university of groningen

faculty of mathematics and natural sciences

Morphological attribute filtering and visualization with 3-D α-partition trees

Bachelor's thesis

June 2011

Student: Robert Jan Kattouw

Primary supervisor: Dr. Michael H. F. Wilkinson

Secondary supervisor: Arnold Meijster

Contents

Chapter 1

Introduction

The α-partition tree, also known as salience tree [4], is a data structure that represents an image at multiple levels of detail. It is essentially an α-partition hierarchy, as the nodes in the tree represent α-connected components [1]. It was originally designed and used for object recognition in 2-D images. In this thesis, I describe my work to extend the α-partition tree to operate on 3-D volumes and use it for morphological attribute filtering and visualization. The α-partition tree is a very new data structure that has only recently been proposed, and no one had attempted to use it in 3-D before.

The code I have developed is integrated with *mtrender*, a small demonstration program originially written to demonstrate attribute filtering and visualization using the max-tree. It is a simplified version of *mtdemo*, which is freely available for teaching and research purposes at `http://www.cs.rug.nl/~michael/MTdemo`. This integration allowed existing interactive visualization techniques used with the max-tree to be reused with the salience tree with relatively little effort. The implementation of morphological attributes and more mundane things such as file formats for volumetric datasets could also be reused. I also had access to the source code of Smit's original 2-D implementation of the α-partition tree, which included a simple filtering implementation of direct filtering with the volume attribute.

The remainder of this thesis is organized as follows. The basic algorithm of the binary partition tree and the α-partition tree is discussed in chapter 2. Chapter 3 introduces the max-tree as the currently preferred data structure for filtering 3-D data and discusses the potential of the α-partition tree in 3-D volumetric attribute filtering. A number of significant implementation details, such as filtering strategies and edge strength measures, are discussed in chapter 4. Chapter 5 discusses visualization of filtering results. In chapter 6, the α-partition tree is compared with the max-tree in terms of performance and result quality. Finally, suggestions for future work are discussed in chapter 7.

Chapter 2

The α-partition tree in 2-D

The α-partition tree is a variant of the binary partition tree (BPT), a filtering method used in 2-D image processing. The BPT segments an image on multiple levels, creating increasingly coarse segmentations [2]. The BPT algorithm divides the image in regions, with each pixel being alone in its own small region initially. It then iteratively merges neighboring regions, until only one region is left. These merges are then represented as a binary tree, with the leaf nodes representing individual pixels, the internal nodes representing multi-pixel regions, and each node's child nodes representing the subregions that were merged to form that region. The root node represents the entire image. Each level of the tree describes a segmentation of the image, with the levels closer to the root describing coarser segmentations, because they contain fewer and larger regions. This process is illustrated in figure 2.1

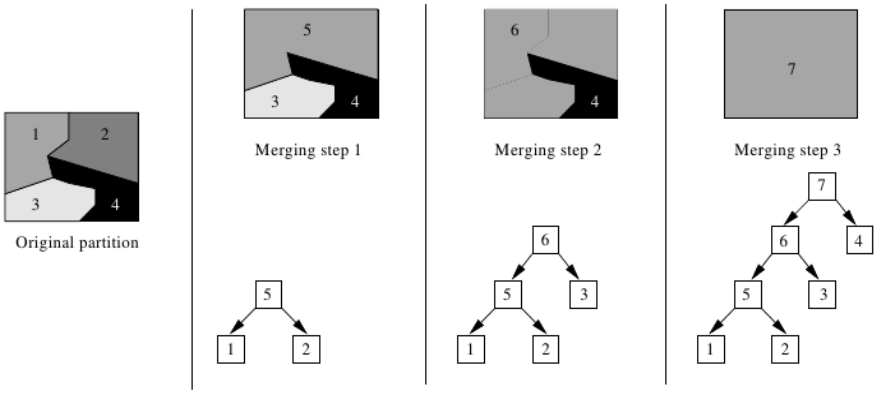


Figure 2.1: The merging of regions in a BPT. This image was taken from [2].

