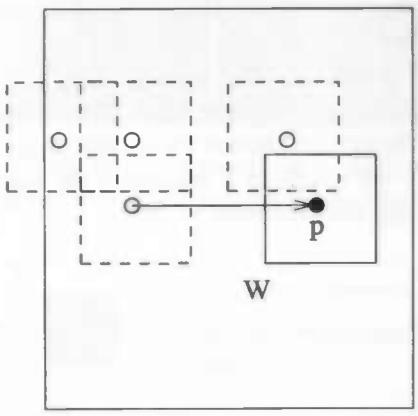


Figure 3.8: Moving window approach: all voxels are processed and the window W is moved along with the current pixel p to compute an optimal threshold

A direct drawback is that more thresholds have to be computed. Advantages are that every single voxel has one (according to RATS) optimal threshold in the case a threshold *can* be determined, no more interpolation is required and we do not have to take care of management of data structures like octrees any more.

Recall that the grey level at position (x, y, z) is given by $p(x, y, z)$. Furthermore, the edge strength is given by $e(x, y, z)$. Now

$$w(x,y,z) = \max \begin{cases} e(x,y,z) & \text{if } e(x,y,z) > \lambda_n \cdot \eta \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

Formally, the threshold surface computed by a moving window cube of RATS can be written as

$$T(x,y,z) = \frac{\sum_{i=x-h}^{x+h} \sum_{j=y-h}^{y+h} \sum_{k=z-h}^{z+h} w(i,j,k)p(i,j,k)}{\sum_{i=x-h}^{x+h} \sum_{j=y-h}^{y+h} \sum_{k=z-h}^{z+h} w(i,j,k)} \tag{3.20}$$

which can be written as the ratio of two convolutions

$$T_h(x,y,z) = \frac{(\Pi_h * (w \cdot p))(x,y,z)}{(\Pi_h * w)(x,y,z)} \tag{3.21}$$

in which $*$ denotes convolution and $\Pi_h(x, y, z)$ is given by

$$\Pi_h(x, y, z) = \begin{cases} 1 & \text{if } |x| \leq h, |y| \leq h, \text{and } |z| \leq h \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

The input image is convolved with a convolution kernel: the "cube". Every element in the convolution kernel array corresponds to a voxel in the cube. At each convolution step, the grey values of each voxel corresponding to a kernel array element are read, then scaled by their corresponding kernel element. The resulting values are all summed together into a single value, and the threshold determined by RATS is the ratio of two convolutions.

Now that we showed that RATS uses convolution, we use the knowledge that one convolution with a separable $N \times N \times N$ filter can be replaced by three convolutions with an $N \times 1 \times 1$ filter in the x-, y- and z-direction. Instead of using N^3 multiplications we now use $3 \cdot N$ multiplications. This can even be reduced to only 6 multiplications. Initially, the filter used is a simple flat filter, all values have the same contribution to the weights and sums. Below we will show that also other separable filters than a flat one can be used. Using this property of separable filters, the results can be obtained in a much faster way than when placing a cube around every single voxel.

Also worth mentioning is what has to be done when no threshold can be determined. In the octree-case we consulted the parent when the edge criterion was too low for an edge to be present. Since this is not possible in the moving cube technique, we decided to try some other strategies.

Initially, a good strategy seemed to be to keep track of the last processed voxel and use this information. When at the current voxel c no threshold can be determined, we go back one voxel p (in the direction the voxels are processed) and look whether this is a foreground voxel or a background voxel. The reason for this is that when p is a foreground voxel and no threshold can be determined for c, the chance is high that c also is a foreground voxel. The problem with this strategy however is that when a wrong decision is made, the error propagates, even visually (figure 3.9). When for a long row of voxels the edge criterion is too low, all those voxels will inherit the color of the first voxel in the row. This means that after the algorithm "leaves" a vessel, a row of voxels of which no threshold can be determined will all get the foreground color. As a result the segmentation contains long one-voxel-thick structures sticking out of the vessels in the volume.

Another strategy is deciding to just give a voxel background color when no threshold can be determined. Recall that this occurs when the edge criterion C is too low. This means that the sum of edge values is low, so there are probably not many vessels in the cube around the current pixel. For this, it is a well considered decision to make it a background voxel. This strategy is better, see figure 3.9. However, we

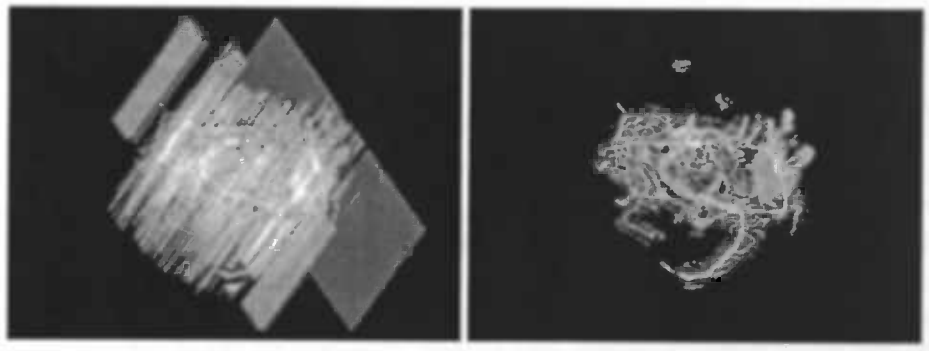


Figure 3.9: Left: One-voxel-thick structures as a result of a propagating error. Right: The same segmentation, with voxels where no threshold can be determined set to background

have to be careful in the case of small cubes and thick vessels. Regarding cube size and vessel width we distinguish two situations:

1. $N < width$: When the cube is completely inside the vessel, the sum of the edge values will be too low to determine a threshold. Setting the voxel value to background causes holes within vessels, which is undesirable. These holes can be removed by a connected component analysis. More on this in section 3.5.4.

2. $N > width$: An edge *must* be present when the cube center is in the vessel, so the criterion C is unlikely to be too low in the neighbourhood of vessels.

Instead of making the decision whether a pixel becomes foreground or background, we also can assign a default threshold to the pixel itself in case RATS is not able to assign a threshold. Here are several options possible as well, among which using a commandline parameter for the default threshold, or using the RATS-threshold of the whole image as a default threshold. In the last case the threshold still depends on the image itself.

An idea which might be worth implementing in the future, is a variation on giving a voxel the color of the last processed voxel. Instead of using the knowledge whether the last voxel was fore- or background, we use the threshold applied to the last processed voxel, in the case no threshold can be determined. The chance for voxel rows to appear is considerably smaller, because thresholds determined when "leaving" the vessels will not quickly cause background pixels becoming white.

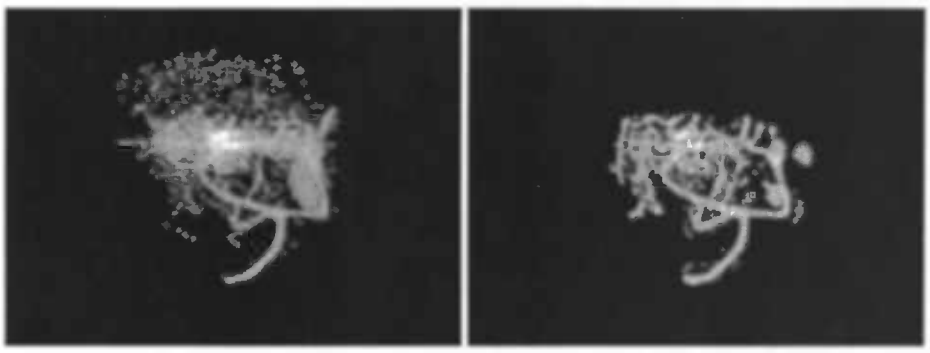


Figure 3.10: Comparison between the segmentations of standard RATS (left) and *moving cube* RATS (right), applied to **MRA1**: Note that the latter contains considerably less noise.

Results

The *moving cube*-version of RATS was tested on the same datasets as the original 3D version, and the results are very promising. Especially the results on **MRA1** are excellent compared to the original RATS. **MRA1** is a small, noisy dataset in which it is hard to distinguish noise from small details. *Moving cube*-RATS does not seem to have many problems with this dataset (figure 3.10).

Two parameters can be set, the noise level η and the cube size N. When the cube size parameter is N, the threshold is locally computed in a cube of size $(2N+1)^3$. Changing η has the same effect as before. The larger the parameter, the more voxels are considered to be noise and more detail is omitted. However, since *moving cube*-RATS is far more accurate, results obtained with the same noise parameter by the two different methods are hardly comparable. For example, a segmentation by *moving cube*-RATS with noise parameter 30 still contains considerably more detail than a standard segmentation with noise parameter 20.

Varying the N has other effects. When N decreases, the region in which the threshold is determined decreases as well. When this region is sufficiently small, the threshold is computed best since no other edges in the neighbourhood influence the computation at the current voxel. The ideal cube size would be the size at which the thickest vessel part of the image still fits in the cube. Regarding the N-parameter, another tradeoff has to be made. A too small cube size causes holes within thick structures in the image, since the edge of the thick structure cannot be covered by the small cube. A too large cube size, on the other hand, will cause the outer region of the cube to contain structures which can influence the threshold computation, which is not desirable. The latter effect is stronger when boundaries of vessels are soft, and hence a slight variation in the threshold can cause a relatively large change in the segmentation of such vessels.

At this moment we advise to keep the N-parameter small, so thin structures are properly segmented as well. Holes within vessels which can occur when using a small cube, can be processed afterwards in a connected component analysis. This subject is treated later in this paper.

3.5.3 Moving Cube - Recursive Gaussian Filter

As mentioned before, it could be interesting to use a non-flat filter for the convolution. A well-known and often applied example of a non-flat filter is the Gaussian filter. As the name states, the Gaussian filter is derived from the same basic equations used to derive the Gaussian distribution $G(x, y, z)$ which is given by:

$$G(x, y, z) = e^{-\frac{x^2+y^2+z^2}{2\sigma^2}} \tag{3.23}$$

where x, y, z are the image co-ordinates and σ is a standard deviation of the associated probability distribution. When needed, the function can be multiplied by a normalizing factor.

The most important property of a Gaussian response filter is that the strongest response occurs around the center, and the response strength decreases when moving away from that center. When plotting the response as function of the width (standard deviation) σ in a graph, this yields a particular bell shape ("Gaussian curve"). When convolving an image with a Gaussian filter, the response around the origin of the filter is the strongest, so the farther away from the origin, the smaller the contribution to the total response. Applied to RATS this means that when an edge is found at a position on a relatively large distance from a voxel, this will hardly contribute to the threshold computation for that voxel. Another property of the Gaussian filter is the separability (for an explanation see 3.5.2).

We have used a recursive implementation of the Gaussian filter. This implementation yields an infinite impulse response filter that is relatively computationally cheap, independent of the value of σ. Moreover, the implementation is faster than many other implementations of a Gaussian filter. For more details on the implementation we refer to the article of Young and Van Vliet [34].

The Gaussian RATS was tested on the same datasets and immediately a remarkable difference can be noticed (figure 3.11): where the segmentations of standard RATS and moving cube still contain noisy regions, the Gaussian segmentations contain far less of this noise, *even* when η is lowered to a level at which the non-Gaussian versions would produce mainly-noise-segmentations.

Because the segmentation by a Gaussian filter standard contains less noise, varying η has less effect than varying it in the normal filter. Another parameter that can be varied is σ. The results which are obtained so far make us believe that a large value

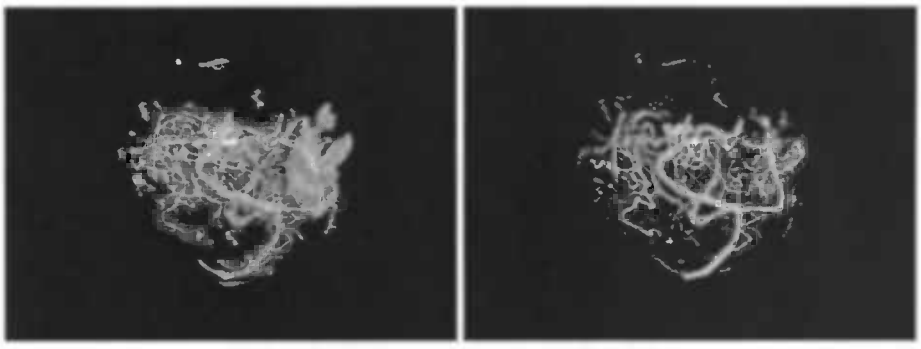


Figure 3.11: Comparison between the segmentations of moving cube RATS (left) and the recursive Gaussian implementation of moving cube RATS (right) with σ = 8, applied to **MRA1**. In both segmentations η is set to 25. Note that Gaussian results may vary strong with σ.

