University of Groningen **Faculty of Mathematics and Natural Sciences**

Department of Mathematics and Computing Science

Explorative volume rendering using the Max-Tree datastructure

Ilja Plutschouw

Supervisors: *dr. M.A. Westenberg, dr. M.H.F. Wilkinson*

prof.dr. J.B.T.M. Roerdink

April 10, 2007

RuG

Contents

Chapter 1

Introduction

When working with volume data such as CT or MRI scans of the human body, it is often challenging to interpret the data. The volume is projected on a two-dimensional image, often resulting in overlapping elements in the image. Techniques have been developed to make it easier, like having an interactive view, looking at 2D slices through the volume, using various colors schemes and applying filters to the data. There are many methods to select components in the volume data and color the volume, but the size of volume datasets makes it hard to use them interactively. The Max-Tree datastructure makes it possible to experiment with new techniques to explore volume data. It allows more speed operations dealing with connected components, making it possible to use them interactively.

This thesis is about explorative volume rendering. The goal of my research was to develop a tool for explorative volume rendering based on the Max-Tree datastructure representation of data. In explorative visualization, the user plays an active role in the way data is represented. The user can change selections, parameters or viewpoints to get a better insight in complex datasets. With volume rendering there is the problem that a volume is rendered to a two dimensional image. Some details might be hard to spot without removing parts of the volume that obstruct the view.

This research looks at ways to modify the volume dataset based on the Max-Tree data structure. This structure holds a decomposition of a volume dataset, and allows modifications without changing the structure of the objects inside. The structure allows selections of components based on shape or size, making it easy to remove or select parts with certain features of a volume. Using data structures for explorative visualization was shown useful before with contour trees [2], where a network of contours at different isovalues is built.

The Max-Tree data structure can also be used to construct various transfer functions. These functions give colors or opacity values to the voxels in the volume, to create more clear rendering of a dataset. There are many attribute setting that can be used to determine parts that should belong together, and have their own color. Part of this research will be to find some useful ways to construct these transfer functions based on the structure and attributes of the dataset.

This report starts with some background information about the subjects that were important for the research. What follows are the actual research subjects, about the rendering of the Max-Tree data structure and its interaction and about the construction of transfer functions. It will end with a discussion about the results.

Chapter 2

Background

2.1 Volume visualization

Volume rendering is a technique for visualizing a discrete three-dimensional dataset by projecting it onto a plane. The goal is to get a better insight into the dataset by looking inside, using transformation, segmentation, translucency, or other methods. The data can come from different sources, such as scans from real objects, computer simulations and geometric models [9]. Sampled data can be medical images taken using CT, MRI or cryosection and can also come from biology, industry or geoscience sources. Applications for weather or flow simulations typically result in three-dimensional data. Computer-aided design applications can use basic geometric models to generate or manipulate volume data. Volumetric data is typically represented as a three-dimensional regular grid of samples. Each sample is a binary, multivariate or vector value that represents a property of the data, such as density, pressure or velocity. One sample in a volume dataset is called a voxel (abbreviation for volume element).

Many methods have been developed to visualize volume data. The early developed methods were mainly indirect, but today direct volume rendering methods are widely used as well. Indirect volume rendering first extracts geometric information from the dataset, a well known example of this is the Marching Cubes algorithm [10] that extracts an iso-surface of the data and only renders the extracted surface. The advantage of indirect methods is speed as compared to direct volume rendering, at the cost of lost information. In direct volume rendering the complete volume is rendered by mapping every sample to a color and opacity and then rendering all the voxels to the screen.

There are three main approaches to direct volume rendering [9], object-order, image-order and transform methods. In *object-order* methods the volume is rendered by calculating the contribution of every voxel on the image plane, and combining all the voxels in the data. Splatting is an object-order method. *Image-order* methods start with the pixels on the image plane by casting a ray through the volume and combining the result of all the voxels on that ray to determine the color of the pixel. Ray-casting and the texture-based volume rendering methods are both examples of this. In *transform* methods, the volume data is first transformed to another domain, for example the frequency domain, before being projected. Figure 2.1 gives an example of an image of a volume dataset, rendered using X-Ray splatting.

The remainder of this section gives more background about specific subjects in volume visualization that are of importance for this thesis. Transfer functions will be introduced as a means to determine color and opacity for individual voxels. Two direct volume rendering methods, splatting and texture-based volume rendering, are explained in more detail. These were used in the devel-

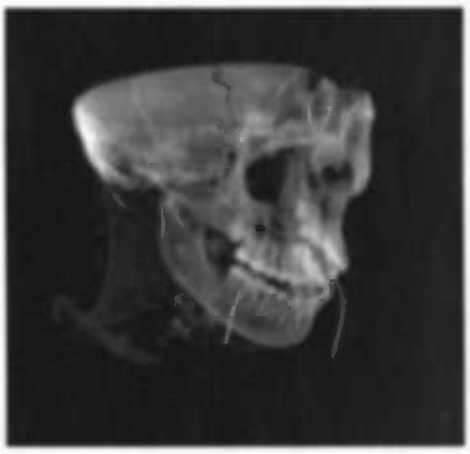


Figure 2.1: *Volume rendering of a human head, using X-Ray splatting.*

opment of the research tools. Finally some background is given about explorative visualization in volume rendering.

2.1.1 Transfer functions

A transfer function is a function that describes the relation between the input and output. In volume visualization transfer functions are used to determine a color and opacity of a voxel, given its properties, to determine what part of the volume data is visible. The property of a voxel is often a scalar value, but also vector attributes can be used by defining a multi-dimensional transfer function. In a multi-dimensional transfer function, different kind of properties are combined to form the final output. Examples of properties that can be used are volume, intensity, eccentricity and derivatives of volume components. The goal of a transfer function is to show or highlight certain parts of properties of a dataset. In CT images one might use a transfer function to highlight the bone tissue by making voxels with a high intensity completely opaque, and making voxels with a low intensity somewhat transparent. In other cases the edges of objects give interesting information. These edges can be highlighted by making voxels with a high gradient more opaque than voxels with a low gradient.

Transfer functions can give a linear relation between the input parameter and the output. In rendering an image from CT scans, the measured density can be linearly transformed to an opacity. But when the interesting parts of the data only occur in a certain range of the input values, non-linear transfer functions are often more informative. Examples are power functions and manually constructed functions. Figures 2.2 and 2.3 give an example of how an input can be mapped using two different transfer functions.

Figure 2.2: *Tables of two different color mappings.*

2.1.2 Splatting

Splatting [5] is a method to make a projection of a volume dataset. Every voxel in the dataset is projected on the viewplane, and rendered using a two-dimensional kernel that represents a single

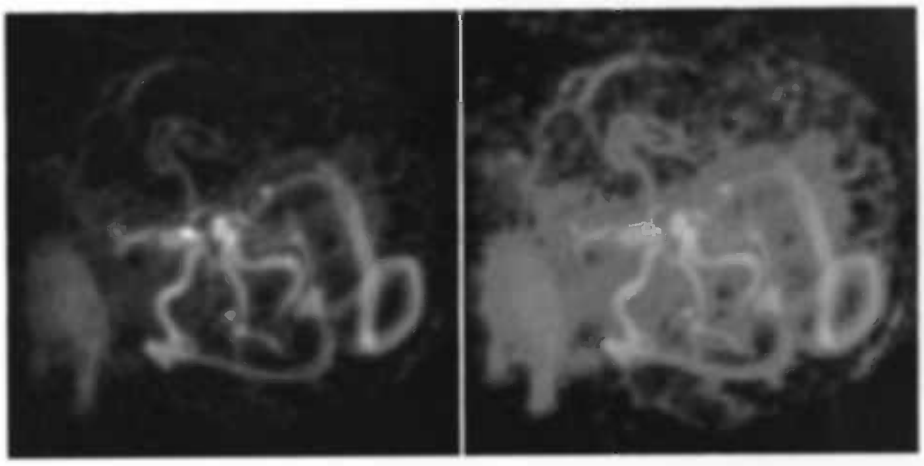


Figure 2.3: *Two volume rendered images using the transfer functions from figure 2.2.*

voxel projection. The names stems from the likeliness to throwing snowballs on a glass surface, where the snow contribution at the center is high and will drop when further away from the center. Every voxel is represented as a sphere or an ellipsoid for square or rectangular voxels. If the projection is orthographic, the projected shape of the sphere or ellipsoid is the same for every voxel position. Therefore a two-dimensional footprint of the projection of the sphere or ellipsoid can be created that represents the voxel on the view plane. The voxels must be processed from front-to-back, and the color and opacity are determined by a transfer function. Splatting is mainly suited for orthographic projections, since in an orthographic projection, the kernel of each voxel is the same. A perspective projection is also possible, but a new kernel has to be computed for every voxel in the dataset.

X-ray projection of the volume data can be performed more efficiently [14]. The order in which the voxels are processed is not important. The centers of the footprint are added to the viewplane, and the final result is convolved with the footprint. Performancewise this is much more attractive than rendering a separate kernel for every voxel, and the results are the same. The result is an image where each pixel represents the sum of densities of a line through the volume, just like an X-ray image. The density is represented as a float value and can have a large range, therefore this data must be transformed in order to render it to the screen. A *transfer function* is used to map the densities to a color palet that is usable by the graphics hardware.

2.1.3 Texture-based volume rendering

In recent years [16], the fast texture capabilities of modern graphics hardware has been used to implement volume rendering. Although CPUs get faster every year, graphics hardware is highly specialized in rendering texture based surfaces, thereby outperforming the CPUs. This capability can be put to use in volume rendering. The volume data is loaded onto the graphics hardware, and planes are rendered using a slice through the volume as a texture. There are two major methods to implement this, one uses the two-dimensional texture capabilities, and the other uses the three-dimensional texture capabilities.

The two-dimensional texture method first slices the volume data along an axis into separate images. These images are rendered to a set of planes that represent these slices in three-dimensional space. If the view changes and the sliced planes have an angle with the viewplane that is larger than 45 degrees, another axis is used to slice the volume data. The three-dimensional texture method uses

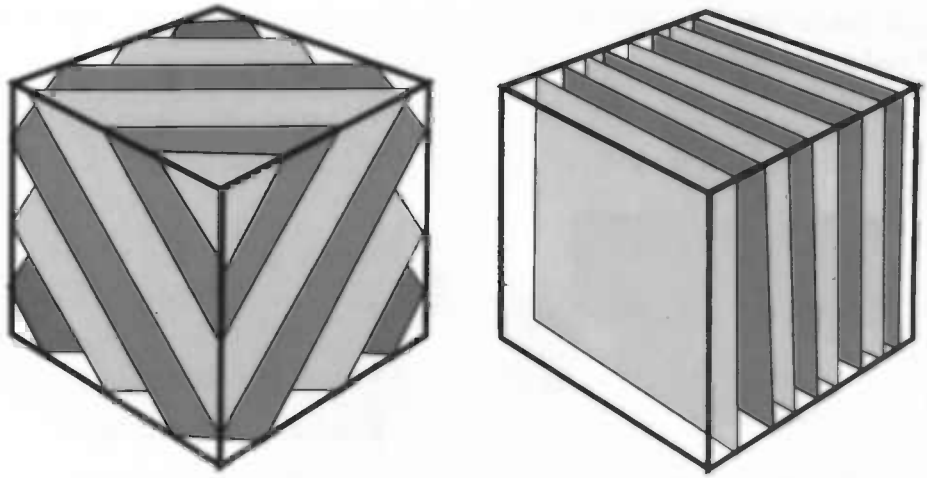


Figure 2.4: *Two texture based volume rendering method, camera aligned slices and volume aligned slices.*

the ability of the hardware to do volume texture lookups. In that case a set of slices perpendicular to the camera is needed, and the graphics hardware uses a texture look-up to find the correct voxel at one point. The slices are rendered from back-to-front, with background set to transparent.

In this research, texture-based volume rendering was used as a way to allow multicolor and transparency based transfer functions at interactive speeds.

2.1.4 Explorative volume visualization

A visualization of volume data has the problem that various parts of the data may overlap in the image, thereby hiding parts of the information. Various techniques have been developed to overcome this problem. Interactive visualization makes it possible to manipulate the rendering in real time. This allows basic operations like changes of the viewpoint. But sometimes more extensive manipulations are needed such as changes of the transfer function or cuts through the volume.