2.1 Distance measure and merging order

The segmentations represented in the BPT are only meaningful if the merging order is chosen carefully. Two very similar regions should be merged early, such

that the combined region appears near the bottom of the tree, while dissimilar regions should be merged later. This results in a tree where the smaller regions farther away from the root are more homogeneous.

In order to merge regions in order of decreasing similarity, a similarity measure needs to be defined quantitatively. For the BPT and the α-partition tree, a quantitative measure of dissimilarity between pixels is used, also known as the distance or salience between two pixels. For gray-scale images, this distance can be trivially defined as the absolute value of the difference of the pixels' gray values. For color images and other kinds of multi-channel data, more interesting measures are available, such as Euclidiean distance, Minkowski distance, or, in the case of colors, a distance in another color space that more closely matches the human perception of dissimilarity between colors. Based on a distance measure defined between pixels, the distance between two regions can be computed by taking the average or median value of the pixels in each component, and computing the pixel distance between those values.

The BPT algorithm starts by filling a priority queue with all links between neighboring pixels in the image. For a 2-D image, 4-connectivity or 8-connectivity is typically used. The weight of each link is the distance between the two regions it connects. Initially, all regions consist of a single pixel, so this distance is defined by the pixel distance mentioned previously.

The links in the queue are then processed in increasing order of distance. When a link is processed, the regions at either end of the link are merged, if that has not already happened. However, when two regions are merged into one, the priority values of the links connected to these regions will change. To maintain the order in the priority queue, these links will have to be reinserted into the queue. This is an expensive operation and needs to be done roughly $O(\log n)$ times for each link, where n is the number of pixels in the image.

Because the constantly changing link values slow down the BPT algorithm immensely, the α-partition tree was developed as a simpler, faster alternative [4]. In the α-partition tree algorithm, the merging order is fixed, and link values are not recomputed when regions are merged. Instead, a link's weight is defined as the distance between the pixels, as opposed to regions, at either end. This value does not change as regions are merged. Because links are not reinserted into the priority queue, the α-partition tree algorithm is much faster than the original BPT algorithm.

Chapter 3

Using α-partition trees in 3-D

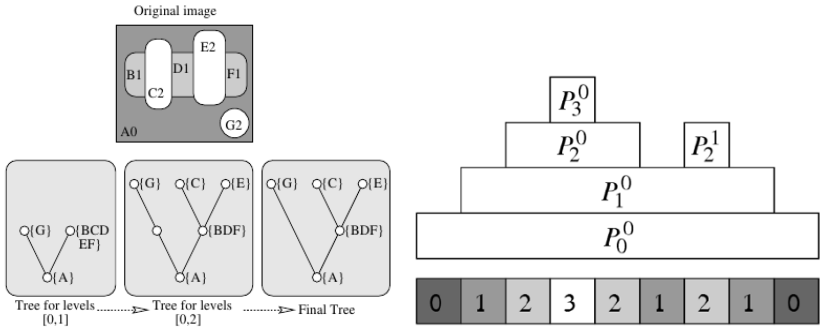
The main focus of my thesis work was to port the α-partition tree algorithm to 3-D and use it for volumetric attribute filtering. The α-partition tree was designed for object recognition in 2-D, but nobody had attempted to use it to filter volumetric data. The α-partition tree algorithm is not bound to any number of dimensions: it will work with any kind of data in any number of dimensions as long as a connectivity and a distance measure are defined.

3.1 Max-tree

A popular tree structure for filtering gray-value volumetric data is the max-tree [3]. Like the BPT and the α-partition tree, the max-tree segments the image on multiple levels of granularity, and has a root node representing the entire image. A node can have more than two children, and each child represents a sub-region in which all voxels have a certain minimum value that is higher than the lowest value in the parent region. The minimum gray value in a node's region is also referred to simply as that node's value. Effectively, this segments the image at each gray value, with finer segmentation for higher gray values, as illustrated in figure 3.1. This structure is optimized for images and volumes with bright structures on a dark background: the background regions will be near the root of the tree, and the regions of interest will be near the leaves.