σ makes the segmentations less accurate. Also a too low value for σ should not be taken, since too much noise is taken then in the image. In figure 3.12 we show the effect of different choices for σ.

In order to find the optimum value for σ we did some experiments. We made a phantom, by segmenting **MRA2** with moving cube rats, with parameters η = 30 and N = 3. We performed a binary AND on the result of this segmentation with **MRA2**, to complete the phantom. Note that the phantom itself is also it's ideal segmentation. Taking this into account, we have a reference segmentation for comparison with other segmentations.

We reduced the image quality of this phantom by adding different levels of Gaussian distributed noise to the image. By "different levels" we mean that random numbers drawn from Gaussian distributions with different standard deviations are added. Thus Gaussian noise of level 16 means that to each pixel in the volume a random number drawn from a Gaussian distribution with standard deviation 16 is added.

The generated phantom images are now ready to be segmented. To determine what the optimum value for σ is, we segmented the phantoms with several values for η and σ. After this, we compared them to the reference segmentation (see above) and computed a measure for quality of segmentation: the *volumetric overlap*:$(S \cap R)/(S \cup R)$ (also see section 3.3).

The results of segmenting **MRA2** are shown in table 3.2. We can see that in general, the best segmentations are obtained for $\sigma \leq 4$. Only for **MRA2_48**, segmented with $\eta = 8$, gives the largest σ the best result, although it is the best of the worst. Besides, it is hardly better than other values for σ.

Table 3.3 shows the results of the same tests performed on **MRA1**. The only dif-

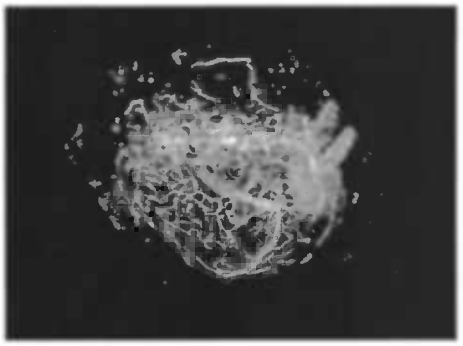
MRA2_8			
σ	η		
-	8	24	48
.75	0.72	0.65	**0.35**
1	**0.77**	**0.67**	0.33
4	0.75	0.56	0.32
8	**0.73**	0.37	0.32
16	**0.73**	0.32	0.32
32	**0.73**	0.32	0.32

MRA2_16			
σ	η		
-	8	24	48
.75	0.60	0.62	**0.35**
1	0.70	**0.65**	0.34
4	**0.73**	0.54	0.33
8	0.72	0.37	0.33
16	0.71	0.33	0.33
32	0.71	0.33	0.33

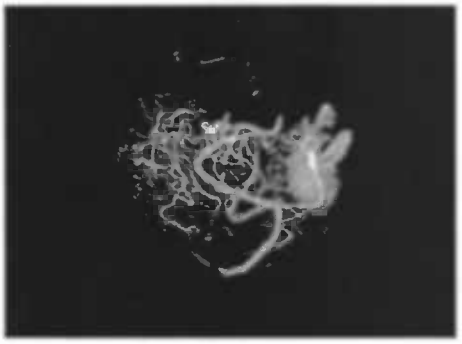
MRA2_24			
σ	η		
-	8	24	48
.75	0.25	0.57	**0.36**
1	0.49	**0.61**	0.34
4	**0.56**	0.51	0.33
8	0.54	0.37	0.33
16	0.51	0.33	0.33
32	0.45	0.33	0.33

MRA2_48			
σ	η		
-	8	24	48
.75	0.01	0.33	**0.25**
1	0.01	**0.34**	0.24
4	0.02	0.31	0.24
8	0.02	0.25	0.24
16	0.02	0.24	0.24
32	**0.02**	0.24	0.24

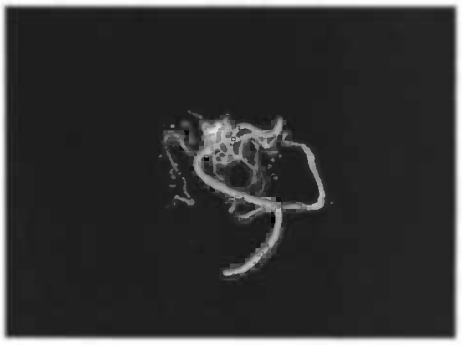
Table 3.2: Results of segmenting **MRA2**.For each η, the highest measured value is printed in bold, so one can see quickly which value for σ gives the best segmentation under these circumstances (and according to the *volumetric overlap* measure). The pictures contain segmentations with added Gaussian noise of respectively 8, 16, 24 and 48.



(a)



(b)



(c)

Figure 3.12: The effect of changing σ in Gaussian filtering: three segmentations with (a) $\sigma = 1$ (b) $\sigma = 8$ (c) $\sigma = 40$

ference is that we added somewhat less noise. The grey scale range of **MRA1** is significantly smaller (more than a factor 5) than the grey scale range of **MRA2** (see table 3.1), so added Gaussian noise has in the first volume relatively a stronger effect. **MRA1_0** has not been preprocessed by adding noise.

In this case, a small value for σ works best, especially in the volumes with a little noise. Note that smaller values for σ give better results (compared to segmentations

MRA1_0			
σ	η		
-	2	4	8
.5	0.37	0.29	0.17
.75	0.92	0.83	0.41
1	**0.92**	**0.86**	**0.41**
2	0.87	0.82	0.38
4	0.79	0.69	0.31

MRA1_4			
σ	η		
-	2	4	8
.5	0.32	0.26	0.16
.75	0.66	0.69	0.38
1	0.68	**0.72**	**0.38**
2	**0.68**	0.70	0.35
4	0.68	0.60	0.30

MRA1_8			
σ	η		
-	2	4	8
.5	0.13	0.54	0.16
.75	0.08	0.55	**0.36**
1	0.08	**0.57**	0.36
2	0.10	0.56	0.33
4	**0.14**	0.51	0.29

MRA1_16			
σ	η		
-	2	4	8
.5	**0.03**	0.08	0.16
.75	0.02	0.06	0.31
1	0.02	0.06	**0.32**
2	0.02	0.07	0.29
4	0.02	**0.09**	0.26

Table 3.3: Results of segmenting **MRA1**. Again, for each η, the highest measured value is printed in bold, so one can see quickly which value for σ gives the best segmentation under these circumstances (and according to the *volumetric overlap* measure). The first picture contains no extra added noise; the other pictures contain segmentations with respectively added Gaussian noise of 4, 8 and 16.

of **MRA2**) because the size of the vessels is about twice as small.

3.5.4 Refinement of the RATS-segmentation

Besides RATS, another segmentation algorithm has been implemented. We made a choice out of one of the four algorithms we described in more detail in section 3.2.

Since the approach of Young et al. [35] has difficulties with border selection near bifurcations we did not implement this one. We decided to choose between the method proposed by Masutani [16] or the method proposed by Stefancik and Sonka [25] . An advantage of either of them is that they both consist of a thresholding step. Especially the segmentation step of Stefancik and Sonka is very straightforward, and it is possible to test whether we can use RATS as a segmentation step within one of the other algorithms.

Taking into account that we will use RATS here as part of the segmentation algorithm, we can consider RATS itself as the core segmentation algorithm, while the rest can be considered as refinement of the original segmentation.

We decided to implement the algorithm based on the paper of Stefancik and Sonka [25], for the reasons mentioned above. It turned out that the biggest part of the algorithm described in that paper is not needed for our actual goal, the segmentation of the blood vessels. Stefancik and Sonka do not only segment the vascular system from the rest of the image, their target is also to generate a vascular tree structure, separate the arteries from the veins, and label them. Since this is more than we need we only use the part for segmenting the vessels from the background.

We start by searching for an appropriate seed voxel in the segmented dataset. The seed voxel is the starting point from where the region growing starts and has to be located within a vessel structure. We have implemented a quite straightforward method to identify a seed voxel. If a voxel, and all the 26 surrounding voxels are white, we consider it as a part of a vascular structure. This test is based on the knowledge that noise usually consists of "speckles" and very small structures. The chance that a noise voxel is chosen as a seed voxel for region growing is quite small. A drawback of this method is that seeds also can be found in large structures, which are neither vascular structures nor noise structures, for example organs. We also have to be careful when we use a dataset which is segmented by RATS3D, because block-like structures can occur in datasets when the noise level η is too low. However, this method of determining a seed voxel works reasonably well. In the implementation, multiple seed voxels are determined before the region growing starts. We will explain this later.

In section 3.4.2 we give a short description of conditional bounded space operations. In this kind of growing two bounded space operations are used, simple growing (bounded space dilation) and growth front smoothing (bounded space closing). Growth front smoothing is only used for controlling the shape of the growth front, which for example is important for detecting bifurcations in order to build a tree which consists of distinct vessel parts. Because we do not need these individual grow-clusters while segmenting the volume, we only use the bounded space dilation, although other bounded space operations have been built in for future use.

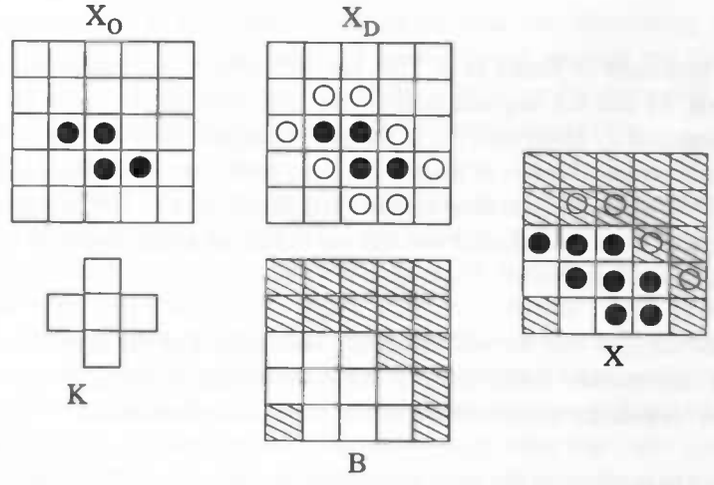


Figure 3.13: Conditional dilation: The original image X_O is dilated with structuring element K and yields X_D. Then X_D is *AND*-ed with the binary mask B. The result is the conditional dilation X.

For the bounded space dilation (figure 3.13) we need a number of volumes. X is the volume which will eventually contain the grown version of the original volume. B is the binary mask, for which the original (binary) volume is used. An AND-operation is performed with the binary mask and the X after dilation to make sure no new voxels are added. We also need a structuring element K, the kernel for performing the dilation. So far we have implemented 8 different structuring elements of different sizes and shapes, to see what effect the different sizes and shapes of the structuring elements have on the results (in case they *have* effect). Figure 3.14 shows the 8 implemented structuring elements, we took them from [24].

The approach we implemented consists of the following steps:

1. Read the binary input volume and binary mask B
2. Define one or more seed voxels

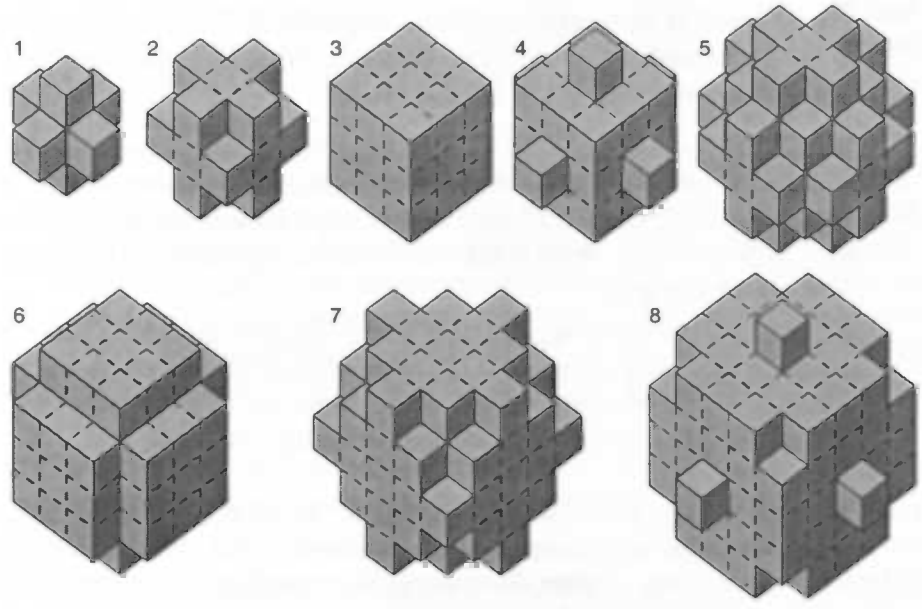


Figure 3.14: Eight different structuring elements used for the dilation

3. Perform the segmentation using the seed voxels as starting points for subsequent bounded space dilations until a stop criterion has been reached. In our case we stop when no more voxels are added in X when performing another bounded space dilation.