2.2 Connected components

Connectivity and connected components are terms that are commonly used in image analysis and they are used to determine various image object concepts, such as regions and boundaries. These objects can then be used for further operations, for example counting or feature extraction. This thesis deals with connected components in gray level discrete volume data.

2.2.1 Connectivity

Connectivity of voxels in volume data describes a relation between two or more elements. The neighbor function defines when two voxels in a grid are neighbors. There are multiple definitions possible for defining a neighborhood relation between voxels, but the two most popular are $N_6(p)$ and $N_{26}(p)$ connectivity. A voxel p at coordinate (x, y, z) has its N_6 neighbors at $(x-1, y, z)$, $(x+1, y, z)$, $(x, y-1, z)$, $(x, y+1, z)$, $(x, y, z-1)$ and $(x, y, z+1)$. The N_{26} neightborhood is defined by all 26 voxels directly surrounding p. A path from voxel p_0 to voxel p_n is a sequence $(p_0, p_1, p_2, .., p_n)$ such

that (p_{i-1}, p_i) are neighbors, with $i = 1, .., n$. When there is a path from one voxel to another, those voxels are said to be connected.

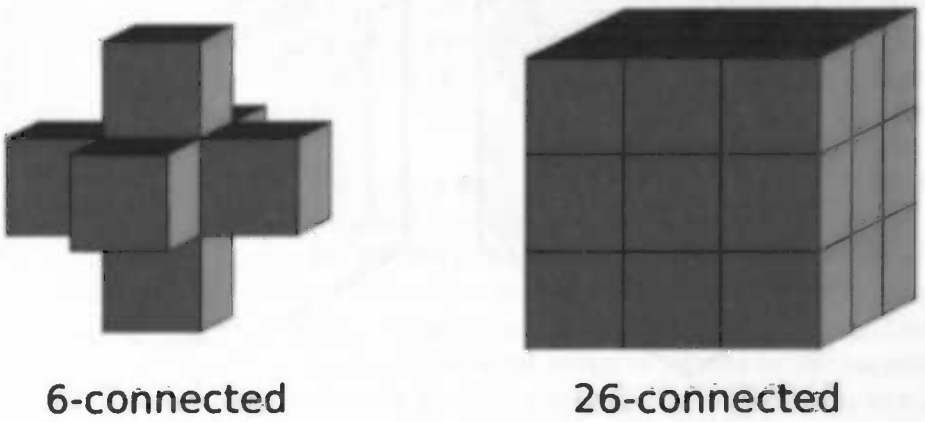


Figure 2.5: *Connectivity in three-dimensional volumes.*

In a binary image, a maximal set C of voxels that are connected to each other within C is called a *connected component*. Figure 2.6 shows an example of two-dimensional binary connected components where every connected component is drawn in a unique color. For three-dimensional volume data the situation is similar.

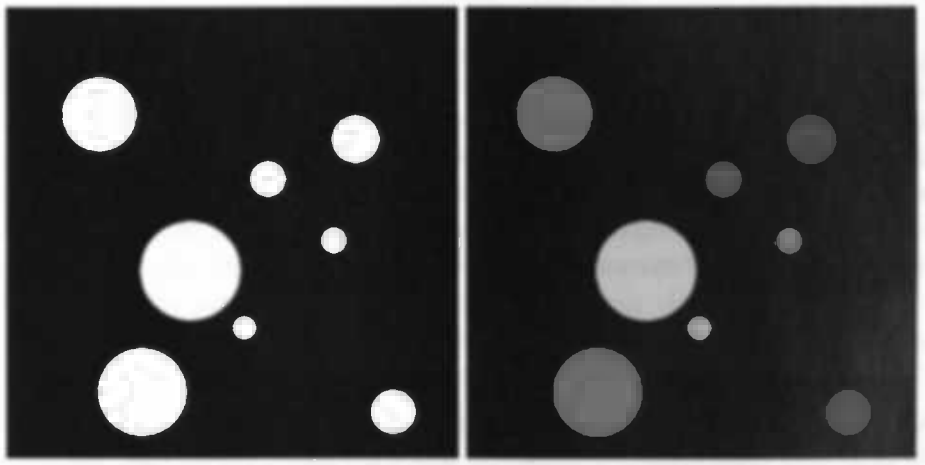


Figure 2.6: *Left is a binary image, on the right the same image decomposed in connected components.*

2.3 Anti-extensive connected operators

Anti-extensive connected operators [7], [13] are operators that work on sets of connected components. They act by either preserving or removing connected components from a set. These operators are especially useful for image analysis and computer vision tasks where the contour of the components is important. A connected component is preserved or removed on the basis of certain criteria. Examples of these are the area, the size or the shape of the connected component. When a connected component does not meet a criterion it is removed from the set.

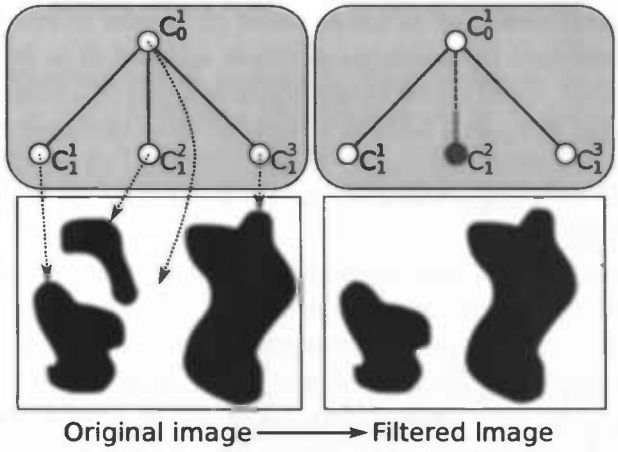


Figure 2.7: *Example of a binary connected operator which removes connected components below a certain area.*

Figure 2.7 gives an example of the filtering process in a binary image. The superscript denotes the number within the current gray level and the subscript denotes the gray level that the node has. The original image has a background C_0^1 and three connected components C_1^1, C_1^2 and C_1^3. These elements can be represented in a tree structure. Filtering is the process of computing an attribute value of each connected component, comparing it to a threshold λ and removing the component if its attribute value is smaller than the threshold. In figure 2.7 C_1^2 has an area that is smaller than λ and is removed from the tree, its pixels arefilter changed to the background color. This process of tree construction and filtering is explained more extensively in section 2.4.1 and 2.4.2.

The definition of anti-extensive connected operators can be extended to gray-level images. The image is thresholded at all possible gray levels, and all connected components in a gray level that do not satisfy the criteria are removed. After this, the filtered image can be reconstructed. Figure 2.8 shows an example image with 4 gray levels. Components with a small area are removed from the image.

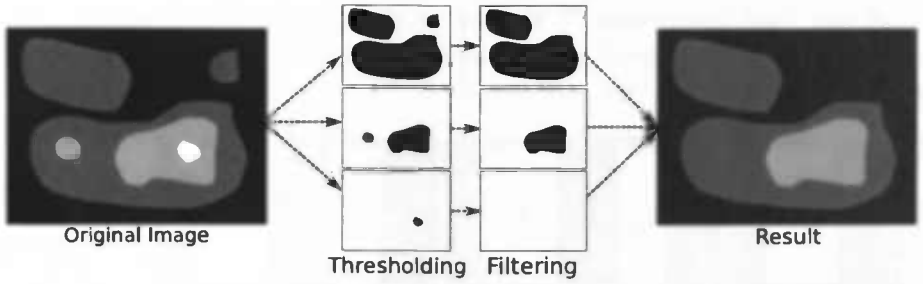


Figure 2.8: *Example of a gray-level connected operator which removes connected components below a certain area.*

2.4 The Max-Tree

The Max-Tree was first proposed by Salembier et al. [12] as an efficient datastructure to implement anti-extensive gray level connected operators on images. These operators can be used as attribute

filters that reject or accept details of an image based on chosen criteria. The Max-Tree is a tree datastructure where every node has a number of voxels assigned to it. For every node a number of attributes can be computed, based on for example size or shape of a connected component. A filter can be designed to accept or reject a node as a result of these attributes. The children of this node are then collapsed onto its parent. Because the structure contains all the connected components of all gray levels, the creation of attribute filters can be separated from the computation of the connected components in the volume.

In addition to images, the Max-Tree can be used for intensity based discrete volume data. In [17] the Max-Tree was used to enhance elongated structures in volume data, and in [15] it was shown how the Max-Tree can be used to implement interactive filtering on volume datasets. Max-Tree-based filtering consists of three phases. The *construction phase*, where the Max-Tree is constructed from an image or a volume. The *filter phase*, where nodes are removed from the Max-Tree according to certain criteria. And the *restitution phase*, where the image or volume is reconstructed from the filtered Max-Tree. The next part will describe the three phases of Max-Tree filtering in more detail.

2.4.1 Construction

The tree is created by thresholding the volume at all possible gray levels. Let $M \subseteq \Re^3$ be the volume domain, and $f : M \rightarrow \Re$ the gray scale volume. The set of points that remain after thresholding at level h is defined as:

$$X_h(f) = \{x \in M | f(x) \geq h\} \tag{2.1}$$

A *peak component* P_h^k is defined as a connected component in the threshold set $X_h(f)$. A Max-Tree node represents sets of *flat zones* of f. A *flat zone* is defined as the maximum size set $F \subset M$ where for all $p, q \in F$ there is a path from p to q with a constant function value. There is a Max-Tree node C_h^k for every peak component. The root node contains the flat zones at background intensity. The Max-Tree node C_h consists of the subset of P_h with gray level h. Children represent flat zones with a higher intensity than its parent, and the leaf nodes represent the local maxima of the volume. This is where the name *Max*-Tree comes from. A subtree represents a peak component. The attributes of a node are based on the peak component with the node as root of the subtree. Figure 2.9 gives an illustration of this. Peak component P_1^0 also covers the area beneath P_2^0 and P_2^1 and the flat zones of P_1^0 with $h = 1$ are the separated areas B, D and F. This defines a hierarchical structure of peak components, that form the Max-Tree.

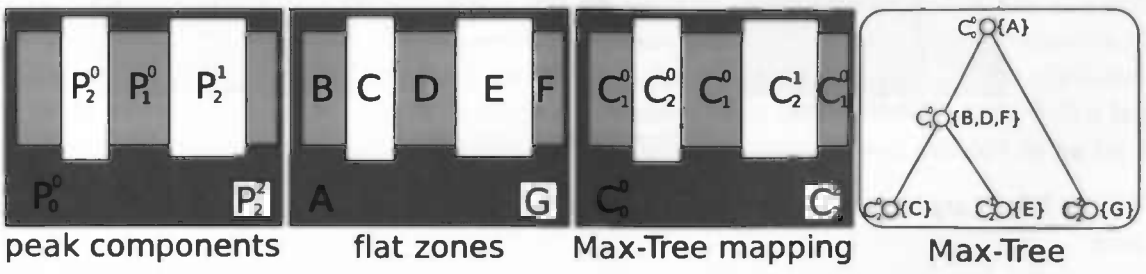


Figure 2.9: *Construction of a Max-Tree.*

2.4.2 Filtering

For every node, certain properties can be computed. These attributes can be related to size or shape of the peak components. In order to create a filter there has to be an order $\leq$ in the attribute values. The filter algorithm decides if a node should be kept or removed based on its attribute and a given threshold λ. If a node C_h^k is removed, the related voxels are lowered in gray level, to equal the level of the first ancestor that meets the criterion. The gray value of the children of C_h^k are lowered to the same level.

2.4.3 Attributes

The attributes in the Max-Tree are usualy computed over the peak component of a node. The voxels in the peak component are the same as one would get as a connected component for the gray level of the node. In this thesis only two attributes were used. The volume and the eccentricity.

2.4.4 Restitution and rendering

In this phase the filtered Max-Tree is used to reconstruct a volume dataset, and show it on the screen. In [15] an adjusted splatting function was used, designed to work with the result of the filtered Max-Tree. Also an isosurface constructor was presented. These methods do not reconstruct the complete volume but work directly from the Max-Tree. Another method does reconstruct the data in order to pass it on to a texture-based volume renderer.

Chapter 3

Max-Tree data structure rendering

The first part of the research is about drawing the Max-Tree data structure and interacting with it. The goal is to have the structure of the tree rendered to screen, see the result of attribute filtering and be able to interact with this structure to get feedback in the volume rendering of the dataset. This can give a better insight into the structure of a tree for a given dataset.