3.1.1 Advantages of the max-tree

Volumes with bright structures on a dark background are common in medical imaging. The max-tree is designed specifically to handle these kinds of volumes. Because it does not require a minimum amount of contrast between foreground and background, but simply that the former be brighter than the latter, it performs very well on low-contrast volumes. The max-tree has also been extensively optimized for speed: after initial tree construction and volume rendering, re-filtering the same volume with a different threshold value and re-rendering it takes less than 100ms on a typical machine for a 256x256x256 volume. This means the user can adjust the filtering parameters and see the results in real time.



(a) 2-D input image and tree diagram of the corresponding max-tree. Image from [3].

(b) 1-D input signal and peak component diagram of the corresponding max-tree. Image from [5].

Figure 3.1: Diagrams illustrating the max-tree structure.

3.1.2 Disadvantages of the max-tree

Because the max-tree is specifically designed for volumes where the structures of interest are brighter than the background, it does not perform well for volumes where this is not the case. Volumes with dark structures on a light background can be filtered using a min-tree, which is essentially a max-tree with the order criterion reversed, or equivalently by inverting the gray values in the volume before and after filtering. However, if the background is dark in some places and bright in others, and the regions of interest have a brightness in between, both the max-tree and the min-tree will fail to distinguish these structures from parts of the background.

Furthermore, the max-tree cannot deal with color volumes at all, or with volumes with any other kind of multi-channel data on which no total order is defined. If there is no total order, it is possible for two different voxel values to be incomparable, i.e. neither is greater or less than the other. This presents a problem for the max-tree algorithm, which assumes the voxels in each region are either equal to or greater than the regional minimum.

3.2 Potential for the α-partition tree

The α-partition tree, however, does not have any of the aforementioned disadvantages. It can deal with dark backgrounds, bright backgrounds, or a mixture of the two, because it uses the contrast between voxels to detect structures, regardless of whether this contrast is positive or negative. The salience tree can also handle multi-channel data without a total order, as long as a distance measure is defined. This means the α-partition tree can be used to filter things like colored galaxy datasets or MRI scans with PD, T1 and T2 channels. Gray value-based filtering techniques can be used to filter each channel separately and combine the results, but a technique that integrally filters all channels at once could potentially perform better in terms of quality. Theoretically, the α-partition tree would also be expected to handle noisy backgrounds and gradients more gracefully.

Chapter 4

Salience tree implementation

The essentials of the α-partition tree algorithm are described in chapter 2, and a slightly more detailed overview is given by Smit [4]. The basic algorithm will not be explained again here, but there are a few more detailed aspects that are worth some elaboration.

4.1 Attribute filtering

The objective of attribute filtering is to filter an input image to only display structures for which a certain morphological attribute exceeds a certain threshold value. Morphological attributes only depend on the shape of a region, which in turn only depends on the coordinates of the comprising voxels, so attribute values can be computed for each node while building the tree. At filtering time, the tree is traversed top to bottom, attribute values are compared against the threshold, and nodes that do not meet the threshold are filtered. In the process of filtering, some nodes have their gray values changed, or are removed entirely.

After filtering, the filtered volume can be reconstructed from the tree. This is a trivial step, but its implementation differs slightly between the α-partition tree and the max-tree. In the α-partition tree, every voxel is represented by a unique leaf node, so each voxel is simply assigned the filtered value of its leaf node. In the max-tree, each voxel is part of one or more nodes, with the root node covering all voxels. Each voxel is assigned the value of the deepest surviving node that contains it.

4.1.1 Filtering strategies for the max-tree

What happens to nodes that don't meet the threshold is determined by the filtering strategy. Westenberg et al [5] describe five strategies for attribute filtering in max-trees.