4. Search for possible other seed voxels which have not been processed yet and continue growing till the stop criterion has been reached and no more seed voxels are found

To speed up the performance of the implementation a few changes have been made which may be worth mentioning. We will discuss them and the reasons we have chosen for them.

- We want to keep the number of iterations (bounded space dilations) low, since this is the slowest part of the program. One way to do this is defining multiple seed voxels before starting the sequence of bounded space dilations, since for each dilation the whole dataset is processed. The best option seems to locate seed voxels not too close to each other. Here fore we divide the volume in 8 subvolumes and look for a seed voxel in each of those parts.

- The bounded space dilation consists of a 'conventional' dilation and a binary AND-operation. Both operations require traversing the complete dataset, so to save time we perform both operations at the same moment. At the moment

a voxel should be made white by dilation we look to the corresponding voxel in the binary mask B. If the corresponding voxel in B is black, the voxel which should be changed in X remains black.

- When a white voxel has been found, the origin of the structuring element is virtually placed on it and voxels below the white voxels of the structuring element are made white. It is possible however, that a part of the structuring element is outside X, so we check whether a voxel is within the bounds of X before we change it.

  Because this test is not needed most of the time (the structuring element is usually very small compared to the input volume) we only test this at the boundary of X. Hence the main part of the volume can be processed without testing boundary conditions. In practice this hardly saves time.

There are some drawbacks to our implementation. As mentioned before, noise voxel groups which are large enough will be considered as seed voxels and appear in the final segmentation. Another disadvantage is the need for a good mask. The quality of the segmentation made by region growing is strongly dependent on the quality of the binary mask. It also could occur that there are vessels which are not connected to parts of the image where a seed point is positioned, and too thin to contain a seed point itself. In this situation this thin structure will not be part of the final result. This problem can be reduced by using other sizes and shapes for choosing the seed voxel.

Connected component analysis using Tarjan's union-find algorithm

The dilation algorithm described above can be improved. Apart from the problems with large non-vessel objects which can contain seed voxels, the speed of the dilation process can also be a handicap for the algorithm, especially when using small structuring elements for the dilations.

If we take a closer look on the dilation algorithm, we discover that the result consists of:

- all connected components in the image which are large enough to contain a seed voxel.
- all connected components which are reached during the dilation when the distance to the dilated volume is small enough for the structuring element to reach such a component

When using a structuring element in which the maximal distance to the origin does not exceed 1 (like SE1, SE2 and SE3 in figure 3.14) the results only contain the

first kind of connected components. Hence we can conclude that it is sufficient to extract those connected components which are large enough to contain a seed voxel. In fact, this is by definition an opening by reconstruction of the input image by the structuring element as a seed voxel.

In [33] a description of Tarjan's Union-Find algorithm in the context of morphological filtering is given. The original Union Find algorithm, which provides a general method for keeping track of disjoint sets, can be found in [26]. Connected components are by definition disjoint sets, so this algorithm is useful for our purpose to divide an image in connected components. This method is considerably faster than dilating, since the dataset has to be traversed only two times to detect and label all connected components. We give a brief description of the algorithm used [4]:

1. We store connected components in tree-like structures. Voxels x and y are part of the same connected component if and only if they are nodes of the same tree.
2. We define an array `parents`, of the same size as the image so we can keep track of the parent of each voxel. The parent of a root-voxel is flagged, by let the root voxel point to `MAXLONG`.
3. Now all voxels will be processed. For each foreground voxel, we look to all neighbours which have been processed *before* the current voxel `p`. If such a neighbour `n` is also a foreground voxel, the root of `n` is set as the parent of `p`. If another neighbour foreground voxel `n2` is found and `root(n2)` $\neq$ `root(n)`, then `root(n)` becomes the parent of `root(n2)`.
4. At this stage, all connected components are found and labeled. We only have to make a new pass through the volume to ensure the parent of every voxel is indeed a root voxel, in order to directly determine which connected component a voxel is part of.

After labeling all connected components we search for seed voxels. If a seed voxel is found, all voxels belonging to the same connected component are made foreground voxels. This process is continued until no more seed voxels are found. As a result we get an image with all connected components from the input image which are large enough to contain a seed voxel, and this is exactly what we wanted.

A drawback of the seed voxel method, which also occurred when using the dilation instead of the connected component labeling, is that when dealing with ghost objects, those object are preserved when they are large enough to contain a seed voxel. Instead of changing the size or shape of a seed voxel, we use the division of the image in connected components for two other goals to refine the segmentation:

1. Removing holes within the vessels

2. Removing ghost objects

Removing holes within the vessels is quite simple after an analysis of all connected components, when a few (logical) assumptions are made. The first assumption is that the biggest connected component with the background color is the background of the image itself, the second assumption is that it is impossible for a vessel system to contain holes. Now the algorithm is very straightforward: if a connected component with background color is found which is not the background itself, change the color of the connected component to the foreground color. In this way all holes are filled properly, and since vessels cannot contain any holes this is exactly what we want. Another criterion which can be used to look if a connected component with background color is a hole is checking whether the component touches the image border. If it does, it is the background, if it does not, it is indeed a hole in a vessel. Using this method, also holes caused by flow-void (non-uniform distribution of contrast agent) will be found. Although they belong to the original MRA (and hence, the "correct" segmentation) it will found useful when they are removed in the segmentation.

Removing the ghost objects is done using the result of the Sobel-filter applied to the image. Ghost objects, as well as noise pixels, have a very small average Sobel value per pixel, since real edges are hardly detected in the neighbourhood. During the connected component analysis we keep track of the average Sobel value of each connected component. Once again, the algorithm is very basic: when a connected (foreground) component is found which has an average Sobel value smaller than a certain value s, the color is changed to the background color. The Sobel value is taken over edge voxels as well as non-edge voxels. In practice a value of 1 for s works fairly well. However, different values for s might be tried. Moreover, other criteria than average Sobel value of voxels can be used to detect and remove ghost objects and other unwanted connected components.

```
FOR ALL VOXELS v
  P:=GetConnectedComponent(v)
  IF P.color = white
    IF P.sobelavg < 1 [
      v := black
    ] // removing ghostobjects
  ELSEIF P.color = black [
    IF P.numberOfVoxels < 0.5 * TotalNumberOfVoxels [
      v := white // filling holes
    ]
  ]
```

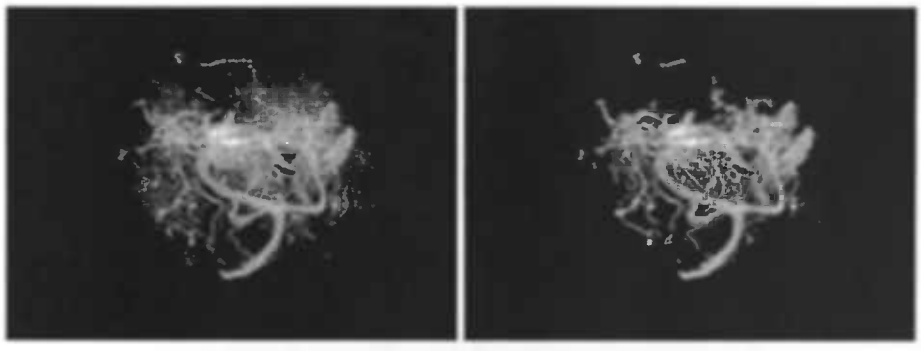


Figure 3.15: A segmented volume before (left) and after (right) the holes are filled and ghost objects are removed.

Results

The connected component analysis provides a useful contribution to the segmentation, which we can see in figure 3.15. On the left we see **MRA2** segmented by normal RATS, with 4 levels in the octree and $\eta = 20$. On the right we see the same volume after the connected component analysis is done, the holes have been filled and the ghost objects have been removed. It is clearly visible that (especially on the outside of the volume) several small voxel groups and single voxels have been removed.

We segmented **MRA2** using all three implemented variants of RATS. For each variant we used at least two different values for η. Using these segmentations we performed some tests. We filled the holes and removed the ghost objects and counted the number of voxels added (in case of filling holes) and the number of voxels removed (in case of removing what is identified as ghost objects). In order to make the effect of the connected component analysis visible, we constructed a "difference image" from the volume, by applying an XOR-operation to the original, and the version in which the ghost objects have been removed and the holes have been filled.

The connected component analysis is not an improvement for every kind of segmentation. In the recursive segmentations mainly vessel-like structures were removed (figure 3.16), which indicates that there is not much "ghost" noise left to remove. In the difference images 3.17 we see other examples of what voxels were removed.

We segmented different data volumes using the bounded space dilation. The binary mask used was the result of segmenting the original data volume with RATS. The results of the dilation algorithm look good. The fact that single voxels or small voxel groups are left out yields a clear binary image. This binary image can also be used as a mask over the original (or filtered) image in order to get back the original

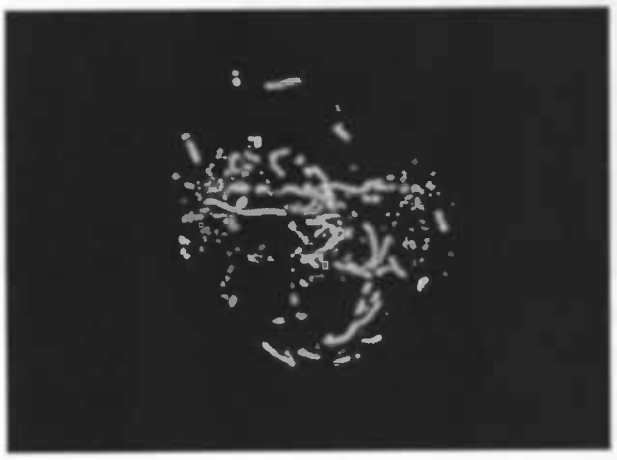


Figure 3.16: Removed ghost objects after a connected component analysis of a recursive Gaussian moving cube segmentation ($\sigma = 2, \eta = 16$)

MRA2			
Method	η	Filled	Eliminated
mc(3)	10	414	93984 (29.44%)
mc(3)	16	145	44300 (34.70%)
mc(3)	24	12	22446 (28,84%)
mc(3)	32	6	15440 (29.22%)
normal(3)	16	125	17854 (17.94%)
normal(3)	24	22	7728 (14.55%)
normal(3)	32	11	4426 (13.31%)
rec(1)	10	138	61325 (57.31%)
rec(1)	16	13	27522 (43.83%)
rec(2)	10	266	9886 (10.99%)
rec(2)	16	57	6292 (10.72%)

Table 3.4: In this table we give the number of pixels added and removed, as well as the segmentation method used. The number between brackets after the segmentation method indicates N in case of moving cube (mc) rats, the number of levels in the octree in case of "normal" RATS and σ in case of recursive Gaussian moving cube rats (rec).