An existing C++ implementation of the Max-Tree data structure, the attribute filtering, a splat-based rendering of the dataset and an FLTK [1] graphical user interface are given. It was desirable that the implementation extends or interacts with this existing code base, so that this implementation can also work with future versions.

Section 3.1 describes what methods were considered to draw the tree, and why using an existing implementation was not possible. Section 3.2 describes the various layout algorithms that were implemented. Section 3.3 is about removing nodes from the tree in the data structure rendering to get a more compact and organized view of the Max-Tree. The last section is about rendering the layout to screen.

3.1 Search for layout algorithms

Much research effort has been spend on drawing of trees and graphs and various programs can be found that implement these algorithms. But the implementations of these algorithms couldn't just be used in this research. The requirements for this research are the ability to draw large tree graphs, to work interactively and easy integrate into the current C++/FLTK program. Various programs fit the first and second requirements. Programs like JGraph, aiSee and ILog are all written in Java, making it difficult to integrate with a C++ program. Graphviz does allow integration but only produces static output. Although it is possible to make a link between the C++ application and a Java application, the result will probably not be able to do interactive updates since the application deals with a large data structure. Because a good C++ implementation was not found, it was decided to go for an implementation described in the book "Graph Drawing" [4].

3.2 Layout

This section deals with the various tree layout algorithms that have been implemented during the research project. The requirements are that the algorithm should produce a layout where the structure of the tree is still clear, such that there are no overlaps between the nodes or the edges. Furthermore parent and child relations should be clear and the view should be compact. It starts with very basic

algorithms that do not take care of overlapping tree parts, and ends with an algorithm that takes care of crossing lines, overlap and space usage. The images in this chapter all show the same Max-Tree of the "angio" dataset. The "angio" dataset contains an MRI scan with dimensions 128x128x62 containing an angiogram, the Max-Tree of this dataset has 7118 nodes, see figure 3.1.

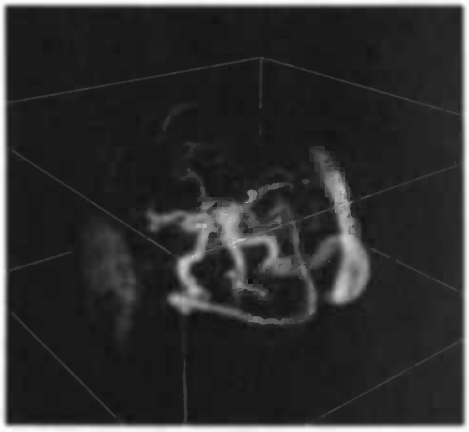


Figure 3.1: *The angio dataset with dimensions 128x128x62.*

3.2.1 Simple tree layout

The first layout algorithm is very simple and it was only used to get an initial feeling for the complexity of the tree, and to see if a more sophisticated layout was at all necessary. The implementation uses a recursive function that assigns for every node an x and y position. The root node is centered at location (0,0) and an initial spread for its children is determined. The children are placed a fixed unit below its parent, and horizontal distance between the children is based on the spread parameter. For every level the spread is divided by two, so that there is more room at the bottom of the tree for the increasing number of children.

```
structure Node
  x, y;
  children;
end

SimpleTreeLayout(node, x, y, spread)
  node.x = x;
  node.y = y;
  numChildren = node.children.size();

  // compute x position for first child
  xPos = -(numChildren-1) * spread * 0.5  + x;

  // iterate over all children
  foreach child in node.children
    SimpleTreeLayout(child, xPos, y+1, spread/2);
    xPos += spread;
```

```
  end
end
```

This algorithm gives good results for binary trees, but the Max-Tree can have many children per node, so the results were not very good. Figure 3.2 and 3.3 show that there is a lot of overlap between nodes. The result looks confusing, it is difficult to see the individual branches at the bottom of the graph because they are just too close to each other.

Figure 3.2: *Overview of a Max-Tree data structure rendered with the simple layout algorithm.*

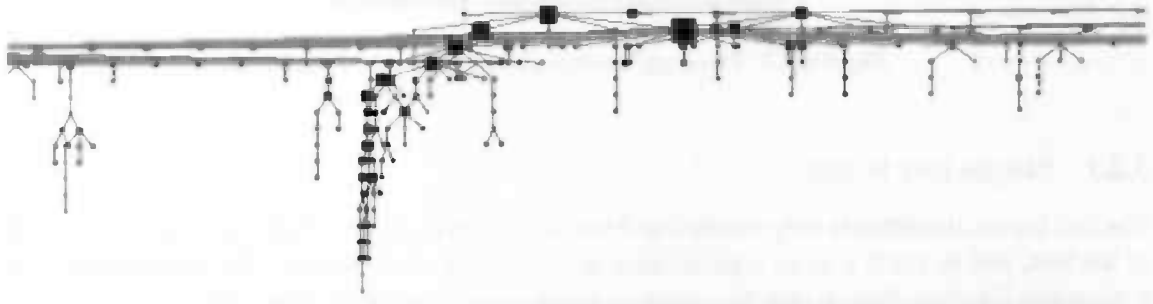


Figure 3.3: *Closeup of a Max-Tree data structure rendered with the simple layout algorithm.*

This simple layout algorithm did give some initial insight in the structure of the Max-Tree. Unlike more common tree structures, most of the nodes can be found in the upper layers of the tree. This makes the structure more difficult to show. There are also some deep branches with many single child nodes. This pattern is the same for Max-Trees generated from different datasets.

3.2.2 Rank layout

The rank-based tree layout determines the x-position of a node by determining its rank, and the y-position by determining its height in the tree. The rank of a node can be computed in three different ways, with a preorder, an inorder or a postorder traversal of the tree nodes. Every node is given a consecutive number, as the rank of this node. This layout algorithm prevents overlap of nodes, but the result gives a very wide view, which makes it difficult to navigate. Figure 3.4 shows the result of the preorder rank layout used on a Max-Tree data structure.

The parent child relations are not clear, and the tree is very wide. Although this layout does not have any overlapping parts, the result are not useful for rendering the large Max-Tree data structures.

3.2.3 Layered layout

The previous two layout algorithms did not give satisfactory tree drawings, so a better algorithm is needed. An algorithm that gives a better tree layout is the Layered-Tree-Draw algorithm, that can

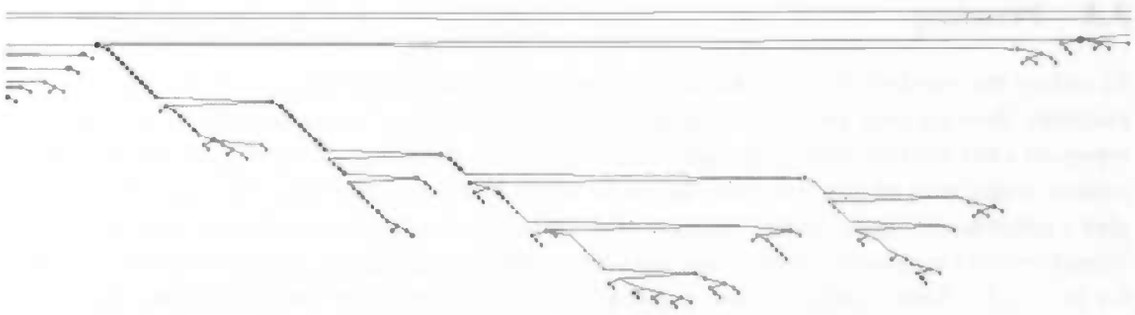


Figure 3.4: *Rendering of a Max-Tree data structure using the rank layout with preorder traversal.*

be found in [4]. This is a layout algorithm for binary trees. It uses a divide-and-conquer strategy to reduce the width using a local optimization heuristic. It is called a layered algorithm because the y-position for every node is simply determined by its depth in the tree, only the x-coordinates are computed by the algorithm. The Max-Tree is not a binary tree, so the algorithm had to be adjusted a bit to fit the needs.

The basic strategy consists of three steps. First, if a subtree is only one single node, nothing needs to be done. Second, if the subtree consists of multiple nodes the algorithm is first applied to its subtrees. Third, we shift the subtrees of this node in such a way that the minimal horizontal distance between two subtrees is two units, and put the current node one unit above the subtrees, and horizontally halfway between its children.

Figure 3.5 shows the result of a layered layout on a Max-Tree data structure. All the subtrees are nicely put next to each other, there is no overlap. Unfortunately the tree still needs a large amount of horizontal space because some of the top nodes have many children. One way to reduce this problem slightly would be by optimizing the layout algorithm to change the order of the child nodes but this is not implemented. Since there is no obvious ordering in the nodes, it is no problem to change the order. This way a more optimal ordering could be found that reduces the width of the graph. Another way is by reducing the number of nodes that are visible using various criteria, this is further discussed in section 3.3.

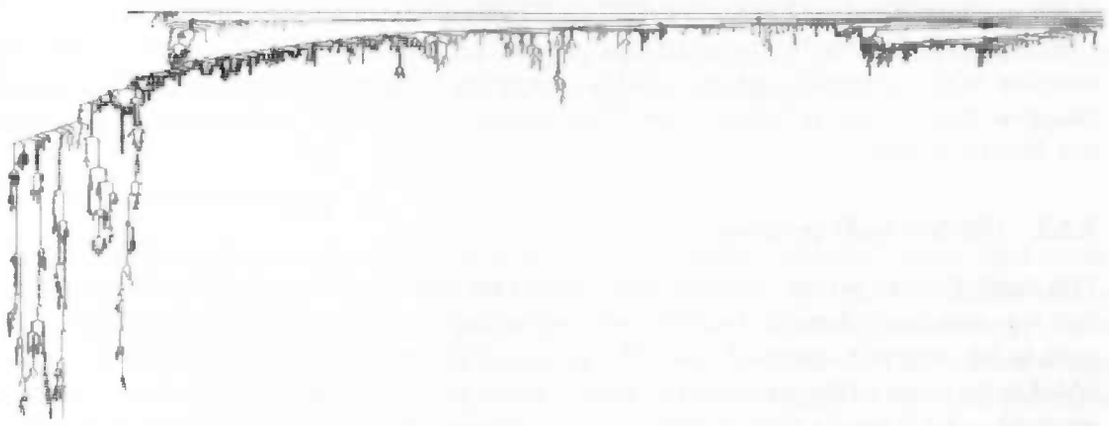


Figure 3.5: *Rendering of a Max-Tree data structure using the layered layout.*

3.3 Pruning

To reduce the number of nodes that are on screen, various tree pruning methods can be used. For example, showing only the nodes that have a certain number of children, only showing nodes that represent a flat zone or peak component with a large enough volume or by using the attribute value to remove nodes with an attribute value below or above a certain threshold. The removal of nodes can give a better insight in the global structure of the tree, but detailed information of the structure is lost. Therefore it is important to clearly state the various pruning algorithms, and to let the user adjust them for his needs. Three methods were implemented: collapsing nodes removed during the Max-Tree filtering, removing nodes with only one child in the filtered tree and removing nodes with less than a certain number of children in the non-filtered tree. Figure 3.6 shows a complete Max-Tree structure without any pruning. The methods described below will use the same data to show the differences.

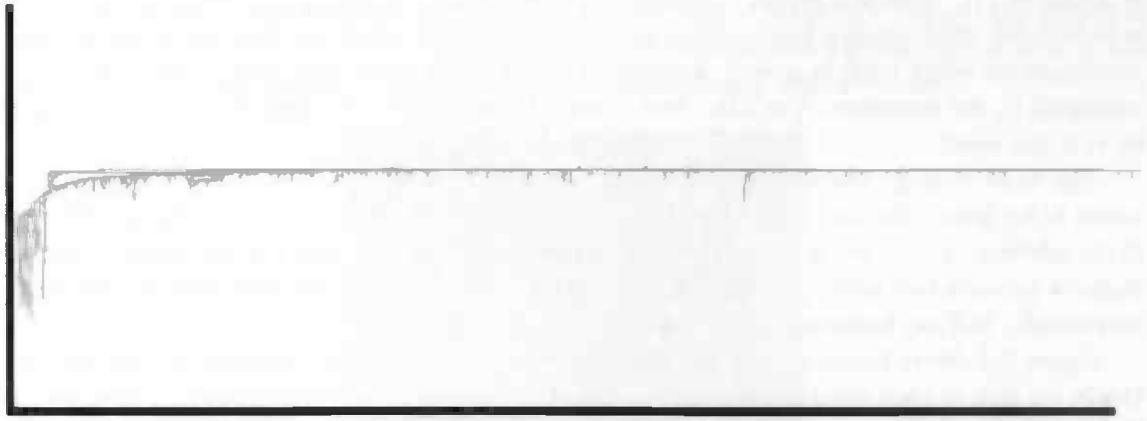


Figure 3.6: *Rendering of a Max-Tree data structure using the layered layout without using a pruning method.*

3.3.1 Number-of-children pruning

A brute force way to remove nodes is by assuming that all nodes with a limited amount of children are of less importance and can be removed from the screen representation. It is an effective way to remove a large number of nodes, but the actual structure of the tree might get lost in the process. Because the complete subtree is removed when a child has not enough children, the underlying structure will also disappear from screen. In figure 3.7 the Max-Tree data was pruned by removing all the nodes with less than 10 children.