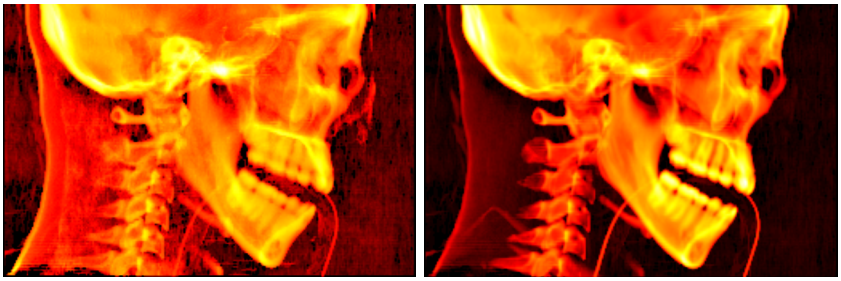
The first three strategies discussed here are pruning strategies. This means that if a node is removed, all of its descendants are removed as well. In the *min* strategy, every node that does not meet the threshold is removed, along with its descendants. This is equivalent to removing a node if the minimum of the attribute values along the path from the node to the root falls short of the threshold. In the *max* strategy, a node that does not meet the threshold is only

removed if all of its descendants have also been removed. This is equivalent to considering the maximum value along the aforementioned path. Finally, the *viterbi* strategy determines which nodes to remove by solving a lowest-cost path optimization problem. Salembier et al [3] describe this strategy in detail. Rendering a tree filtered with a pruning strategy is simple: voxels whose nodes survived are drawn with their original gray value, voxels whose nodes were removed are not drawn at all.

Pruning strategies are not very effective: they tend to remove too little or too much. Non-pruning strategies typically work better. When a node is removed, these strategies do not remove its children, but move them up in the tree, making them children of the nearest surviving ancestor of the removed node. If there is no surviving ancestor, the node's value is set to zero. This has the effect of removing noisy details in preserved structures and changing these voxels to match the ret of the structure, 'smoothening' it. It also removes the background, because the background voxels have no ancestors meeting the threshold, so they 'inherit' the value zero. If this latter rule were not in place, the background voxels would inherit from the root node instead and merely be lowered to match the darkest part of the background, rather than being eliminated completely.

In the *direct* strategy, nodes that do not meet the threshold have their gray value updated to equal the gray value of the nearest surviving ancestor. In the specific case of a max-tree, this always means the node's gray value is lowered. The node's children are not touched. In the *subtractive* strategy, a node that falls short of the threshold has its value lowered, as in the direct stategy, but its children are also lowered by the same value. In other words, the difference between the node's original value and its inherited value is subtracted from each of its children's values as well.

Both of these strategies work quite well. The subtractive strategy usually works better than the direct strategy, as it preserves the contrast between structures in cases where a preserved node is inside a filtered node which is itself inside a preserved node. In practice, the subtractive strategy is more effective at filtering out noise, as seen in figure 4.1.



(a) Max-tree with direct strategy (b) Max-tree with subtractive strategy

Figure 4.1: Comparison between filtering strategies using the *fullHead* dataset, filtered with the inertia attribute thresholded at $\lambda = 1$.

4.1.2 Filtering strategies for the α-partition tree

Attribute filtering in α-partition trees is a bit more involved, because the differences in structure between the α-partition tree and the max-tree introduce a few complications.

Level roots

While in the max-tree, the children of a node always represent a segmentation at a higher level of granularity, this is not necessarily the case in the α-partition tree. Conceptually, this is because in the max-tree a node can have any number of children, but because the α-partition tree is a binary tree, multiple tree levels may be needed to represent one conceptual level of segmentation. Algorithmically speaking, this happens because nodes are merged based on the values of the links between them, so the merging order will be undefined when a voxel has the same distance to two or more of its neighbors. For these reasons, the concept of level roots is introduced. A conceptual level of segmentation may span multiple tree levels, as long as the value of the link used to merge each node's children, called the node's alpha value, is equal in every node. These levels are also α-flat zones. The alpha value of a leaf node is zero. The highest node of the level is called the level root. A node is a level root if and only if its alpha value is different from its parent's alpha value. The root of the tree, which does not have a parent, is also a level root. Nodes that are not level roots belong to the same level as their nearest ancestor that is a level root. The level root of a node's level is also referred to simply as that node's level root. Levels and level roots are illustrated in figure 4.2.