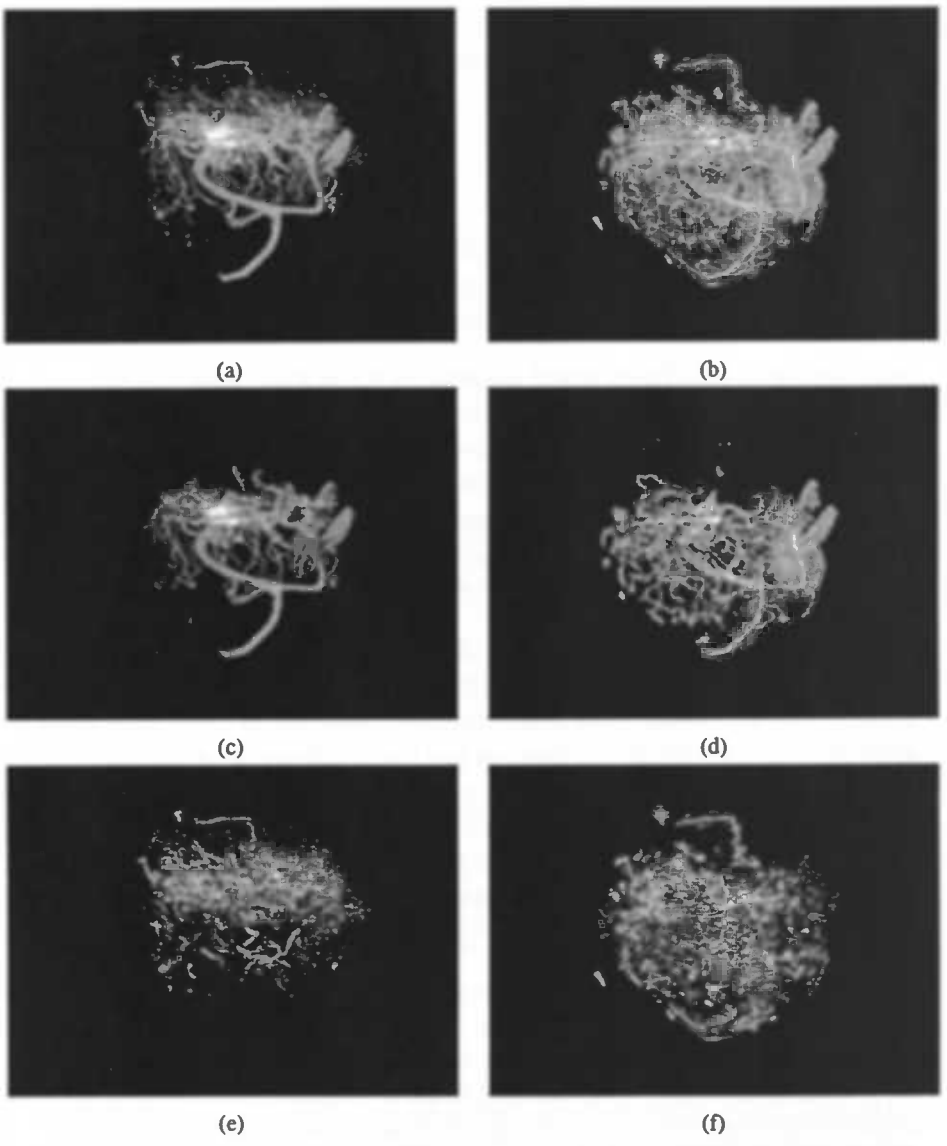


Figure 3.17: Segmentations, the result of removing "ghost voxels" and the removed ghost voxels.. First the original segmentations: (a) **MRA2**, normal segmented, octreelevels=3, $\eta = 16$; (b) **MRA2**, segmented with Moving Cube RATS, $N = 3$, $\eta = 16$; In (c) and (d) the volumes which are the result when all foreground voxels with Sobel value < 1 are removed; In (e) and (f) the difference images: the removed ghost voxels

colors.

One of the drawbacks mentioned above is that very thin vessel ends (with thickness

less than or equal to one voxel) which are not continuous anymore after segmentation are completely left out when sufficiently far from the thicker parts.

A more detailed analysis

Although the quality of segmentations is hard to measure, we performed many tests and segmented many volumes to get a good insight of the qualities, and the weaker points of the different segmentation algorithms.

The segmentation methods tested are Moving Cube Rats, normal RATS and the Recursive Gaussian Moving Cube RATS. Furthermore, the refinement of the segmentation in which Tarjan's Union Find algorithm is used is treated. The segmentation methods are used on the earlier mentioned volumes **MRA1**, **MRA2** and **CTA1**, modified versions of these volumes (by softening these using a Gaussian filter, or adding Gaussian noise) and phantom volume, which can be constructed using original vessel volumes or completely manually.

The segmentations can be judged visually, but this is not always a good measure. Besides, what one thinks is a reasonable segmentation can be considered as an unreasonable segmentation by another person. However, a visual judgement is a good first estimate and gives a more or less rough indication. A segmentation can quickly be considered as "very bad" by taking a quick look at the result. When the result is as good as empty, or it contains all kinds of strange shapes and noise which should not be expected in normal segmentations, this can be judged very quickly as a bad result. It would be a waste of time to compute a (costly) measure for quality of the segmentation, when this can be told by just looking at it.

As mentioned before in section 3.3, there are some quantitative measures for the quality of segmentations in the case an "ideal segmentation" is available as a reference. Since the volumes which we segment are *real* vessel volumes, there is no ground truth available. A way to provide a ground truth is creating a volume ourselves (a phantom volume), which will be segmented by the different methods after it is blurred or noise is added. A reference segmentation is now available because we know exactly what the segmentation should look like, since we created the volume ourselves.

Which things are remarkable?

- When random Gaussian noise is added and using the volumetric overlap as a measure for segmentation quality, the Recursive Gaussian Moving Cube RATS seems to yield the best results. Using this method, the value of η can stay low. In figure 3.18 an example of **MRA2**, distorted with Gaussian noise of level 48, and three segmentations.

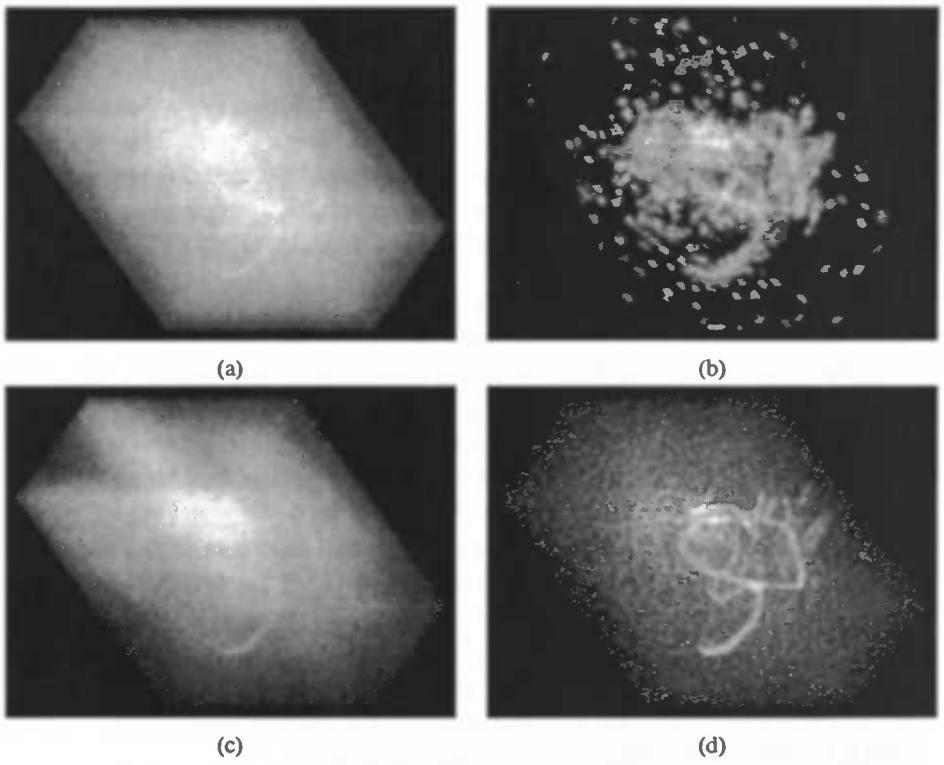


Figure 3.18: (a) **MRA2**, with Gaussian noise of level 48 added; (b) contains the results of Moving Cube segmentation ($\eta = 32$, $N = 3$); (c) shows the result of standard RATS ($\eta = 32$, 3 octree levels) and in (d) the result of the Recursive Gaussian Moving Cube RATS is given. In the last one the desired segmentation is clearly visible between the little noise dots. A volume opening could improve this segmentation a great deal.

- When dealing with really strong noise, standard RATS yields quite low-quality segmentations. For higher values of η, the volumetric overlap values for the moving cube and the recursive Gauss segmentations do not differ much.

- Segmentations by the MC-algorithm, especially those applied to images which contain much noise, contain little cube structures sized N. They occur mainly at the boundary of the volume, which might indicate the boundary conditions are not optimal yet. An example of this can be seen in figure 3.18 (b).

3.5.5 Conclusions

The attribute filtering provides not only a useful contribution to the visualisation of blood vessels, also the segmentation is strongly improved by attribute filtering in advance. Blood vessels are usually filtered by extracting vascular structures from a volume. Because noise is most often not vascular of shape, much noise is removed by the attribute filtering, besides removal of non-vascular objects.

Although RATS is suitable for segmenting noisy images, removing noise before segmenting will even improve the results. We showed that the 3D version of RATS is suitable for segmenting MRA-volumes containing blood vessels. Apart from the 3D extension to standard RATS, also two alternative variants have been implemented: Moving Cube RATS, and Recursive Gaussian Moving Cube RATS. Both methods do not have the shortcoming which local thresholding using the octree with standard RATS has; where the Moving Cube versions have a region of interest (ROI) centred on the current pixel, standard RATS has the limit that the position of the ROI is predetermined by the number of levels in the octree. Of these two Moving Cube variants, Recursive Gaussian Moving Cube RATS, gives the best results. Using the correct parameters for the estimated noise level η in the image, as well as the cube size N (Moving Cube RATS) or σ (Gaussian RATS), is very important as well. When the noise level η can be estimated automatically from the image, the thresholding becomes even more automatic.

A drawback of Moving Cube RATS is that artefacts can appear around the vessel boundary. These have to be removed by post-processing. Segmentations by Recursive Gaussian MC-RATS, on the other hand, hardly need post-processing. Even when segmenting noisy datavolumes, this algorithm works the best. Quality of the results can be even optimized further by post-processing. The better the segmentation quality, the less post-processing is required.

A segmentation can be, if needed, refined further by post-processing. We showed that holes can be filled and ghost objects can be removed by performing a connected component analysis. An alternative for refining the segmentation is an opening by reconstruction, which removes connected components of smaller width than the structuring element.

Chapter 4

Visualization

In this chapter we start exploring some visualization techniques and divide them into categories. After that we will focus on of the Maxtree algorithm and the advantages this will have for fast visualization. Meijster et. al. [17] showed that interactive filtering rates are possible on standard commodity hardware. By using the advantages of our implementation of the Maxtree in which relevant voxels are easily isolated, combined with a suitable visualization method we will show that filtering *and* visualizing of 256^3 datasets is possible at interactive rates. Finally, some additions to the rendering method are proposed to improve results.

4.1 Volume Rendering

Volume visualization is used to create images from scalar and vector datasets defined on a three dimensional grid, i.e., it is the process of projecting a 3D dataset onto a 2D image plane to gain an understanding of the structure contained within the data. The algorithms that create a 2D image of a 3D dataset fall into two categories, *direct volume rendering* (DVR) algorithms and *surface fitting* (SF) algorithms [6].

(a) (b) (c)

Figure 4.1: Different renderings of MRA2 using surface fitting (a), X-ray (b), and maximum intensity projection (c).

4.1.1 Surface fitting

SF algorithms extract iso-surfaces from the dataset using some threshold or other technique. These iso-surfaces are used as object boundaries. SF algorithms try to fit geometric primitives such as triangle meshes to the iso-surfaces.

A straightforward algorithm is the *opaque cube* or *cuberille* algorithm [1]. This algorithm traverses the dataset and generates a little cube where a threshold is found. All the generated cubes are sent to the 3D renderer afterwards or during the threshold finding using the painters algorithm. Another method is the *marching cubes* algorithm [13] which uses a table-based surface fitting procedure. It finds the gradients between the grid points and fits small triangles within each cell through

which a threshold-value surface passes. It can fit up to four triangles to one voxel. SF methods are typically faster than DVR methods because the SF methods only traverse the volume data once to extract the surface. After extracting the surface, well-known rendering techniques and 3D hardware can be used to render the scene.