3.3.2 Hidden-node pruning

This method works in conjunction with the Max-Tree filtering. When the Max-Tree is filtered, the data structure is not changed, but nodes are only collapsed by changing the current intensity of the node to the node it is collapsed onto. If a node is collapsed, it disappears, but all its children are added to the parent of the original node. By collapsing these nodes in the visualization, the number of rendered nodes is greatly reduced when the tree is filtered. Figure 3.8 shows the result of this method when the tree is filtered using the inertia attribute set to 2.1. This is a natural way to simplify the tree visualization, since it maps directly to the volume rendering and filtering of the tree. The image shows

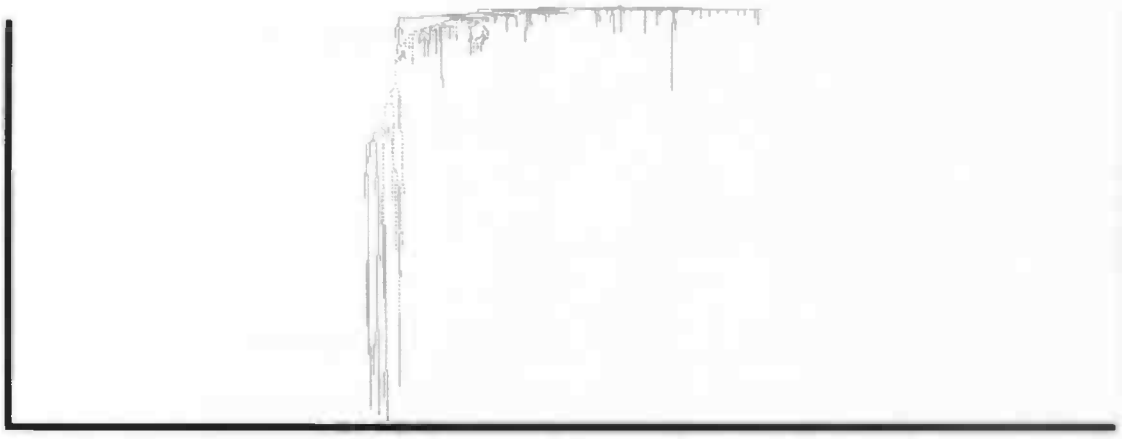


Figure 3.7: *Rendering of a Max-Tree data structure using the layered layout. All nodes in the Max-Tree with less than 10 children have been removed from the tree visualization.*

that there are lots of long strands of nodes with only one child, the next method filtered these out as a next step after the removal of the hidden nodes.

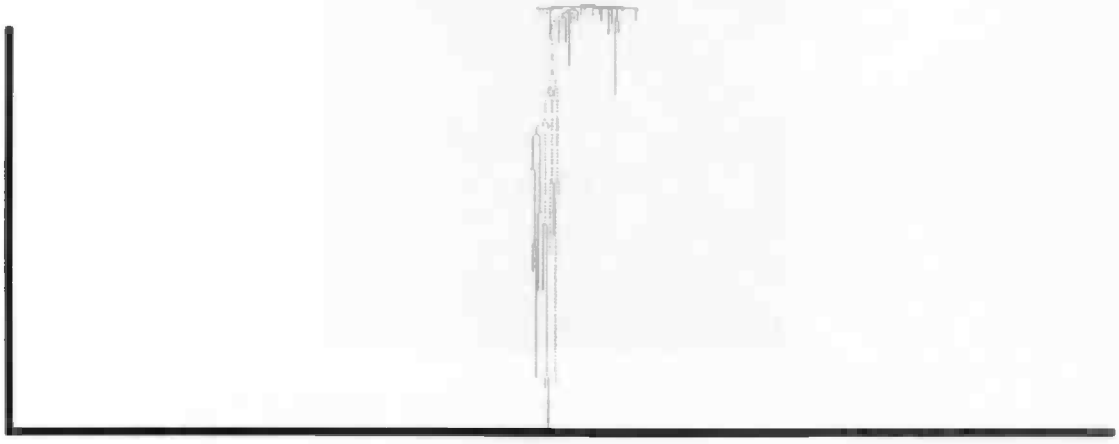


Figure 3.8: *Rendering of a Max-Tree data structure using the layered layout. The tree was first filtered with an inertia filtering set to 2.1, and all the collapsed nodes where removed from the visualization.*

3.3.3 Single child pruning

After all the hidden nodes are removed, a lot of long stands of nodes with only a single child remain. These strands do not give much extra information about the structure and can be collapsed into the topmost node. This method works best in conjunction with the removal of the hidden nodes. Figure 3.9 shows the result of this pruning. The resulting tree gives a nice overview of the structure without hiding too much information. A disadvantage of this and the previous method is that it only works on trees that were filtered already by the Max-Tree filter.

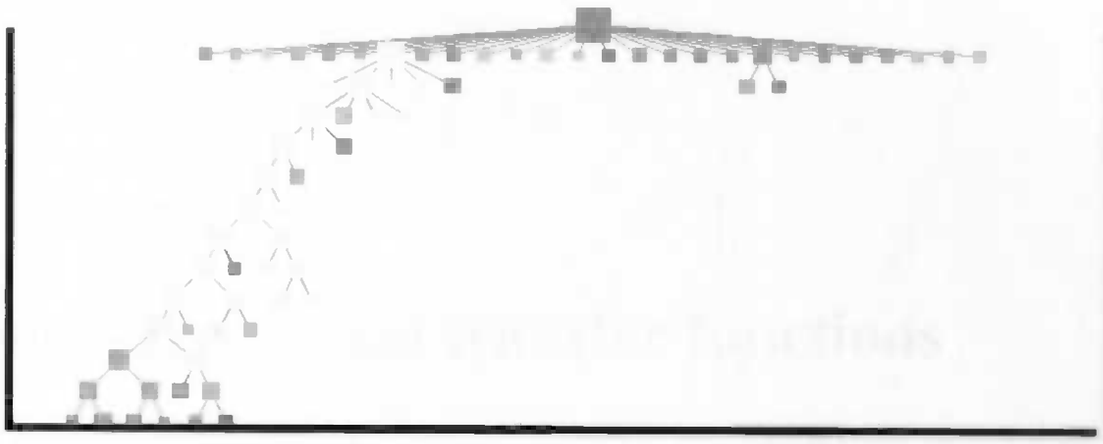


Figure 3.9: *Rendering of a Max-Tree data structure using the layered layout. The tree was first filtered with an inertia filtering set to 2.1. First the collapsed nodes where removed from the visualization, then all nodes with only one child were collapsed. Figure 3.10 shows the corresponding volume rendering.*

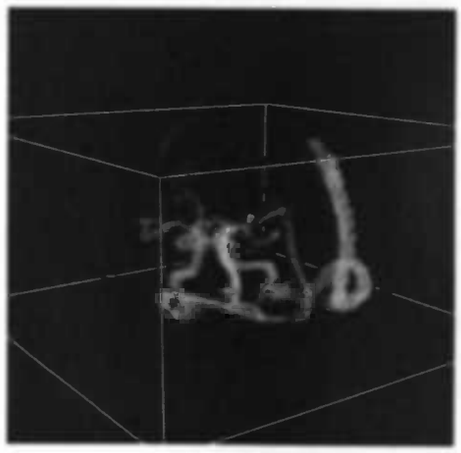


Figure 3.10: *Volume rendering corresponding to the Max-Tree in figure 3.9.*

3.4 Rendering

Rendering the tree layout is straightforward, since the layout algorithm determined the locations of all the nodes. A tree consists of nodes and edges. Every node is rendered as a small box and a parent-child relation is drawn as a line between the two nodes. To provide some extra information, the size of the node box is dependent on the number of voxels belonging to this node. The color of the box is determined by the transfer function when the node is part of the filtered Max-Tree and gray when the node is filtered out. It is also possible to select subtrees in the graph to get an idea of the connection between the Max-Tree datastructure and the volume rendering of the dataset, this is described in section 5.1. These selected nodes are drawn in red. Figure 3.11 shows the result of the rendering.

All rendering was done using OpenGL because it provides easy and fast rendering functionality. The number of nodes might increase a lot when using complex datasets. It is expected that software rendering might not be fast enough to keep the drawing in real time. OpenGL is fast when rendering

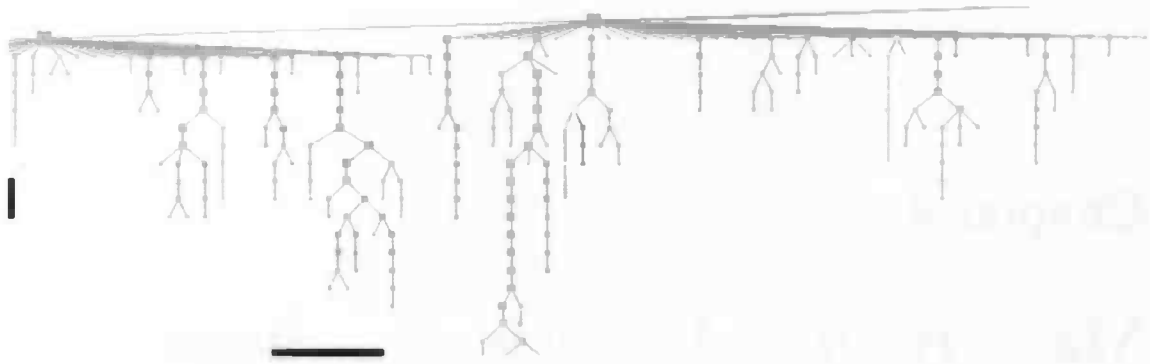


Figure 3.11: *Tree rendering using OpenGL. The boxes represent nodes of the Max-Tree, the size of the box is related to the number of voxels of the node. The color was determined by a transfer function described in section 4.*

small Max-Trees with around 6000 to 50,000 nodes. Trees with more nodes are also feasible, but eventually some extra optimizations might be needed.

Chapter 4

Max-Tree based transfer functions

To render objects with more information than just the intensity of a voxel one can use a transfer function. Transfer functions use one or more parameters to determine a color and opacity. A simple transfer function maps the intensity of a voxel to a gray level in the image. If the voxel also has other attributes such as direction or gradient, it is possible to combine these into a color by using a lookup table. The goal of this research is to create a transfer function that uses the connected-component information in the Max-Tree.

A problem in one-dimensional transfer functions is that features with the same properties can not be separated. In a CT image, bones can be separated from the tissue, but it is not possible to separate one bone from another [11]. To gain more flexibility transfer functions were improved to use first or higher order derivatives. These multi-dimensional transfer functions are highly flexible but this comes at the cost of complexity. By using the connectivity information in the Max-Tree we expect to be able to separate the objects, and maintain a simple user interface.

This chapter discusses a method for color splatting used for several transfer functions. One transfer function that uses the attribute of a peak component directly to choose a color and three that use changes in the attributes between different nodes to detect the different objects in the dataset and assign them separate colors.

4.1 Multi-color splatting

To support multi-color segmentation, a multi-color splatting method was implemented. Instead of using a one-color kernel function, the color of the kernel is now determined by a two-dimensional transfer function. One axis determines the color of the voxel, and the other determines the intensity. The kernels are rendered in a color buffer. Figure 4.1 shows the result. Before the image is rendered to the screen, it must first be scaled to fit the screen color depth. This scaling can simply be implemented by finding the minimum and maximum color value in each channel, and then mapping this range to 0-255.