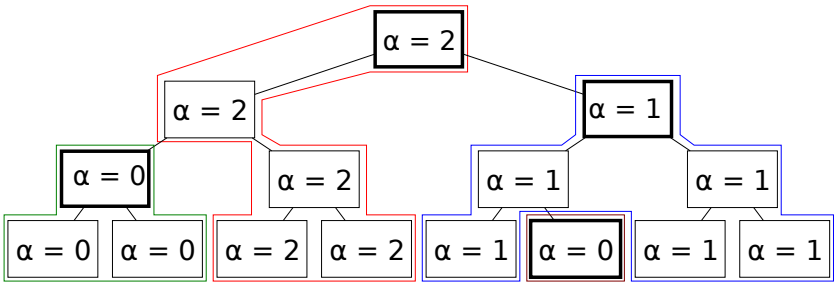


Figure 4.2: A α-partition tree with four levels. The colored lines surround the levels, the bolded nodes are level roots.

In the non-pruning filtering stategies described below, comparing the attribute value against the filtering threshold is only done for level roots. Nodes that are not level roots inherit their level root's value.

In the 2-D α-partition tree as used by Smit, a so-called omega factor is used to prevent levels from containing too wide a range of voxel values. Each node keeps track of the lowest and highest voxel value in its region. If a node would normally be a level root, i.e. its alpha value is not equal to its parent's alpha value, but the difference between the lowest and highest voxel value in its region would exceed the alpha value multiplied by the omega factor, i.e. if $max - min > \alpha \cdot \omega$, that node is not considered a level root and its level is effectively merged with its parent level. In my experimentation with 3-D volumes, I found that the

omega factor only degraded the quality of the filtering results. For that reason, I removed it from my 3-D α-partition tree implementation.

Level roots are also used for path compression. There is no conceptual or algorithmic difference between having nodes in the same level indirectly descend from their level root, or having them all be direct children of the level root, but the latter is faster. The algorithm frequently traverses the tree searching for level roots, subtree roots or common ancestors when deciding whether two voxels' regions have already been merged and, if not, which nodes to join together. Every time such a traversal happens, each node's parent link is overwritten to point to its level root. This speeds up subsequent lookups enormously. The difference between this approach and root caching is that the parent links are actually overwritten, so the original parent information is lost and the tree structure is changed. However, this does not present any problems. As mentioned before, the way nodes are structured within a level is meaningless because the merging order is undefined. Furthermore, the fact that this approach violates the otherwise binary nature of the tree is not problematic either, because tree traversal is only ever done bottom-up using parent links, never top-down using child links. In fact, the implementation does not even store child links. Figure 4.3 illustrates the changes path compression makes to the structure of a tree.

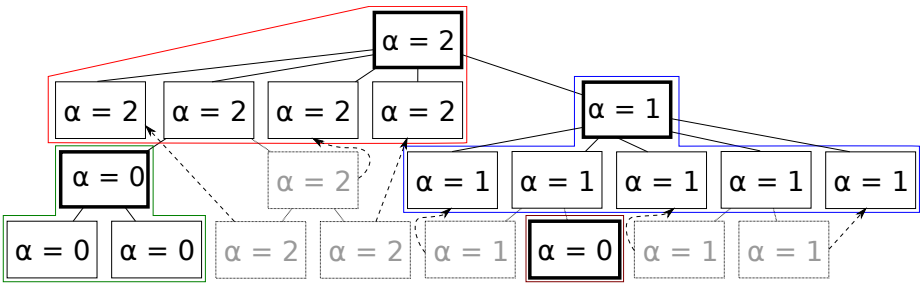


Figure 4.3: Path compression applied to the tree from figure 4.2. Relocated nodes have dotted outlines, grayed-out text and a striped line to their new position.

Non-pruning filtering strategies

For the α-partition tree, I have implemented direct and subtractive filtering. I have not implemented any of the pruning strategies, because direct and subtractive are known to work better.