A disadvantage of SF-methods is that they generate huge numbers of polygons. Simplification algorithms can be used to limit the amount of polygons but this, as noted in the introduction, gives rise to other problems. Simplifying a structure can cause small details on which a diagnose can be based, to disappear. Besides that, SF-methods are only useful when a boundary can be chosen. Another disadvantage of SF-methods is that the iso-surfaces extraction has to start all over again when a new threshold value is chosen. An example of a surface fitting rendering can be seen in figure 4.1(a).

4.1.2 Direct volume rendering

DVR methods do not use an intermediate graphical primitive, but map the data directly onto the screen. DVR methods are especially appropriate for creating images from datasets containing amorphous features such as clouds, fluids, and gases. The drawback of DVR is that the entire dataset has to be traversed each time an image is rendered.

The type of generated images is also different from SF methods. DVR methods do not use an iso-surface and render the whole dataset. Voxel values of all the voxels contributing to a single view plane pixel are combined to generate the image. Typical examples of DVR results are *X-ray rendering* (figure 4.1(b)) and *Maximum Intensity Projection* (figure 4.1(c)).

X-ray rendering sums the contributing voxel values. This has the same effect as integrating along the line from the pixel on the view plane into the dataset. The direction of this line depends on the perspective used. The generated images look like X-ray images (hence the name). Maximum Intensity Projection uses the maximum encountered voxel value instead of integrating. DVR methods can be subdivided into three subgroups: Image order, object order and hybrid techniques.

Image order

Image order techniques, also called *ray casting techniques*, traverse the pixels in the view-plane and cast a 'virtual' light ray into the data for each pixel [5, 12]. The ray is sampled at evenly spaced intervals and sample values are computed using interpolation between surrounding data points. The acquired values can then be combined to generate the image. The type of image depends on the manner of

combining these values. Integrating the list of values for example creates an X-ray image, while taking the maximum generates a MIP image.

Object order

Object order techniques traverse the data and calculate the projection and contribution of a data-point to the pixels in the view plane. Splatting [31] is an example. This algorithm creates a "spot", called a *footprint*, on the view plane for every voxel in the dataset. A spot is required because the dataset usually has a lower resolution than the screen. This will cause gaps between two adjacent data points on the screen. Different types of images can be created depending on the blending algorithm that is used to blend the spots onto the view-plane.

Hybrid techniques

Some methods cannot be placed in the categories above. They use techniques of both types. An example is *Shear-warp factorization* [10]. Shear-warp factorization is based on a factorization of the viewing transform into a 3D shear parallel to the slices of the volume, a projection which forms a distorted intermediate image, and a 2D warp to produce the final image. Shear-warp factorization utilizes the fact that scan-lines in volume data and of the intermediate image are always aligned. This allows efficient, synchronized access to data structures that encode coherence in the volume and the image. *Fourier volume rendering* [15, 20, 27] makes use of frequency domain techniques, and is based upon the Fourier slice theorem [18]. This method provides an efficient way to perform X-ray rendering. It supports shading and depth-cueing [27], but no occlusion. Fourier volume rendering starts with a 3D Fourier transform of the volume data. An image for a given viewing direction is made by computing the values of the Fourier transform in a slice plane parallel to the view plane and through the center of the volume, followed by an inverse 2D Fourier transform in the slice plane.

4.2 Extending the Maxtree

In this section we will first look into our implementation of the whole Maxtree process from generating to filtering. Our implementation has some advantages that can be exploited for fast rendering.

4.2.1 Maxtree implementation

For the generation of the Maxtree a special flood algorithm is used. Our implementation is a $3D$ adaption of Urbach's proposal[29]. Urbach's Maxtree nodes only know their parent. In our implementation the node also knows its children and more importantly the voxels that belong to it. These small changes have a huge impact when the filtered data has to be extracted from the Maxtree, as we will see in section 4.3.

Variables used

The Maxtree consists of node structures linked together. The layout of a node C_h^k is as follows:

`attribute` To store the attribute value.

`k` To store the node number k at the current level.

`level` The original grey level h of the voxels in this node.

`currentValue` The grey level after filtering (Currently not used).

`voxelOffset` The location of the voxels belonging to the node in the voxel list.

`numberOfVoxels` The total number of voxels belonging to this node.

`parent` A pointer to the parent node.

`numberOfChildren` The number of children of this node.

`children` A list of pointers to the children of this node.

There are also some variables used in the flooding process:

`_hQueue` is an array of queues so every grey level has its own queue. The queue can be filled with voxel positions. Only voxels belonging to the currently processed peak component can exist in the queue because of the way the flooding algorithm traverses the dataset.

`_numberOfNodes` is an array denoting the number of nodes at each level currently processed. This value is used to give k its correct value.

_nodeAtLevel is an array of nodes with the size of the number of grey levels in the input data. It is used to store the current working branch or peak component of the Maxtree. Since only one peak component is processed at a certain time there can never be more than two nodes of the same grey level processed at the same time.

_nodeList is a queue to store all the nodes so they can be accessed later. This list is only used in the finalization step.

_status is an array to store the status of a voxel. It tells the algorithm if a voxel has not been visited before, is currently in the queue, or has already been processed.

_voxelList is an array in which all the voxels will be stored. The voxels are grouped by node they belong to and are also ordered by grey level.

_voxelOffsetAtLevel is an array the size of the number of levels in the input data. Stored within is the position in `_voxelList` the next voxel with a given value has to be stored.

Initialization

Before the flooding is started the following is done. The status of all the voxels is set to `ST_NotAnalyzed`. The `_voxelOffsetAtLevel` array is filled with a cumulative histogram of the input data. This way the `_voxelList` can be allocated before the flooding because the size is known. The lower and upper bound of voxels with a certain grey level are fixed and traversing the dataset peak component by peak component assures that at a certain grey level only one node is processed at the same time. Voxels belonging to the same node can now be put together and no overhead is needed.

The root node is allocated and stored in `_nodeAtLevel`. The position of the voxel having the minimum grey level is put in `_hQueue` and flood is called with `h` equal to the minimum grey level.

The flooding

The flooding is started with a current grey level `h` and begins with a loop to pop every voxel with value `h` from the corresponding `_hQueue`. This voxel position is added to `_voxelList`. The location within the `_voxelList` is read from `_voxelOffsetAtLevel[h]`. The offset is incremented so the next voxel with the same grey level will not overwrite the current, and the status, stored in

_status of the voxel is changed to ST_Analyzed. Also numberOfVoxels of the node from _nodeAtLevel is incremented.

After this the neighbors of the current voxel are retrieved. The number of neighbors depends on the connectivity. The next loop will process all the neighbors that have ST_NotAnalyzed as status. The neighbor voxel is added to the corresponding _hQueue and its status is set to ST_InTheQueue. If no working node exists at the value of the voxel, a new node is created and stored into _nodeAtLeveland pushed into the node queue _nodeList. Since the flood handles one peak component at a time and every found neighbor belongs to the same peak component, whether it be in the root or in a leaf, only one branch of the tree has to be remembered. The queue _nodeList is used later to add the children to a node. During the flooding it is not known how many children a node will get. To make the algorithm memory efficient this will be calculated afterwards.

After this the value of the neighboring voxel is compared to the current value h. If the neighboring voxel has a higher value, flood is called with this new value until it returns a value equal to h again. The return value of the flood is the maximum value above which the current peak component is completely analyzed.

This is repeated until _hQueue is empty at level h. Now flood will look for the working node with the maximum value in _nodeAtLevel lower than h. This will be used for the return value and for assigning a parent to the current node. The return value of the flood is the maximum value above which the current peak component is completely analyzed. The node at that level is automatically the parent of the current node because there is only one branch to work with. If a node should be between them it would have been found while processing the neighboring voxels.

In this way the parent is assigned to the node. After this, level and currentValue are set to h, k is read from _numberOfNodes, _numberOfNodes at the current level is increased, and the node is removed from _nodeAtLevel because it is finished and new voxels found at that level belong to another node.

Finalization

At this point the nodes are stored in _nodeList and know their parent and the voxels solely belonging to it. The node list is traversed three times:

1. Count the number of children every node has.
2. Allocate a list for every node to store the pointer to these children.
3. Store a pointer to the children into these lists.

4.2.2 Attributes

To make the Maxtree class versatile, a special abstract attribute class is created. This way extending the Maxtree with a new type of attribute is rather simple. The original algorithm from Salembier calculated the attributes during the flooding process. In our implementation the attribute calculation is done separately from the flooding process. For example, load and save methods for the tree structure are also implemented. Flooding and assigning new attribute types will therefore be seen as separate tasks.

We will focus on this abstract class and on the shape filter attribute discussed in 2.3.2.

Abstract class

Calculating the attributes is done by a method defined in the Maxtree class. It takes an instance of a `MTAttribute` and traverses the tree. For every node an instance of the `MTAttribute` is created using `getNewInstance`. Method `addData` is called with every voxel in the node and `mergeAttribute` is called with every child node. A child node is processed before its parent so the attribute of the child nodes are final. After every voxel and every child is processed the attribute is stored in the node using `getAttribute`.

Shape filter

The shape filter defined in section 2.3.2 is implemented in the following way. Every attribute instance has the following variables:

`_area` This variable is used to store the volume. It being called area might be misleading, but this is a legacy of the 2D version and should be changed in the future.

`_sumX` The summation of all x values of all the voxels is stored in this variable.

`_sumY` Same as for `_sumX` but with y values in stead of x values.

`_sumZ` Same as for `_sumX` but with z values in stead of x values.

`_sumSquares` In this variable the summation of the square distances from the origin to the voxels is stored.

The `addData` and `mergeAttribute` methods maintain the above variable in a straightforward manner. Adding a voxel is done by increasing `_area` and adding the required location info to the corresponding variables. Merging node data is done by adding the child node attribute variables to the current node's attribute variables.

Finally, the finalization of the attribute calculation is done in the `getAttribute` method. The following algorithm is used:

```
inertia = _sumSquares;
inertia -= ((_sumX * _sumX) +
            (_sumY * _sumY) +
            (_sumZ * _sumZ)) / area;
inertia += area / 4.0;
attribute = inertia / pow(area,5.0/3.0);
```

4.2.3 Filtering

As noted in section 2.3.3 there are a number of filter algorithms. Except for the Viterbi strategy, all of them are implemented in the Maxtree program. The filtering itself is done by traversing the tree. Since only the voxels with the same grey value are stored in the node itself, the rest is stored in the child nodes and the whole tree still has to be traversed, even if a branch is rejected. This is because voxels of a rejected node still have to get the grey level of the first surviving ancestor.

The algorithm is recursive and uses the following parameters:

`vqueue` A list of queues. The empty `_hQueue` structure is reused. All voxels except background voxels, are stored in the queue corresponding to the grey level assigned to the specific voxel.

`d` A filter-specific δ value indicating how much levels the grey value of the current branch has dropped. It is only used with the subtractive strategy and is initially set to 0.

`node` The node that is currently under investigation. The first call is with the root node.

`lambda` The filter parameter λ_f.

`parentLevel` The grey value of the last accepted node. This value is filter specific. Initially set to the minimal grey value in the dataset.

`removed` A filter specific boolean indicating if a branch is removed. It is only used with the Min strategy and is initially set to false.

`filterType` A value indicating which strategy is to be used. Currently only Min, Max, Direct, and Subtractive are implemented.

Within the function also some other variables are declared.

`s` In this variable the grey value the voxels in `node` will get is stored. It is initialized on the minimal grey value.

`postRecursion` This boolean is only altered by the Max strategy. This strategy sometimes does recursion before processing the current node. It is initialized on true.

`result` This boolean stores the return value of the filter function. The value is only altered and used by the Max strategy.

After the initialization of the above variables the strategy-specific part of the recursive filter algorithm is selected by a switch statement using the `filterType` parameter. Within these parts, the different parameters will be updated and the grey value the voxels in `node` will get is stored in `s`.

Min The `removed` boolean indicates if an ancestor is removed. If this is the case or if the test of λ_f against the node attribute fails, the current node is removed. The voxels contained in this node have to be drawn with the grey value of the last surviving ancestor, stored in `parentlevel`. If `removed` is not set and the test against λ_f succeeds, the node is drawn with its own grey value and `parentLevel` is updated.

Max This strategy has a slightly different approach. If the test of the attribute against λ_f is successful, the current node is drawn with its own grey value and `parentLevel` is updated. If the test fails `postRecursion` is set to false. The function is called recursively to find out if a descendant still passes the test. The result is stored in `result`. If this boolean is true, meaning there are still descendants drawn, `node` is drawn. Otherwise the voxels will be drawn with the grey value stored in `parentlevel`.