4.2 Attribute-based coloring

In attribute-based coloring, a voxel is rendered with a color that is determined by the attribute value of the Max-Tree node to which it belongs. First the range of the attribute values in the Max-Tree nodes must be determined, then these values are mapped to a color. Depending on the kind of attribute,

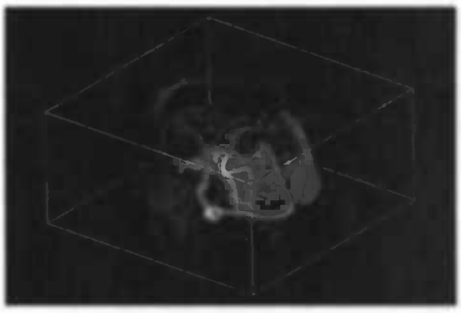


Figure 4.1: *The "LargeAngio" dataset, rendered with multi-color splatting.*

it might be necessary to use a nonlinear function for this mapping. Figure 4.2 shows an angiogram dataset where the inertia attribute is mapped to a color.

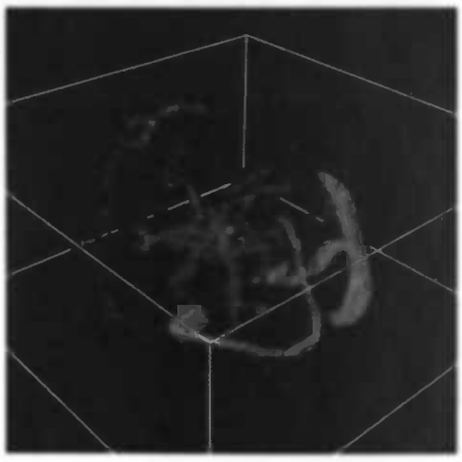


Figure 4.2: *Volume rendering of an angiogram dataset, where the transfer function uses the inertia attribute to determine the color. The large number of noise-like voxels that are part of peak components represent large inertia.*

The attribute coloring gives immediate feedback to the way an attribute filter works. But it also shows how non-intuitive the max-tree structure is when dealing with single nodes. Figure 4.3 gives an example. Components that are connected in the max-tree, seem disconnected in the attribute based coloring. Many of the volume renderings result in a noisy image because of the way Max-Trees are built. For a Max-Tree it is more natural to assign colors to peak-components (or Max-Tree subtrees) instead of individual nodes. This is what the following methods do.

Figure 4.3: *Example of peak component where attribute coloring will give a nonintuitive result. When using a volume attribute, the dark-gray pixels in the left image will have a high volume attribute value, since its peak component is large, and the bright pixels will get a low attribute value. In the right image, components that are connected seem disconnected in the resulting image or volume rendering.*

4.3 Segmentation based on volume attribute

A way to select peak-components in the Max-Tree is by using the volume attribute of the peak components. While traversing the tree, select the peak components with a volume smaller than a certain value ν. Since volume is decreasing from the root to the leaf nodes, several components in the tree will be selected. To all components smaller than a certain volume a random color is assigned. The method was implemented by a preorder tree traversal. When a node is reached with a volume-attribute value smaller than ν, a color is selected and is assigned to all the nodes in the subtree. Figure 4.4 shows an example of this selection method, rendered using multi-color splatting. This method gives a nice segmentation but can only select components of a fixed size, it can not handle components that are nested inside other components and only works on attributes that decrease while h increases like a volume attribute.

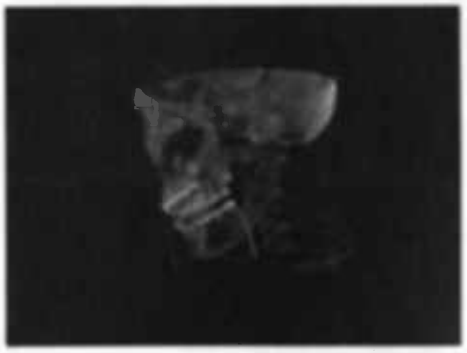


Figure 4.4: *The "fullHead" dataset, rendered with multi-color splatting, created with segmentation based on the volume attribute. All highlighted connected components have a volume < 900.*

4.4 Segmentation based on attribute change

This method uses the change between the attribute value of a node and its parent as a criterion to determine if a subtree represents a peak component of importance. If the change is larger than some value ν, the subtree gets a random color. If there is another node with a change larger than ν within the subtree, this subtree gets yet another color. The parameter ν can have values between zero and the largest difference in attribute value in the Max-Tree.

The method is implemented with a preorder traversal through the Max-Tree. If the difference between the attributes of a node and its parent in the filtered Max-Tree is larger than ν, a random color is generated and is assigned to the complete subtree. Because the algorithm assigns a random color to the important subtrees, this method is not consistent in the colors it assigns to a node in different runs. A slight change in parameter values can result in a complete different color set. This problem is addressed in the next method. Figure 4.5 shows a result of this coloring method on the "fullHead" dataset. All the spine bones are given a random color and the teeth stand out from the skull.

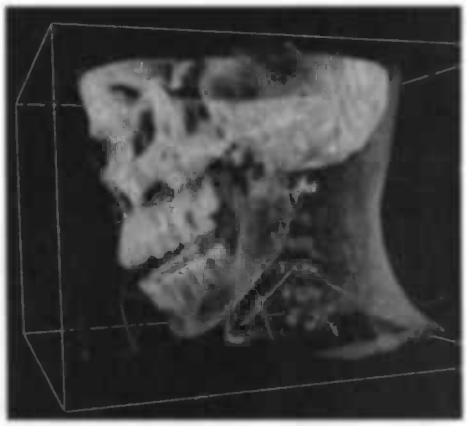


Figure 4.5: *The "fullHead" dataset filtered with the inertia attribute set to 2.2, colored using absolute attribute changes to determine the connected components. Subtrees/Peakcomponents with a node-parent attribute difference of 1.1 and larger are marked with a random color. The values were determined experimentally while looking for a segmentation of the vertebrae.*

4.5 Coloring based on relative attribute changes

4.5.1 Attribute signatures

Instead of using the attributes of the Max-Tree to determine the color, the Max-Tree node attributes can be analyzed to try to find important changes in the attribute values, and thereby finding potentially interesting connected components in the volume data. A similar idea was used in [8] as a way to build non-flat tree filters in component trees. An attribute signature is defined as a sequence of node attributes starting at a leaf node and ending in the root. Large changes in an attribute signature indicate changes within connected components, see figure 4.6.

Figure 4.6: *Two different attribute signatures. The x-axis shows the nodes from leaf to root drawn with the color of the node. The y-axis is the value of the attribute. The left signature has no relative changes larger than* 50%, *so every node has the same color. In the right signature there is a relative change larger than* 50%. *The color of the nodes has changed in this point.*

4.5.2 The method

Instead of using an absolute change in the attribute values between a node and its parent, the difference can be compared to the attribute range in its attribute signature(s). When the ratio between the difference of the minimal and maximal value in the attribute signature and the difference between the attributes of a node and its parent is larger than a value ν, the node is marked. In our definition, an attribute signature always starts with a leaf node. Therefore all the non-leaf nodes are part of multiple attribute signatures. If a node is marked in all signatures, this node is part of a structural change in the volume and its subtree gets a random. When all important nodes are registered, the final colors are

assigned to the nodes. To overcome the problem of changing colors every time the method is started, the randomizer is initialized by a constant seed, and all Max-Tree nodes are given a random color. A preorder traversal through the Max-Tree is used to determine the final colors of the nodes. When a node is marked it can keep its color, when it is not marked the node is given the same color as its parent. Figure 4.7 shows the fullHead dataset segmented by this method.

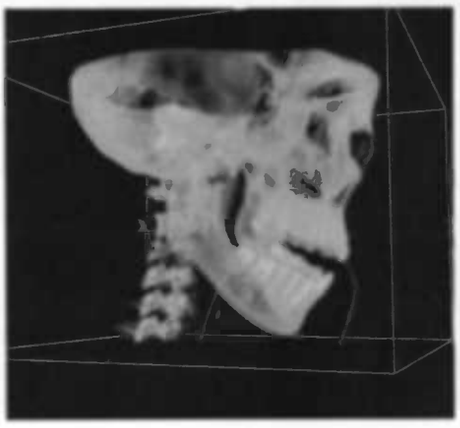


Figure 4.7: *The "fullHead" dataset with components decided by the relative attribute change set to 30 and with graylevels <1800 not shown to remove the background noise and skin.*

4.6 Gradient-based opacity

This technique is not based on the Max-Tree but is using the gradient of the voxels to determine the opacity of the voxels. Voxels with a higher gradient are drawn more opaque than the voxels with a low gradient. The result is a volume rendering that highlights the parts of the volume with high intensity variations such as edges. Figure 4.8 shows a head rendered using gradient based opacity.

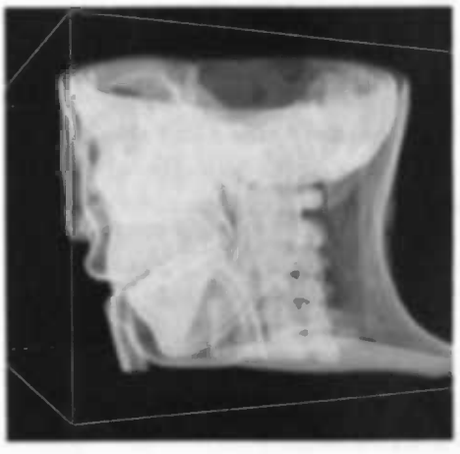


Figure 4.8: *"fullHead" dataset rendered with gradient based opacity.*

Chapter 5

Interaction

5.1 User interaction

The window that displays the data structure visualization of the Max-Tree supports user interaction. Because the tree can be quite large without filtering, it was necessary to implement zooming and panning. For example, the "angiolarge" dataset Max-Tree has over 350.000 nodes. With the mouse wheel the user can zoom in and out and with the middle mouse button the graph can be moved to view other parts of it.

Another part of the interaction with the tree is the selection of subtrees. Selecting a subtree is done by shift-clicking on a node with the left mouse button. When selecting a subtree, all the nodes that are collapsed into it by any filtering are also selected. The selected nodes are rendered in red. The volume rendering window also shows the selected nodes by rendering the voxels belonging to the selected nodes in red. With a button on the user interface, a color can be assigned to the selection.

Every voxel is rendered in the color of the max-tree node it belongs to. Figure 5.1 shows how this is displayed. The chapter on transfer functions gives more background into how the colors are determined. By selecting sub-trees and assigning colors to them a manual selection of the interesting parts of the sub-tree can be made. The selection interface makes it possible to quickly learn about the structure of the volume data and also about the relation between the volume data and the Max-Tree.

5.2 Connected component selection in volume projection

In addition to selection of nodes in the Max-Tree data structure window, a method has been developed to select connected components directly in the projection of the volume. In the data structure rendering, there is no obvious relation between the position of a node and the position of the voxels belonging to that node in the volume visualization. Selecting connected components or subtrees, goes by trial and error. Selecting a node directly from the projection is far more intuitive, but not without problems. The selection of a position on a two-dimensional projection of a three-dimensional dataset can be done by the selection of many voxels on a line parallel to the viewing direction. These voxels do most likely not belong to the same connected component. Components can be enclosed or overlap each other. Furthermore, many voxels are part of only a small connected component, much smaller than one may want to select.