However, these methods need to be adapted slightly, because the way they are implemented in the max-tree relies on certain characteristics of the max-tree that it does not share with the salience tree. In the max-tree, the value of a node is clearly defined as the minimum voxel value in the node's region. This is a natural choice because of the ordered nature of the max-tree, but this is not necessarily a natural choice in the α-partition tree. The 2-D α-partition tree implementation on which I based my 3-D implementation uses the average voxel value in the node's region. I have also used this in my 3-D implementation, but it is possible a better definition exists.

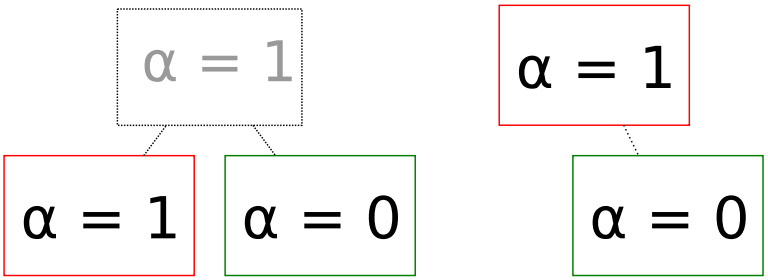
Direct filtering was the only filtering strategy used for 2-D α-partition trees, but subtractive filtering can be used as well, and I have implemented it as part of my 3-D α-partition tree implementation. It does require subtractive filtering to be redefined, however. In the max-tree, the subtractive strategy is based on the assumption that node values are strictly increasing; therefore, a node inheriting its value from an ancestor will necessarily be lowered in value, that difference can then be subtracted from the node's descendants' values, and it makes sense to speak of subtractive filtering. However, the α-partition tree does not necessarily have strictly increasing node values, and this introduces complications. It is possible that a node's value is raised, not lowered, when it inherits from an ancestor. In this case, the α-partition tree implementation adds the difference to the descendant nodes' values rather than subtracting it. Additionally, if a node has a lower value than one of its ancestors, it is possible for the algorithm to try to lower that node's value below zero. Similarly, when adding values it is conceivable that it will try to increase a node's value above the maximum value for the datatype used and cause an overflow. These issues are dealt with by clamping node values.

4.2 Other optimizations

As mentioned before, the α-partition tree algorithm relies heavily on path compression for optimization. Two other small optimizations are used, but they merely serve as shortcuts to do path compression during tree construction; in other words, they are slight changes to the tree construction algorithm that produce a tree with some path compression already done.

One optimization occurs very early in the process, when populating the priority queue with links between voxels. In α-partition tree parlance, these links are referred to as edges, the priority queue is called the edge queue, and the edges' values in the queue are called edge strengths. Edges between voxels with identical values are not inserted into the edge queue: instead, the voxels are merged immediately. This does not materially change the tree structure: these edges have an edge strength of zero, so they would have been the first ones to be processed for merging anyway, and the order in which edges with the same strength are processed is undefined and irrelevant. It does, however, reduce the size of the edge queue significantly for volumes with large flat regions such as backgrounds. Since populating and processing the edge queue contributes most to the algorithm's overall complexity, weighing in at $O(n \log n)$ where n is the number of edges in the queue, reducing the size of the edge queue is a particularly effective optimization.

The other optimization occurs when merging two nodes would cause the newly created parent node to have the same alpha value as one of its children. In that case, the node with the lower alpha value is made a child of the node with the higher alpha value, and no new parent node is created. This is illustrated in figure 4.4. This does not introduce any semantic differences, because the way nodes are structured within a level is not relevant, but it does reduce the size of the tree. This saves precious time in the filtering algorithm, which traverses all nodes in the tree in linear time every time it runs.



(a) Merging two nodes normally by adding a common parent.

(b) Optimizing the common parent away.

Figure 4.4: Diagrams illustrating an optimization where common parents are optimized away if they would have the same alpha value as one of their children.