Direct This is a non-pruning strategy. It focuses only on the node at hand. This means the algorithm is very simple. The attribute is tested against λ_f. If the test fails the voxels are drawn with the grey level stored in `parentLevel`. If it succeeds the voxels are drawn with the grey level stored in `node` itself.

Subtractive The last strategy is also a non-pruning one. It also focuses on a single node, just like Direct, but it uses a different approach. Nodes are not accepted ore rejected, but their grey value is decremented with some value δ. If the test of the attribute against λ_f fails, δ is increased with the difference between

the grey level of `node` and its parent. This will effectively lower the grey value of the whole branch starting at the current node. The voxels in the node are drawn with a grey value of the value stored in `node` minus δ, and `parentLevel` is also updated.

At this point `s` is compared to the minimum value. If they are the same, all the voxels in `node` are ignored since they only contribute to the background. Otherwise all voxels belonging to `node` are pushed onto the queue belonging to grey level `s`. If `postRecursion` is set, the filter function is called on all the child nodes of `node` and in the end the `result` boolean is returned.

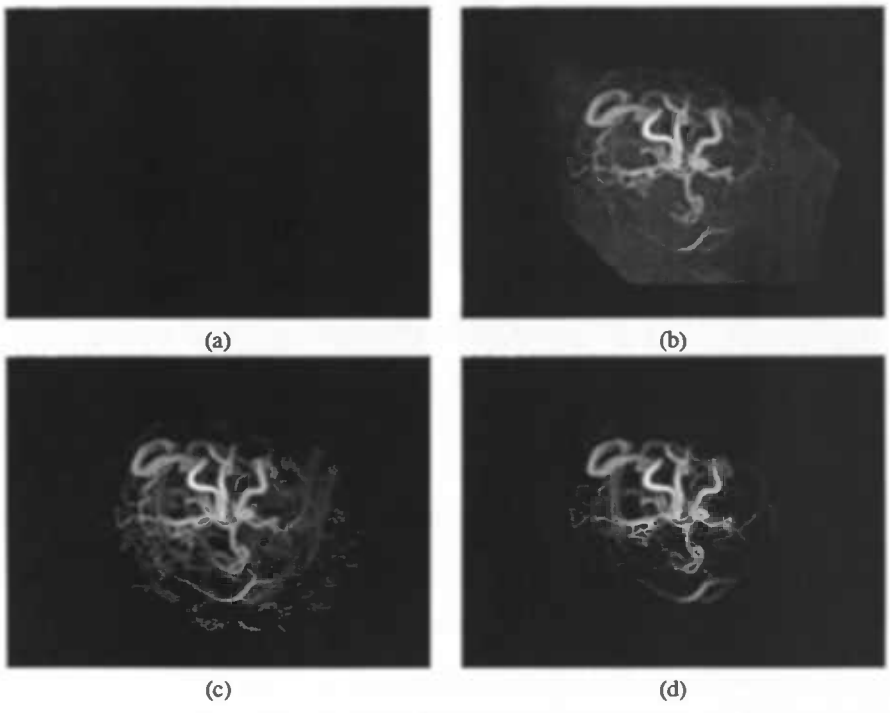


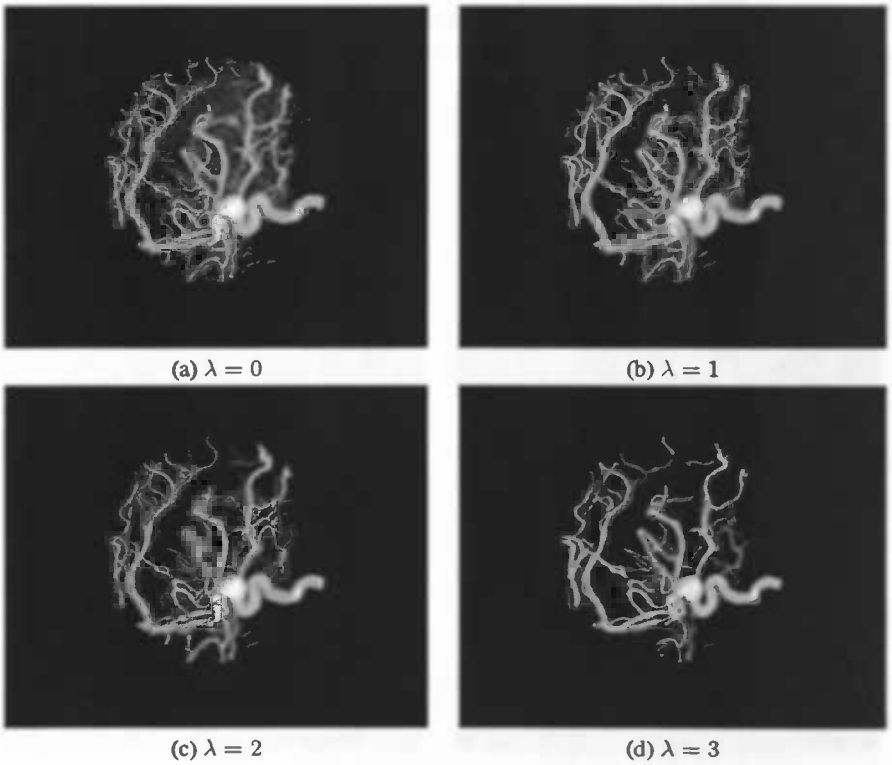
Figure 4.2: MIP rendering of **MRA2** after using a (a) min, (b) max, (c) direct, and (d) subtractive filter strategy. Attribute is the shape attribute with a threshold of $\lambda_f = 2$. Depth queueing (see section 4.6.2) is turned on.

After the filtering step is done `vqueue` is filled with the filtered data. The data can, for example, be written into another volume dataset and be processed by another program. In section 4.3 we will see another conversion that has some nice advantages.

Although pruning strategies are implemented, we will focus mainly on the non-pruning strategies. The background is not elongated so it will have a low attribute value with this shape filter. On the other hand vessel data might have some peak

values consisting of just a few voxels. These will also have a low attribute value. Using pruning strategies, in which whole branches are removed instead of just the nodes, the result will be a completely empty dataset or no change in data at all. This can be seen in the top two images in figure 4.2.

4.2.4 Some results

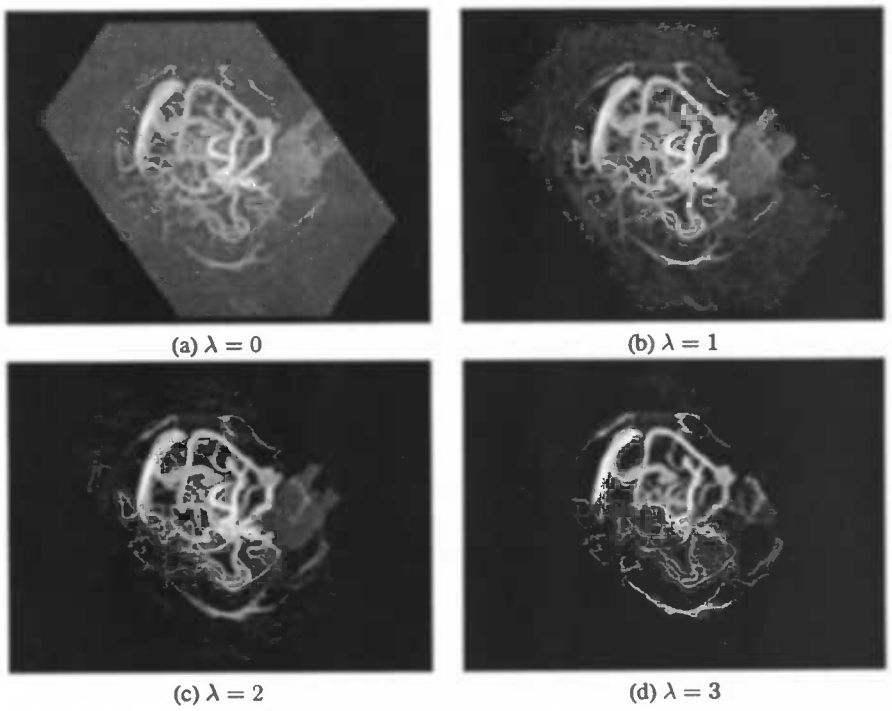


(a) $\lambda = 0$ (b) $\lambda = 1$

(c) $\lambda = 2$ (d) $\lambda = 3$

Figure 4.3: Filter results of **CTA1** using the elongated filter with different λ_f. $\lambda_f = 0$ shows all data. Gamma is exaggerated to show the noise reduction.

The initial construction of the Maxtree in which the whole dataset is flooded takes some time. The filtering on the other hand turns out to be fast enough to let a user choose a threshold interactively and view the result depending on the speed of the visualization. Table 4.1 on page 82 shows some timings of the interactive part using λ_f as a filter parameter. These timings are made on a Pentium 4 at 1.9GHz with 512MB memory. The column 'filter' gives the time in milliseconds it takes to filter the data. The column 'size' gives the amount of voxels remaining after filtering in kilobyte. For $\lambda_f = 0$ this amount is not the same as the total number of voxels, because the voxels with the background color are always discarded.



(a) $\lambda = 0$ (b) $\lambda = 1$

(c) $\lambda = 2$ (d) $\lambda = 3$

Figure 4.4: Filter results of **MRA2** using the elongated filter with different λ_f. $\lambda_f = 0$ shows all data. Gamma is exaggerated to show the noise reduction.

Generating the Maxtree from **MRA2** takes one and a half minute and **CTA1** takes half a minute. The second dataset is larger, but does not have as much grey levels as the first. These extra grey levels generate a lot of extra nodes in the Maxtree which in turn cause the generation to take longer.

The result of the filter itself can be seen in figure 4.3 and figure 4.4. The images show the result of the filtering using the elongation criterion. It shows that by increasing λ_f only the elongated structures remain.

4.3 Visualization with the Maxtree

As we already stated the Maxtree is extended with a few extra features. Every node knows its parent, its children and the voxels it contains. In the original implementation from Urbach every pixel knows to which node it belongs. To retrieve the filtered result, still all voxels have to be visited to see to which node it belongs and what its current grey value is. Since Urbach's implementation worked with $2D$ datasets this was not a problem, but in the $3D$ case this is a huge disadvantage,

especially for visualization.

A data structure is introduced which can easily be extracted from vqueue used in section 4.2.3 and also allows fast visualization. The structure consists of three lists:

- a list V of grey values
- a list C with the number of voxels with a particular grey value
- a list P with voxel positions.

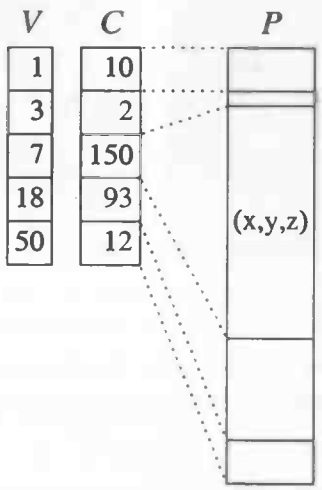


Figure 4.5: The data structure that is used to transfer the voxel data from the filter to the visualization. The lookup table V contains the grey values. C contains the number of voxels with that grey value and P contains the voxel positions.

The fast visualization part is achieved by transferring only meaningful voxels, i.e. not background voxels, from the filtering part to the visualization part. These are not stored in these lists. The background color is even used as a marker for the end of list V. Extracting these list from vqueue is straightforward as can be seen in the pseudo code below:

```
vp=0
pp=0
for i=minlevel to maxlevel [
  if vqueue[i].size>0 [
    V[vp]=i
    C[vp]=vqueue[i].size
    vp++
```

```
    for j=0 to vqueue[i].size [
      P[pp]=vqueue[i].element[j]
      pp++
    ]
  ]
]
V[vp]=minlevel
```

An advantage of this method is the known upper bound of these lists. V and C have the same length which is no longer than the number of grey levels. P can never be longer than the total number of voxels minus the number of background voxels of the original dataset. During the execution of the program even less memory is used, especially because after filtering more voxels become background voxels and P will be even shorter. Reallocating these lists however is rather costly and not necessary because only the first part of the list is used and hardware caching techniques handle this correctly. This structure however should not be used to query a certain voxel position for its grey value.