The problem of overlapping and enclosing voxels was addressed by using the first non-transparent voxel for the selection, but many other schemes could be used. After the voxel is determined, the Max-Tree is traced to the root until a node is detected that is also visible in the data-structure rendering. Of

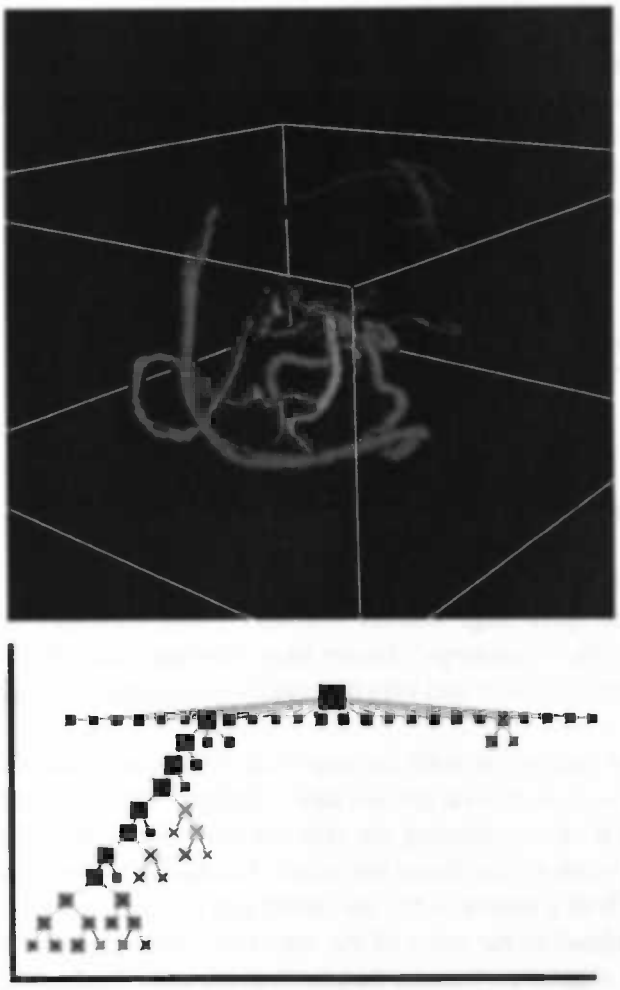


Figure 5.1: *Voxels belonging to the selected nodes are the ones displayed with a color and a cross in it.*

this node the complete subtree is selected. This way somewhat larger components are selected, and they can be easily manipulated in the data-structure rendering.

5.2.1 Implementation

The selection in the volume projection uses three steps. First the 2D coordinate of the selection is transformed to projection space, resulting in a line. This reverse projection was performed with the OpenGL *gluUnproject()* function. The function computes the 3D position at a given distance from the camera. When using this function twice, with a small distance and a large distance, it is possible to get two coordinates on the selection line. The second step finds the intersections between the line and the volume data cube. this was implemented with a 3D version of the Cohen-Sutherland line segment clipping method [3]. Finally, using the two intersection points, the line is traced through the volume data to find a non-transparent voxel. The first non-transparent voxel is returned.

5.3 Graphical user interface

A basic user interface was built to control the filtering, the transfer function, the tree graph and the selection. The interface was built using FLTK and the Fluid user interface builder. First an extension to an existing user interface was written, but the final result is an interface specifically written to reflect the options of the data structure rendering and the transfer functions. Figure 5.2 shows a screenshot of the current interface. The following sections will go into more detail about the different parts of it.

5.3.1 Max-Tree filter settings

This section controls the attribute filter functionality. In the attribute drop-down list, one can choose between two attributes, an inertia attribute and a volume attribute. Changing the attribute causes the application to recompute the attributes for all Max-Tree nodes, so it might take a few seconds before completion. With the filter slider, the value of the attribute can be changed. The range of the filter is the same as the range of attributes in the Max-Tree.

5.3.2 Volume transfer function settings

In this part of the screen the transfer function method can be changed and modified. The standard setting is the *relative attribute change* method described in section 4. This method has four parameters, the relative attribute difference to adjust the ratio of difference before a new segmentation is made. The second is the difference in gray-values between a parent and a node before a segmentation is made. The opacity parameter changes the opacity of all the voxels that are not assigned another color by selection. Finally, the clipping parameter adjusts the visible gray-levels. All voxels with a gray level lower than this clipping value are rendered completely transparent.

5.3.3 Graph window settings

To save some layout computations and allow a faster response in filtering, the data structure window is closed on startup. In this settings section, the data structure window can be opened, and some layout settings can be changed. By changing the pruning factor, nodes can be removed from the data structure rendering to get a better overview. The nodes that have a smaller number of voxels than the pruning factor are removed. The *hide filtered nodes* flag can show or hide the nodes that were removed by the Max-Tree filtering. The *collapse hidden* flag allows the filtered nodes to collapse to show the filtered Max-Tree. The *collapse one child* flag allows long strands of nodes with only one child to be removed from the data structure rendering of the filtered Max-Tree.

5.3.4 Attribute signature graph

To analyze the attribute signature from node to the root, this window gives a quick view of the signature when a node is clicked in the graph window. The signature also shows the colors of the nodes, to check the result of the transfer functions. The attribute values of the signature can be saved to a text file named *roottrace.txt*, which can be opened in gnuplot.

5.3.5 Selections

The selection part of the user interface allows the user to give selections made in the graph or volume window a color to manually segment the volume. Four sliders allow the user to choose a color and

an opacity. This color-opacity combination is assigned to the current selection. This way interesting parts in the volume can be highlighted, or non-interesting parts of the volume can be suppressed or made completely transparent so that they do not disturb other parts of the image. There is a button to select all the non-segmented parts in the volume. When a user has segmented all the interesting parts, the other parts of the volume can be selected and made completely transparent. Finally there is a button to remove the current segmentation, and go back to a volume only colored by the transfer function.

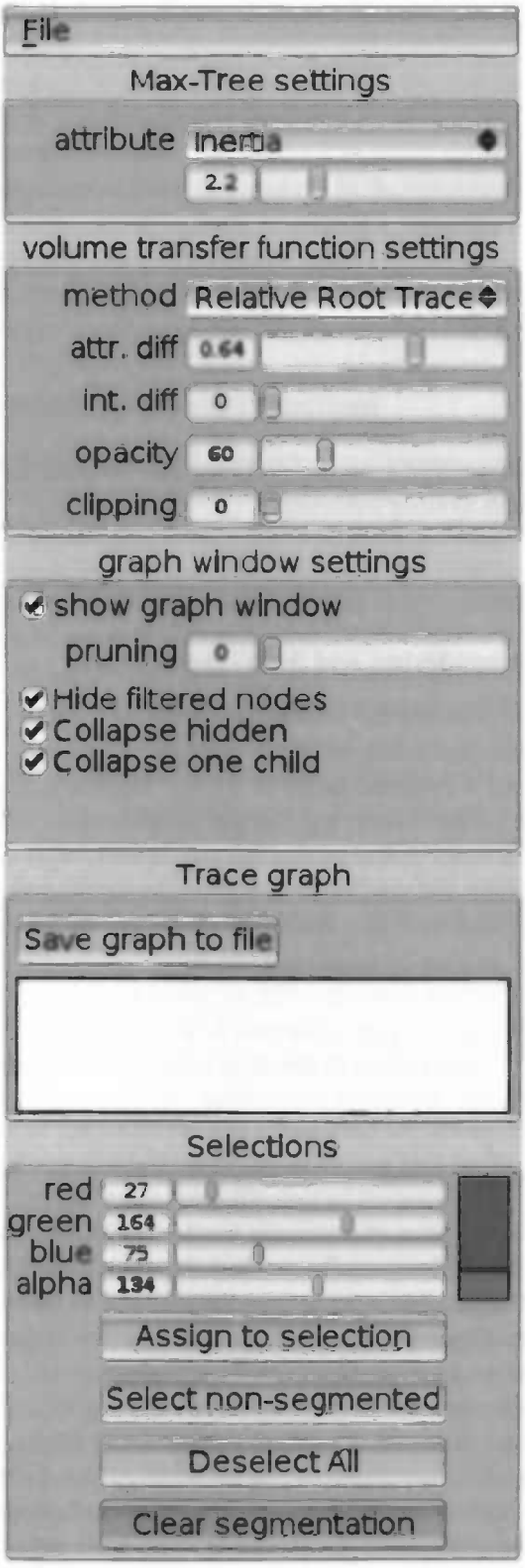


Figure 5.2: *Graphical user interface used to control the layout of the data structure rendering.*

Chapter 6

Experiments

This chapter will give examples of how the program can be used to explore volumetric data and compare the transfer function to other methods.

6.1 Examples

6.1.1 Relation volume rendering and Max-Tree graph

To demonstrate the relation between the volume rendering of a dataset and its Max-Tree graph, this section will give a few examples using the angio dataset. The angio dataset is an angiogram with dimensions 128x128x62 and a dynamic range of 8 bits. Figures 6.1, 6.2 and 6.3 show an array of images that show both the volume rendering and the data structure rendering of the dataset. Figure 6.1(a) shows how the result looks just after loading the dataset, without any filtering. The dataset is rendered somewhat transparent and the data structure rendering is completely zoomed out to show the entire tree. The part of the data structure rendering that is on the far left of the tree seems the most interesting part since there is a lot of branching. In figure 6.1(b) the left part of tree is shown. By selecting the tree, figure 6.1(c), one can see what elements in the volume data are related to this part of the data structure. In this case this part of the tree is related to many elements in the volume dataset. Selecting smaller parts of this tree results in selection of smaller elements, figure 6.2(a), 6.2(b), 6.2(c), 6.3(a) and 6.3(b).

6.1.2 Segmentation of vertebrae

In this example we have a CT scan of a human head. The dimensions are 256x256x176 and the dynamic range spans 12 bits. The goal is to select the vertebrae. Selection of these vertebrae is something that would not be possible using a one-dimensional transfer function, the vertebrae all are in the same intensity range. Figures 6.4 and 6.5 show the various steps. When the program is started with this dataset, the result is a large purple blob of voxels, figure 6.4(a). This is because the renderer initially renders all the non-zero voxels with the same opacity value, even the low intensity noise voxels. In this case the noise is removed by applying an inertia filter set to a value of 1.8. This filter will remove all the spherical shaped elements, figure 6.4(b). The result is an image where the noise and large parts of the skin are removed. The transfer function attribute difference is adjusted to a value of 0.20, figure 6.4(c). Various shapes in the skull have now been segmented. The teeth are separated from the skull, the vertebrae are separated and the jawbone is separated from the skull. Figure 6.4(d)

gives an overview of the data structure. The nodes in the data structure rendering share the color of the nodes in the volume rendering.

The vertebrae can be selected in the volume window, or by trial and error in the data structure window. The volume window can be used as first step in seeing what nodes are connected to a component, and the data structure window can be used to select the nodes more accurately. Figure 6.4(e) and 6.4(f) show the selection of the first vertebra. In the graphical user interface a color can be assigned to the selected element. The procedure of selecting vertebrae and assigning colors to them can be repeated for the others, see figure 6.5(a) and 6.5(b).

Figure 6.5(c) shows the dataset after making the non-segmented parts of the image completely transparent. The result is a segmented image that can be exported for further use.

6.1.3 Comparison to spatialized transfer functions

The spatialized transfer function is a method developed by Roettger, Bauer and Stamminger [11] that combines spatial information with a histogram to automatically setup a transfer function. In a spatialized transfer function spatial information is used to derive a color. The method allows manipulation of the transfer function by changing the colors in a histogram of the volume. This gives the possibility to interactively change the transfer function and highlight the various parts of the volume. This method is interesting for comparison because it also uses spatial information in the coloring of the volume data. The way it uses the spatial information is very different from our method.

The opacity in the spatialized transfer function is determined by the gradient. A separation of features is made using the color channel of the transfer function. A 2D histogram $H(s,t)$ is made of the volume dataset with the normalized scalar value s on the x-axis and the normalized gradient t on the y-axis. Two properties are computed for every histogram entry using the voxels that belong to that entry. The barycenter of the voxels $b(s,t) = \frac{1}{n}\sum_{i=1}^{n} p_i(s,t)$, where $p_i(s,t)$ is the normalized position and n is the number of voxels belonging to an entry. And the spatial variance of the voxels $v(s,t) = \frac{1}{n}\sum_{i=1}^{n} \parallel p_i(s,t) - b(s,t) \parallel$. The histogram entries are grouped in tuples T_i based on a spatial correspondence measure $N(T,T_0) = \parallel b(T) - b(T_0) \parallel + \mid v(T_0) - v(T) \mid$. The method starts with a reference tuple T with the highest histogram count and is repeated until the classification is complete. Every tuple is assigned a random RGB value.

The method is compared to our method using the tooth dataset. Unfortunately we were unable to use other datasets because the given test application used only these for demonstration purposes. Figure 6.6 shows renderings of the tooth dataset using both the spatialized transfer function and the Max-Tree based transfer function. Both transfer functions are able to discriminate various features in the tooth. The top of the tooth is different from the root of the tooth. The spatialized transfer function allows manipulation through a two-dimensional spatial representation of the volume. With feedback in the volume window it is possible to find the relations between the volume and the spatial view. The Max-Tree based transfer function uses a data structure rendering of the Max-Tree to manipulate the volume. In volume data with large changes in the connected components through the gray levels, this gives a nicely branched tree. In a dataset where the changes are more like a gradual change in gray level, such as the tooth dataset, it just gives a very deep tree without many branches. In this case the spatial transfer function will give a much better result for the transfer function.