4.3 Alternative edge strength measures

The choice of the distance function discussed earlier, also called the salience function or edge strength measure, can significantly affect the filtering output. A suitable edge strength measure is key to successfully identifying regions of interest. As mentioned previously, there are a number of edge strength measures to choose from for multi-channel datasets like RGB volumes, but for gray-value volumes there seems to be only one obvious candidate. However, even for gray-value volumes, interesting edge strength measures can be designed by taking neighboring voxels into account.

4.3.1 Edge strength measures based on neighboring voxels

In gray-value volumes, the obvious choice for edge strength would seem to be the *traditional* measure: the absolute value of the difference between the gray values of the two voxels. In mathematical notation, denoting the voxel values with a and b, this would be $|a - b|$.

However, because the traditional measure only considers single voxels, it is potentially more sensitive to noise. In an attempt to make the algorithm less sensitive to noise, I have experimented with two edge strength measures that take the values of neighboring voxels into account as well. The *averages* measure computes the average in each voxel's immediate neighborhood, and uses the absolute value of the differences between those averages as the edge strength. The *maxima* measure does the same with the maximum of each neighborhood.

If the two voxels connected by the edge are called A and B, the immediate neighborhood of A consists of A itself and all voxels immediately adjacent to it in any direction, except for B. Assuming 6-connectivity, this means six voxels, A and five of its neighbors, are examined to find A's average or maximum. The naming of these neighbors is illustrated in figure 4.5. Denoting voxel values with lowercase letters such that A's voxel value is a, A_1's is a_1, etcetera, the averages measure can be expressed as $\left|\frac{a+a_1+a_2+a_3+a_4+a_5}{6} - \frac{b+b_1+b_2+b_3+b_4+b_5}{6}\right|$ and the maxima measure as $\max(|a-b|, |a_1-b_1|, |a_2-b_2|, |a_3-b_3|, |a_4-b_4|, |a_5-b_5|)$.

The formulas above assume that all of the neighbors A_1 through A_5 and B_1 through B_5 exist. This is not the case if A or B is near one of the edges of the

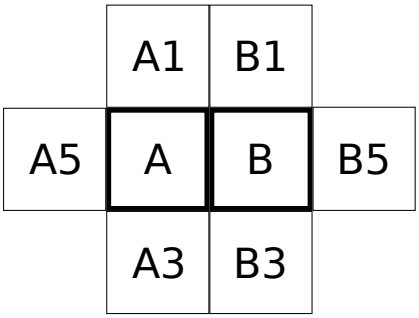


Figure 4.5: The naming of the neighbors of A and B. A_2, B_2, A_4 and B_4 are not shown because they are not in the same plane. A_2 and B_2 are in front of A and B, and A_4 and B_4 are behind A and B.

volume. In the averages measure, such missing neighbors are simply dropped from the calculation, which may result in averaging five or fewer values for one or both of A and B. It is possible for A and B to have different numbers of existing neighbors, in which case the averages measure will simply average different numbers of values for A and B. However, this is not an option for the maxima measure, which pairs each of A's neighbors with the corresponding neighbor of B. Therefore, the maxima measure only takes pairs of neighbors into account if both neighbors exist; if a neighbor exists but its counterpart does not, it is not used in the computation.

Chapter 5

Visualization

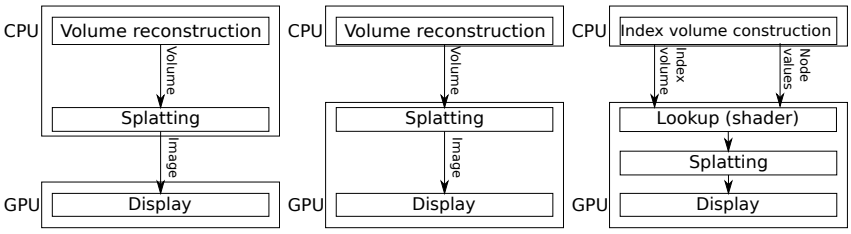
After filtering, the filtered volume typically needs to be either written to a file, or visualized for the user. The former case is not very interesting: the volume is reconstructed from the filtered tree and written out. Visualization is more interesting, because the user will typically want to filter the volume again with a different treshold. The goal of interactive visualization is to make re-filtering with a different threshold and visualizing the result fast enough, typically in at most a few seconds, so the user can adjust the threshold and see the results in real time.