Now that we have the data we can take a look at how to render this. All SF methods require a preprocessing step to extract the iso-surfaces from the data which is very expensive. Although the generated geometry can be rendered using widely available hardware implementations, changing the geometries, i.e. changing filter parameters, requires another expensive preprocessing step to generate the new geometry. The gain of the quick refiltering with a different parameter is lost when an expensive surface extraction algorithm has to be run.

The main disadvantage of a DVR method is the amount of data it usually has to process. As seen in section 4.2.4 a large part of the data is classified as background and thus contains no relevant voxel data. As noted in this section extracting this relevant voxel data can be done quickly and is easily accessible. This makes DVR very suitable. Any object-order rendering method that does not impose a particular spatial ordering on the voxels can be used.

We have chosen splatting since it is a rather straightforward technique. Splatting, as noted in section 4.1.2, is done by projecting the voxels onto the image plane. First a transformation matrix is calculated depending on the view angle. After this the projection is done via the following algorithm:

```
p1 = 0
p2 = 0
while V[p1]!=backgroundColor [
  for i=0 to C[p1] [
    project position P[p2] with color V[p1]
    p2++
  ]
```

```
  p1++
]
```

Because all voxels to be rendered have a grey value other than the background, the background color is used as the stop criterion. The projection can produce different kind of images depending on the blending performed in the view plane. We implemented an X-ray and a MIP renderer. Each have their own blending function and their own way of compensating for gaps that arise because adjacent voxels in the dataset are not necessarily projected to adjacent pixels in the image plane.

4.4 X-ray rendering

When creating an X-ray image using splatting, no spatial information is necessary for blending the footprint with the resulting image. A projected footprint is simply added to the current image. An obvious advantage of this is that we do not need to take into account a z-buffer or a local maximum. We can interpolate the resulting footprint along the pixels without problems. After all voxels have been splatted, the pixel values are scaled between the upper and lower boundary of the available luminance range.

Although the result is satisfactory, some additions to the splatting algorithm can be made to make it look better and run faster, especially the handling of the footprint.

4.4.1 The view-direction dependent footprint

The size and shape of the footprint is important for the quality of the resulting image. Making the footprint to big will cause blurry images. Making it too small will generate gaps between voxels that are adjacent in the dataset, but when translated lie too far apart from each other. To make things even worse, the ideal footprint size varies with the view direction and, in case of perspective correct rendering, depth.

By excluding perspective the ideal footprint is not dependent on the voxel position but only on the rotation. Since all voxels are rotated by the same amount, every voxel in a single rendered frame has the same ideal footprint. This footprint now only depends on the view direction. A $1 \times 1 \times 1$ voxel cube is rotated by the same amount as the dataset and rendered onto a 3×3 pixel footprint. The percentage of area filled in a certain part of footprint denotes the grey level of that part (figure 4.6). This footprint is used to splat the voxels. The advantage of this method is that looking along an axis produces a sharp image. In that case the rotated cube

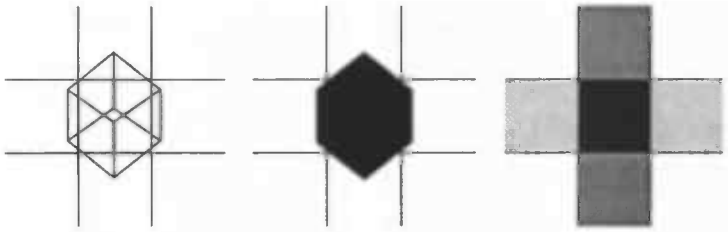


Figure 4.6: Different steps in the footprint calculation

produces a footprint of just one pixel. Not looking along any axis minimizes the amount of gaps because a bigger footprint is created.

4.4.2 Convolving the footprint

In the case of an X-ray renderer it is not necessary to place a splat for every splatted voxel. The same effect can be reached by adding only the bilinear interpolated pixels to the image and convolving the resulting image with the footprint. This way the footprint only has to be applied for every pixel in the view-plane instead of every projected voxel.

4.5 MIP rendering

Another method which does not depend on spatial ordering is maximum intensity projection. Instead of integrating along a view line, the maximum encountered value is taken. The retrieving and transforming of the voxels is done in the same way as for X-ray rendering. Since the maximum is needed along a ray, nearest neighbor interpolation is used. This way a value can be compared with the value already placed at that position. Also the maximum is not lost because it is not distributed across multiple pixels. On the other hand, using nearest neighbor interpolation forces us to take another approach towards the removal of the generated gaps. The error introduced by using nearest neighbor interpolation is 0.5 pixels at its maximum.

4.5.1 Morphological footprint approach

Using the same method as with X-ray images will require a much bigger footprint to compensate the generation of gaps. These gaps can be 0.5 pixels wider then in the X-ray case so the footprints also have to be at least 0.5 pixels wider. This will

cause a high amount of blurring and loss of detail. In this case we use a morphological approach to remove the gaps. The morphological approach is discussed by Roerdink[19]. He proved that the footprints for classical splatting can be replaced by a morphological closing in the view plane.

4.6 Additional improvements

Other additions can be made independent of the different rendering method. Most of these methods are aimed toward a better perception of the visualized data without decreasing the speed of rendering significantly.

4.6.1 Nonlinear luminosity mapping

In case of an X-ray rendering, big objects and long objects perpendicular to the view plane generate bright objects. This is because a large number of of voxels are splatted to the same position on the viewing plane. This can cause smaller objects to disappear because they do not have enough voxels contributing to make them bright enough. Especially in a vascular image it is not unthinkable to have a vessel perpendicular to the view plane which causes all other data to fade to black. A good example can be seen in figure 4.8(a). The bright spot in the top center is such a vessel. All other detail is much less visible.

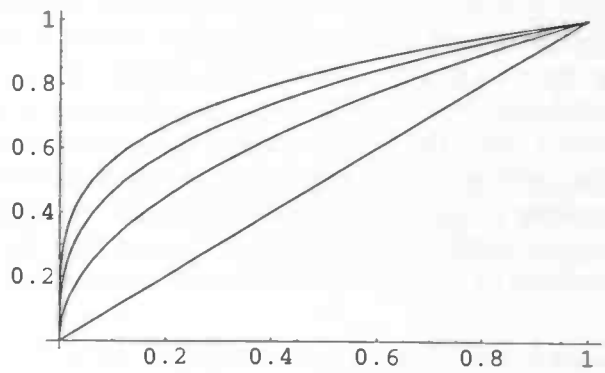


Figure 4.7: Different mapping curves for $m = 1$, $m = 2$, $m = 3$, and $m = 4$.

A solution to this problem is mapping the result of the integrated X-ray result onto the luminosity of the resulting image in a non-linear way, also called gamma

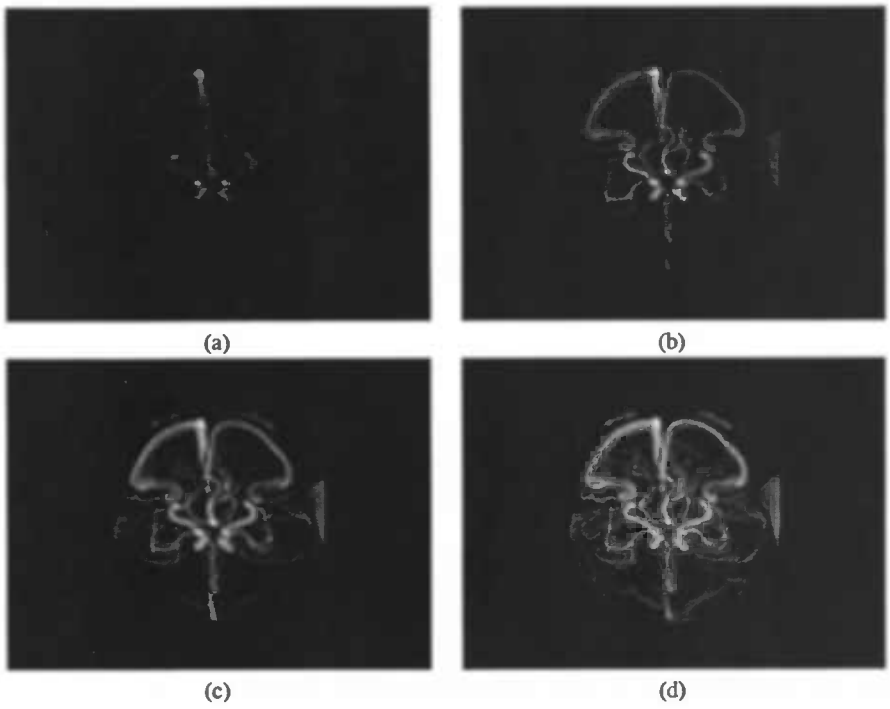


Figure 4.8: Influence of different mapping values. Resulting X-ray rendered images with mapping parameter $m = 1$ (a), $m = 2$ (b), $m = 3$ (c), and $m = 4$ (d).

correction. This is done by the following formula[1]:

$$dl = dl_{\text{min}} + dl_{\text{range}} \left(\frac{sl - sl_{\text{min}}}{sl_{\text{range}}} \right)^{1/m}. \tag{4.1}$$

In the formula sl stands for the source luminosity and dl stands for the destination luminosity. When $m = 1$ the mapping is still linear. Increasing m will cause a flattening in the high luminosities and a stretch in the low luminosities as can be seen in figure 4.7. Figure 4.8 show the result for $m = 1$, $m = 2$, $m = 3$, and $m = 4$.

4.6.2 Depth cueing

Since perspective is not used, adding another method to increase depth perception is needed. Turlington and Higgins showed[28] that making objects that are further from the viewer darker aids the perception of depth.

[1]The formula uses $^1/_m$ instead of m because the algorithm uses the inverse of this formula.

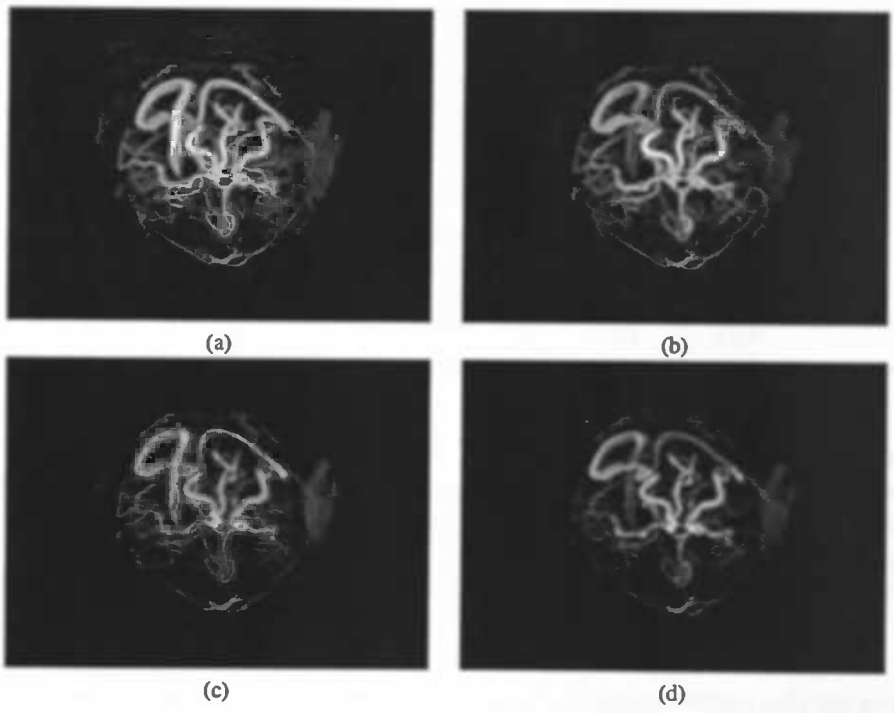
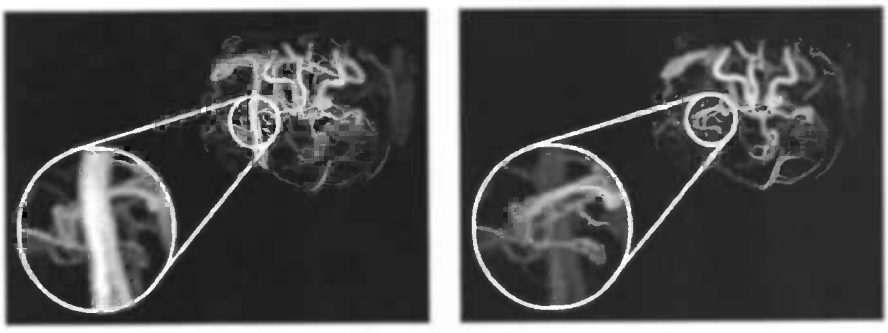


Figure 4.9: Influence of darkening further objects on MIP rendering ((a) and (b)) and X-ray rendering ((c) and (d)). On the images on the right it is clearly seen that the vessel in the top left is further away.