The tooth dataset is difficult to handle for Max-Tree transfer functions. Inside is a low density or hollow component. In a Max-Tree these low density or hollow of components are part of the background, even though they can be a real part of the dataset. This makes it impossible to separate these components. A Min-Tree would solve this problem, but only at the cost of the maximum components.

The spatialized transfer function method does all the coloring using a two-dimensional histogram,

the Max-Tree based transfer function has additional functionality built in that allows picking in the volume data to select component in the volume and assign a color to them. This manipulation is more intuitive, but comes with the problem of objects that hide others in a three-dimensional view.

Spatialized transfer function have an interesting way of using spatial information in a transfer function and give more flexibility in the types of datasets that it can easily operate on. Max-tree based transfer functions have the advantage of using attribute information when creating a segmentation. Due to the limited functionality of the spatialized transfer function demo, the comparison between the two transfer functions was not extensive enough to draw conclusions.

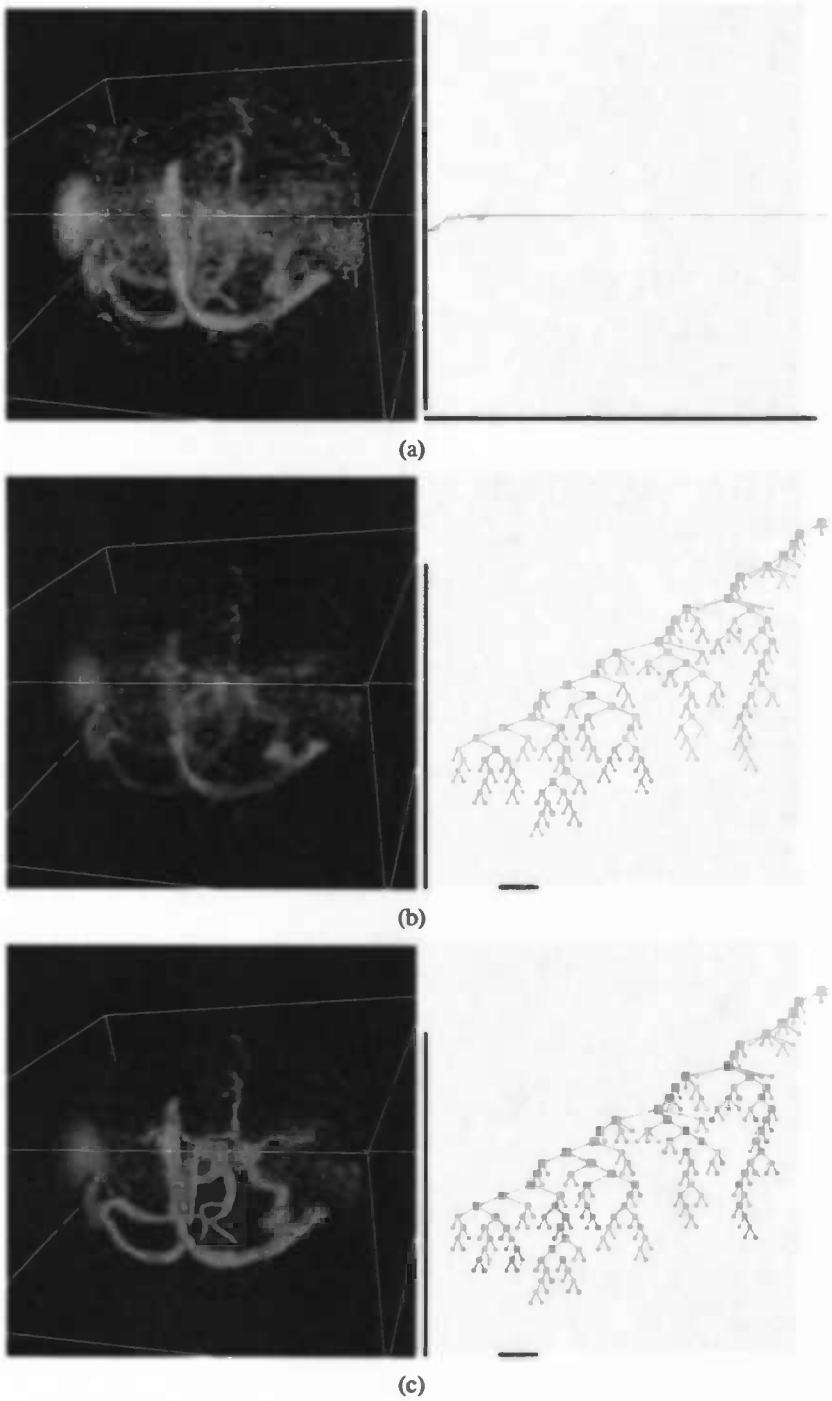


Figure 6.1: *Images showing the relation between renderings of the volume dataset and the tree-graph of the Max-Tree, or parts of it.*

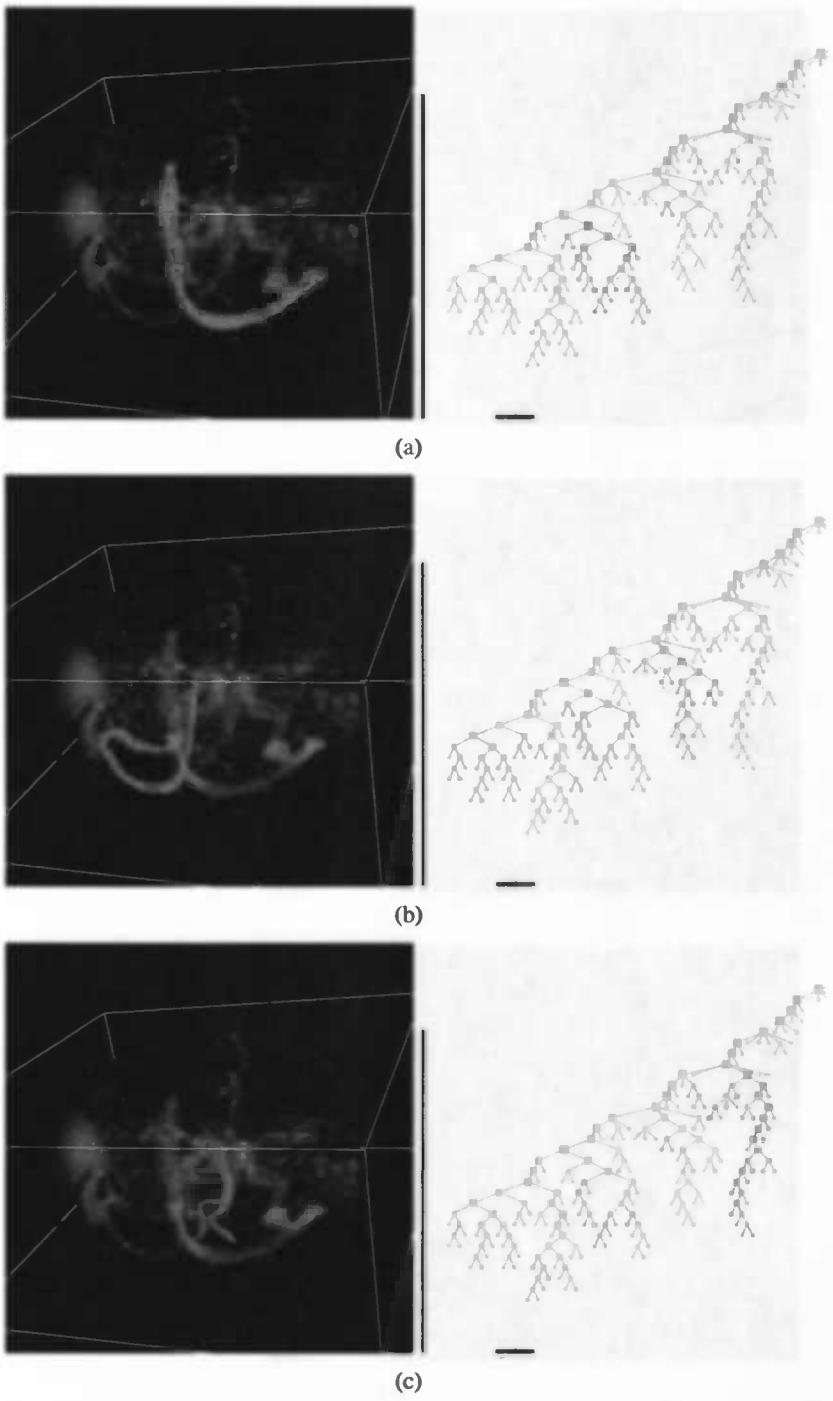
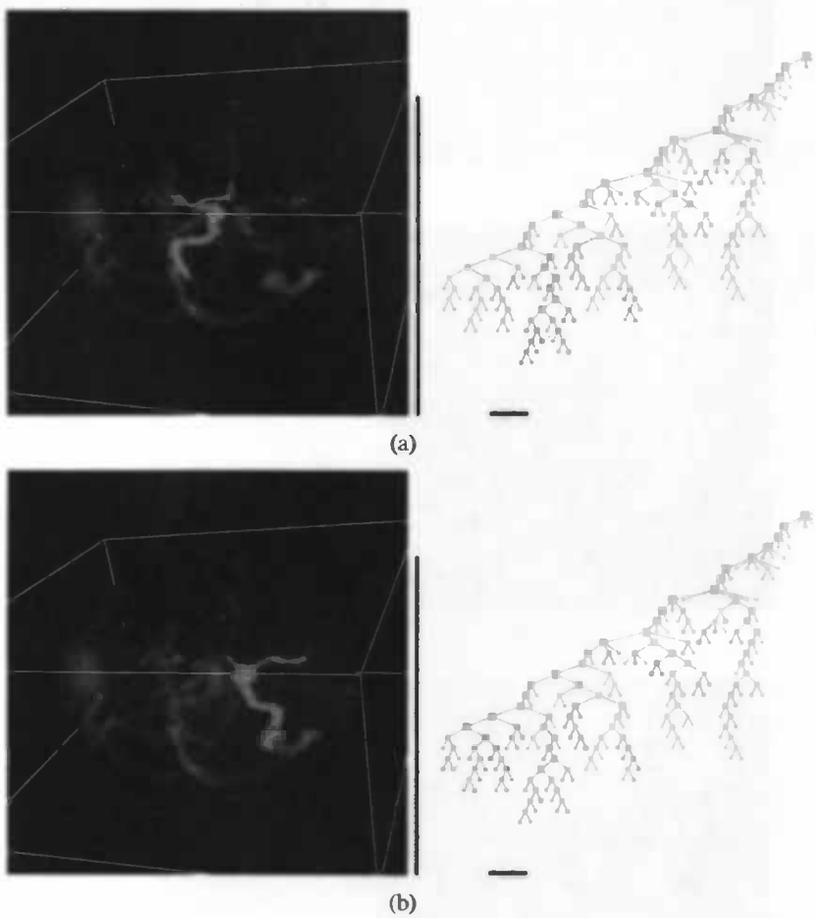


Figure 6.2: *Images showing the relation between renderings of the volume dataset and the tree-graph of the Max-Tree, or parts of it.*



(a)

(b)

Figure 6.3: *Images showing the relation between renderings of the volume dataset and the tree-graph of the Max-Tree, or parts of it.*

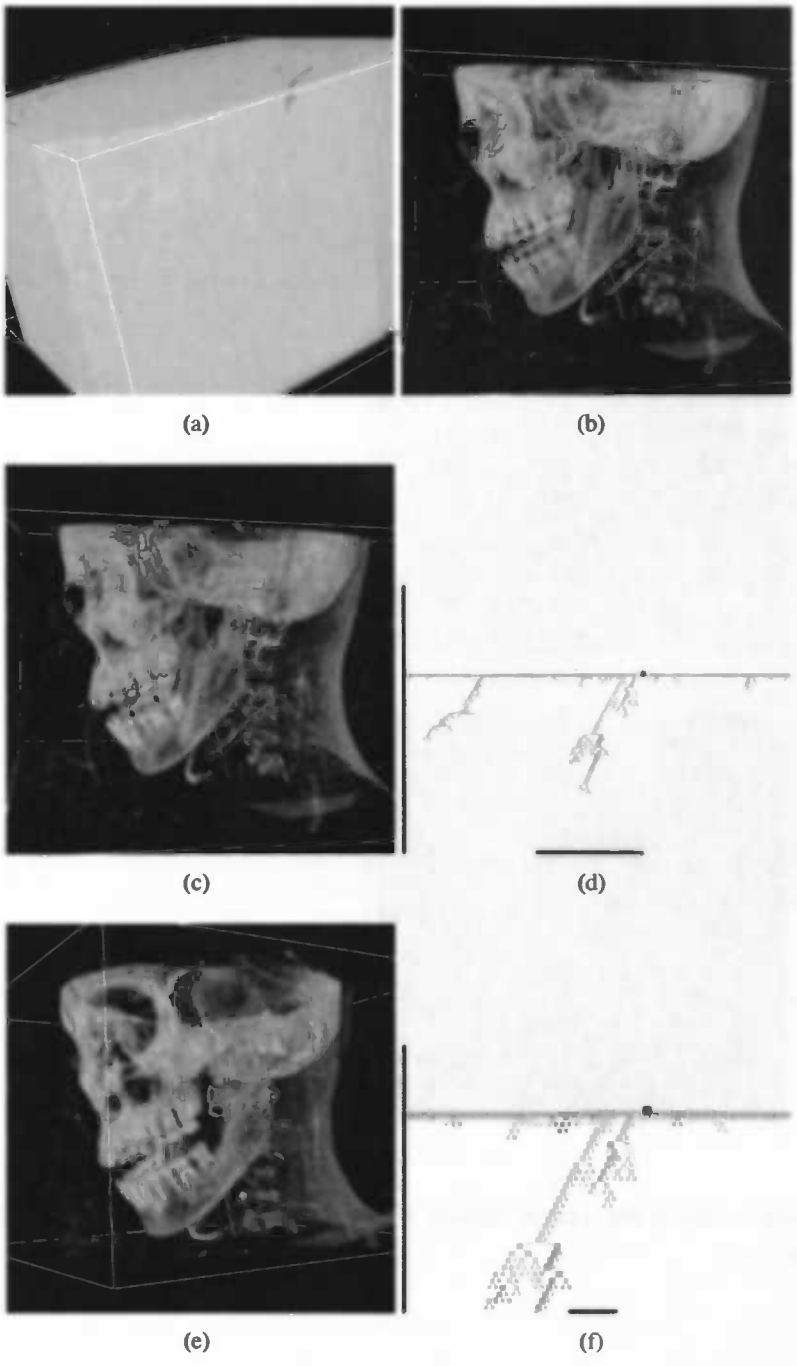


Figure 6.4: *Various steps in segmenting the vertebrae in the the fullHead dataset; (a) the dataset without any filtering; (b) the segmented dataset using an inertia filter with a value of* 1.8*; (c), (d) the dataset and the Max-Tree with a relative attribute change of* 20%*; (e),(f) the dataset with one vertebra selected.*

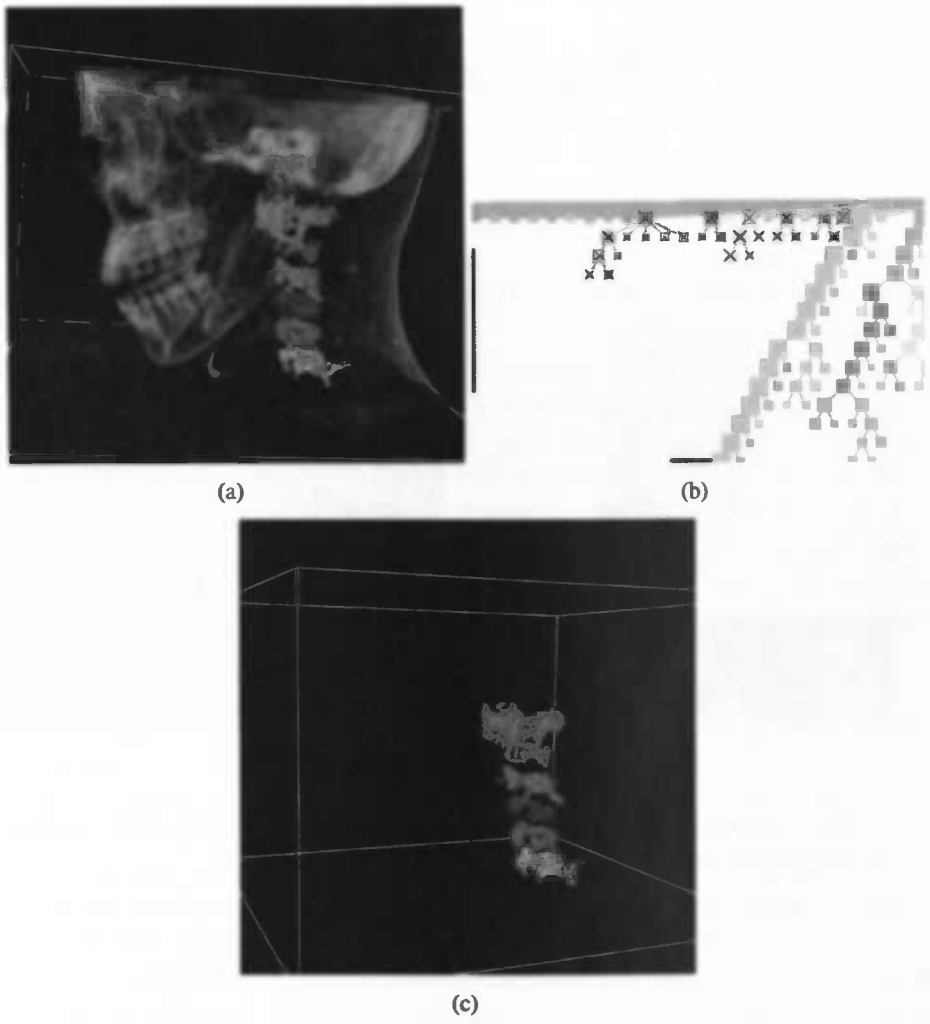
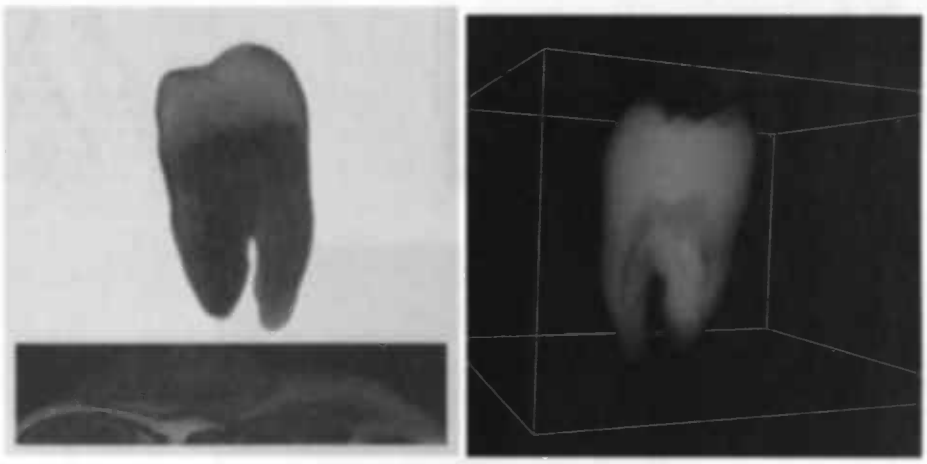


Figure 6.5: *Various steps in segmenting the vertebrae in the the fullHead dataset; (a), (b) the dataset and a part of the Max-Tree with six vertebrae selected; (c) the same selected vertebrae but with the rest of the data made transparent.*



(a) (b)

Figure 6.6: *(a) Tooth dataset colored using the spatial transfer function; (b) Tooth dataset rendered using the Max-Tree based transfer function.*

Chapter 7

Discussion and future work

In this thesis methods were presented to explore, visualize and manipulate volume data in novel ways by combining the Max-Tree data structure with transfer functions and manual selections.

A method to layout a binary tree was extended to a layout function for trees with arbitrary structure. Some pruning methods were developed to deal with the large number of nodes in the Max-Tree data structure. The rendering of the data structure was extended with manipulation tools to explore the data structure and see feedback in the volume visualization. The volume rendering was also made interactive by allowing selection of elements inside the volume data, and giving feedback in the data structure rendering. Finally a few new methods were developed to use the structure that is available in the Max-Tree in a transfer function to segment the various elements in the dataset.

A returning problem with working with the data structure rendering of the Max-Tree is the large number of nodes and the width of the tree. Some of this was addressed by implementing pruning methods. But even after pruning there are still many nodes that might be removed to reduce the visual clutter. In addition to selecting the interesting nodes, a function can be added to remove nodes from the data structure visualisation. Right now only selection of subtrees is possible, but for this kind of interaction a different way for multi-node selection might be interesting, selection of rectangular regions for example.

Nodes in the data structure visualization sometimes represent more than one Max-Tree node. The rendering of this node does not give any information about the nodes that are collapsed into it. A problem occurs when one of the hidden nodes is part of a selection of a segmentation set, but is not visible on screen. By giving feedback in the rendering of the node on the collapsed nodes and adding the possibility to fold out the collapsed nodes, this problem has been addressed.

The described transfer function makes a segmentation of the connected components found using the Max-Tree and the attributes. To discriminate between the elements, each element gets its own color. The gray value of the voxels is currently not used to determine the output color of the voxel. The logic behind this is that components with a lower intensity are not necessarily less important in the final image, although in some cases it might make the resulting image more clear when one also uses the intensity value in some way.

Right now the transfer function segments all the objects that accommodate certain features. Another idea is to only segment the most important n objects in the volume, instead of all, although the result might be the same as just adjusting the attribute values in a different way. The current transfer functions only judges by the trace of a single attribute. It might be very interesting to try to use multiple attributes for a decision. Objects with similar features could be given the same color.

A few known techniques could make the quality of the volume rendering a bit better. The edges

between the different segments in the segmented volume are not always clear right now. A method that has been used a lot in other transfer function algorithms is by determining the opacity of a voxel-based on the gradient. This method is implemented within our algorithm, but does not highlight the segment edges. The rendered volumes still look flat, no lighting algorithm is applied to them. Lighting could improve the sense of depth. A pre-integration technique can make the result of the GPU computed volume rendering better [6], especially in areas with thin opaque parts of volume, which is typical when highlighted edges and lighting.

Bibliography

[1] fltk.org. Website about the Fast Light Toolkit (FLTK), a C++ graphical userinterface toolkit.

[2] H. Carr and J. Snoeyink. Path seeds and flexible isosurfaces using topology for exploratory visualization. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 49–58, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.

[4] G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice hall, 1999.

[5] T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–2001, 1992.

[6] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16, New York, NY, USA, 2001. ACM Press.

[7] H. J. A. M. Heijmans. Connected morphological operators for binary images. *Computer Vision and Image Understanding: CVIU*, 73(1):99–120, 1999.

[8] R. Jones. Connected filtering and segmentation using component trees. *Comput. Vis. Image Underst.*, 75(3):215–228, 1999.

[9] A. E. Kaufman. Volume visualization. *ACM Comput. Surv.*, 28(1):165–167, 1996.

[10] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[11] S. Roettger, M. Bauer, and M. Stamminger. Spatialized transfer functions. 2005.

[12] 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.

[13] P. Salembier and J. Serra. Flat zones filtering, connected operators, and filters by reconstruction, August 1995.

[14] M. A. Westenberg and J. B. T. M. Roerdink. X-ray volume rendering through two-stage splatting. *Machine Graphics and Vision*, 9(1/2):307–314, 2000.

[15] M. A. Westenberg, J. B. T. M. Roerdink, and M. H. F. Wilkinson. Nonlinear volumetric filtering and interactive visualization using the max-tree representation. 2006.

[16] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 169–177, New York, NY, USA, 1998. ACM Press.

[17] M. H. F. Wilkinson and M. A. Westenberg. Shape preserving filament enhancement filtering. *Lecture Notes in Computer Science*, 2208:770+, 2001.