The luminosities are multiplied by the distance from the far clipping plane. This plane is perpendicular to the view plane and is, for now, positioned behind all the voxels. This is done before the splatting to the view plane. For X-ray rendering this just focuses the attention towards the structures in front. Besides this effect, a MIP rendering is also influenced in another way. Bright voxels in the back can be overwritten by less bright pixels in the front because the forward pixels become brighter. This creates a sort of depth sorting of objects. This effect can be clearly seen in figure 4.9 and 4.10. In the right side image of 4.10 the little vessels are obviously in front of the big vertical vessel while this can not be seen in the left image.

For efficiency reasons depth cueing is implemented before the nonlinear luminosity mapping. The depth queueing needs a normalization to keep the grey values within the range. Since this normalization is also done in the nonlinear luminosity mapping the multiplication is done before the luminosity mapping. To compensate for the non-linear behavior of the luminosity mapping the grey value should be multiplied by the distance to the power of m instead of just the distance. Since m is a float and a power operation is costly, especially if it has to be done for every



(a) (b)

Figure 4.10: Example of a bright object in the back overlapping a less bright object in the front. (a) The big vessel in the back is drawn 'in front of' smaller vessels. (b) No overlapping because the darkening of objects in the back does not make it bright enough.

voxel, this is replaced by a fixed integer power operation that is implemented by multiplications. Tests have showed that a value of $m = 3$ gives good results for both types of renderings (see figure 4.8) so this value is embedded in the depth luminosity algorithm.

4.6.3 Other small visual enhancements

Another implemented extension is the ability to use clipping planes and a bounding box. The bounding box causes the viewer to see more intuitively what the current orientation is. It can also be used as a reference point for other features as will be shown with the clipping planes.

The user controlled clipping planes make it easier to isolate certain features which would normally be occluded by bigger or brighter objects. The front and back clipping plane are parallel to the viewing plane. The user can shift both planes forwards and backwards. To give the user a better perception of the position of the clipping planes, a bounding box is drawn together with a line across the surface of the bounding box to show where the clipping planes intersect with it.

4.6.4 Some results

The 'render' column in table 4.1 shows the amount of milliseconds it takes to render a frame. The 'frame/s' column gives the frame rate of rendering combined with filtering and show that interactive investigation of the data and the effect of a different λ_f parameter is possible.

	λ_f	size (kB)	filter (ms)	render (ms)	frames/s
CTA1	0	165	15	154	5.9
256 x 256 x 256	1	142	13	133	6.9
255 grey levels	2	132	12	125	7.3
	3	58	7	65	13.9
	4	7	4	26	33.4
MRA2	0	6008	392	4478	0.2
256 x 256 x 124	1	510	71	396	2.1
1322 grey levels	2	143	40	120	6.2
	3	57	33	55	11.3
	4	47	32	48	12.5

Table 4.1: Timing results on the set of test data. The first column contains some information on the data set. A more detailed description of these datasets can be found in section 3.5. The next columns lists the different values for λ_f with corresponding data set size (in kilobytes) of the result after filtering, the filtering time using the subtractive strategy and rendering time (in milliseconds), and frame rate.

This frame rate however is only applicable when λ_f changes. When only the rotation changes the frame rate is only based on the render time and thus even faster.

One timing differs significantly from of the rest. Rendering **MRA2** with $\lambda_f = 0$ has a frame rate of 0.2. This is because **MRA2** contains a lot of noise, especially in the background. This can also be seen in figure 4.2(b) which is a MIP rendering of the complete dataset because the max filter does not reject anything. When $\lambda_f = 0$ this noise is not removed as can be seen in the 'size' column. When λ_f is increased however the size drops significantly because most of the noisy background is now classified as being background and the frame rate rises to an acceptable rate.

4.7 Conclusion

The Maxtree splits the filtering task into two separate stages. The first stage, in which the tree structure is constructed, takes long compared to the second stage, in which the actual filtering is done. The first stage however is independent of filter parameters or filter type which means it only has to be done once per dataset. We have shown that filtering and visualizing can be done at a reasonable frame rate using standard hardware. Compared to Meijster et. al. [17] our implementation is faster because we only consider foreground voxels and do not rebuild the whole volume. These where the bottlenecks in their implementation. Their timings are similar to our timings when no filtering is applied. When filtering is applied though, our implementation is up to ten times faster.

Chapter 5

Discussion

We developed a vessel segmentation method and which works fine in the case of the test images we used. When applied to other angiograms other results might be obtained, which are not necessarily better or worse. We did not reach the goal of full automatic vessel segmentation, but the developed method is fast, easy to implement and user interaction is hardly needed. The number of tunable parameters can even be reduced by estimating the noise level η from the images.

Other possible improvements include trying other threshold-selection strategies when RATS cannot determine a threshold. Also, pre- and post-processing methods can be further developed and improved. Optimal values for pre-segmentation attribute filtering could be found, as well as better criteria to improve the segmentation afterwards during the connected component analysis. Maybe segmentations can be even improved afterwards by attribute filtering of the volume again.

The implementation of the Maxtree class in `C++` is done in such a way that extending it with new features or using it in other applications can be done easily by extending one or more of the available classes. An example addressed in this document is the abstract `Attribute` class which makes it possible to implement a new attribute type without changing the Maxtree sources itself.

The attributes stored in the Maxtree are now only used as a parameter for the filter algorithm. It is not unthinkable these values can be used for other purposes. Especially the component structure inherent to the Maxtree structure can be used for a better progressive refinement method. The incrementing is not done by for example increasing the resolution, but by starting at big structures at full resolution in the Maxtree and refining it by adding smaller structures.

Attribute filtering turned out to be not only a clear and fast way for visualizing blood vessels, it is also an important addition for segmentation of blood vessels. Filtering filamentous details before segmenting gives considerably better results than segmenting the unfiltered dataset. Furthermore, the visualization method used can even be used to visualize (and possibly judge) the segmentation in a fast way. Recall that the splatting visualization method works fast, because we visualize only the relevant (foreground) voxels. RATS (and all other segmentation methods which give a binary output) divides the images in relevant (vessel) voxels and irrelevant voxels, and so can be visualized quickly.

Bibliography

[1] E. Artzy, G. Frieder, G. T. Herman, and H. K. Liu. A system for three-dimensional dynamic display of organs from computed tomograms. In *Proceedings of the 6th Conference on Computer Applications in Radiology and Computer-Aided Analysis of Radiological Images*, pages 285–290, Newport Beach, California, jun 1979.

[2] S. R. Aylward, J. Jomier, S. Weeks, and E. Bullitt. Registration and analysis of vascular images. *International Journal of Computer Vision (In Press)*, 2003.

[3] A. C. S. Chung and J. A. Noble. Statistical 3D vessel segmentation using a rician distribution. In *MICCAI '99*, pages 82–89, 1999.

[4] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM (JACM)*, 39(2):253–280, 1992.

[5] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 65–74. ACM Press, 1988.

[6] T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, August 1992.

[7] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever. Multiscale vessel enhancement filtering. *Lecture Notes in Computer Science*, 1496:130–137, 1998.

[8] G. Gerig, M. Jomier, and M. Chakos. Valmet: A new validation tool for assessing and improving 3D object segmentation. *Lecture Notes in Computer Science*, 2208:516–523, 2001.

[9] J. Kittler, J. Illingworth, and J. Föglein. Threshold selection based on a simple image statistic. *Computer Vision, Graphics, and Image Processing*, 30(2):125–147, May 1985.

[10] P. Lacroute and M. Levoy. Fast volume rendering using a shear–warp factorization of the viewing transformation. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 451–458. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[11] M. D. Levine and A. Nazif. An experimental rule-based system for testing low level segmentation strategies. In Kendall Preston, Jr. and Leonard Uhr, editors, *Multicomputers and Image Processing*, pages 149–160. Academic Press, 1982.

[12] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[13] W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California, July 1987).

[14] L. M. Lorigo, O. D. Faugeras, W. E. L. Grimson, R. Keriven, R. Kikinis, A. Nabavi, and C. Westin. Curves: Curve evolution for vessel segmentation. *Medical Image Analysis*, 5:195–206, 2001.

[15] T. Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.

[16] Y. Masutani, T. Schiemann, and K. H. Höhne. Vascular shape segmentation and structure extraction using a shape-based region-growing model. *Lecture Notes in Computer Science*, 1496:1242–1249, 1998.

[17] A. Meijster, M. A. Westenberg, and M. H. F. Wilkinson. Interactive shape preserving filtering and visualization of volumetric data. In *Fourth IASTED Conference Signal and Image Processing, SIP 2002*, pages 640–643, Kauai, Hawaii, USA, August 12–14 2002.

[18] F. Natterer. *The mathematics of computerized tomography*. Teubner/Wiley, Stuttgart/Chichester, 1986.

[19] J. B. T. M. Roerdink. Comparison of morphological pyramids for multiresolution MIP volume rendering. In D. Ebert, P. Brunet, and I. Navazo, editors, *Data Visualization 2002. Proc. Joint Eurographics – IEEE TCVG Symposium*, pages 61–70, Barcelona, Spain, May 27–29 2002. Association for Computing Machinery.

[20] S. Dunne S. Napels and B.K. Rutt. Fast fourier projection for MR angiography. *Magnetic Resonance in Medicine*, 19:393–405, 1991.

[21] P. Salembier, A. Oliveras, and L. Garrido. Anti-extensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7(4):555–570, 1998.

[22] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction. *IEEE Transactions on Image Processing*, 4(8):1153–1160, August 1995.

[23] Y. Sato, S. Nakajima, N. Shiraga, H. Atsumi, S. Yoshida, T. Koller, G. Gerig, and R. Kikinis. 3d multi-scale line filter segmentation and visualisation of curvilinear structures in medical images. *Lecture Notes in Computer Science*, 1205:213–2227, 1997.

[24] T. Schiemann, M. Bomans, U. Tiede, and K. H. Hohme. Interactive 3d-segmentation. In *Proceedings of the Second Conference on Visualization in Biomedical Computing*, pages 376–383, 1992.

[25] R. M. Stefancik and M. Sonka. Highly automated segmentation of arterial and venous trees from three-dimensional mra. *The International Journal of Cardiovascular Imaging*, 17:37–47, 2001.

[26] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.

[27] T. Totsuka and M. Levoy. Frequency domain volume rendering. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 271–278, August 1993.

[28] J. Z. Turlington and W. E. Higgins. New techniques for efficient sliding thin-slab volume visualization. *IEEE Transactions on Medical Imaging*, 20(8):823–835, August 2001.

[29] E. R. Urbach. Automatic identification of diatoms using multi-scale mathematical morphology. Master's thesis, University of Groningen, 2001.

[30] E. R. Urbach and M. H. F. Wilkinson. Shape-only granulometries and grey-scale shape filters. *ISMM 2002*, pages 305–314, 2002.

[31] L. Westover. SPLATTING: A parallel, feed-forward volume rendering technique. TR91-029 (Ph.D), Dept. of CS, University of North Carolina, 1991.

[32] M. H. F. Wilkinson. Optimizing edge detectors for robust automatic threshold selection: Coping with edge curvature and noise. *Graphical Models and Image Processing*, 60:385–401, 1998.

[33] M. H. F. Wilkinson and J. B. T. M. Roerdink. Fast morphological attribute operations using Tarjan's union-find algorithm. In J. Goutsias, L. Vincent, and D. S. Bloomberg, editors, *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 311–320. Kluwer, 2000.

[34] I.T. Young and L.J. van Vliet. Recursive implementation of the gaussian filter. *Signal Processing*, 44:139–151, 1995.

[35] S. Young, V. Pekar, and J. Weese. Vessel segmentation for visualization of MRA with blood pool contrast agent. *Lecture Notes in Computer Science*, 2208:491–498, 2001.

List of Figures