Traditionally, filtering results are visualized by reconstructing the filtered volume and using splatting to render a projection. Splatting is a volume rendering technique introduced by Westover in 1990 [6]. It uses approximations to trade quality for speed. Each voxel is projected onto the 2-D viewing plane. The projection of a voxel, called a *splat*, is a small circle or polygon whose color and transparency are based on the color and transparency of the voxel, transformed using a transfer function and vary according to a Gaussian function. These splats are then rendered back-to-front to produce the final image.

The splatting can happen either on the CPU, in which case the projection is sent to the graphics card, or on the GPU, in which case the entire volume is sent to the graphics card. These two approaches are illustrated in figure 5.1(a) and figure 5.1(b), respectively. The latter approach requires more data to be transmitted, but splatting on the GPU is typically faster than splatting on the CPU, and re-rendering the volume with different parameters, e.g. after rotating or zooming, does not require any data to be retransmitted.

5.1 Indirect volume rendering

Westenberg et al [5] introduce a third alternative, which uses an index volume and a representation of the tree on the graphics card. The volume that is sent to the graphics card consists of indices instead of RGBA or gray values. For each node, this index volume stores the index of the tree node it belongs to. The values in the index volume index into an array of node values that is stored in the texture buffer. To render this, the graphics card has to translate the indices in the volume buffer by looking up the associated node value in the texture buffer. This is done by a small shader program. The data flow for this



(a) Traditional data flow with splatting on the CPU

(b) Traditional data flow with splatting on the GPU

(c) Data flow for the Westenberg et al approach

Figure 5.1: Illustrations of the data flow for each of the visualization approaches discussed in this chapter.

technique is illustrated in figure 5.1(c).

The advantage of this technique is that, when the volume is re-filtered, it is not explicitly reconstructed or sent to the graphics card. Instead, only the changed node values are retransferred; the index volume depends only on the structure of the tree and does not change when re-filtering. Assuming the number of nodes is much lower than the number of voxels, this means less data is transferred, which speeds up rendering. According to Westenberg et al, data transfer is a major bottleneck in interactive visualization, and this technique allowed them to achieve refresh times below 100 ms.

5.2 Indirect rendering with α-partition trees

Westenberg et al developed their indirect volume rendering technique for the max-tree, but it can easily be adapted to work with the α-partition tree, or any other data structure. The only requirement is that the number of nodes is significantly lower than the number of voxels. This is not readily true in the α-partition tree: every voxel has its own leaf node, and the tree structure is built on top of that. However, each of these leaf nodes will always inherit its filtered value from its level root. Because leaf nodes always have an alpha value of zero, its level root will also have an alpha value of zero, so all leaf nodes descending from this level root, collectively called a leaf level, will have equal voxel values. This means that if each index in the index volume is set to point to its voxel's level root rather than its voxel's leaf node, and only nodes that are pointed to are stored on the graphics card, only one node will be stored for every connected equal-value area.

This reduces the number of nodes immensely. For the *foot* dataset from `http://www.volvis.org` with 16,777,216 voxels, a α-partition tree with 18,724,805 nodes is built. The dataset has a connected, all-black background consisting of 11,870,561 nodes with gray value zero, all of which are pointed to one single node in the index volume. Overall, the level root approach reduces the number of nodes transferred to the graphics card to 4,340,761, or 26% of the number of voxels. However, a max-tree for the same volume with the same connectivity only has 627,975 nodes.

Chapter 6

Results

To measure the performance of my α-partition tree implementation, I used three publicly available datasets: *piggy bank* (courtesy Michael Bauer, Computer Graphics Group, University of Erlangen, Germany) from The Volume Library at `http://www9.informatik.uni-erlangen.de/External/vollib/`, *fullHead*, a dataset that used to be shipped with VTK and is now hosted at `http://www.cs.rug.nl/~michael/MTdemo`, and *vessels*, hosted at the same URL. The benchmarks were done on a laptop with 8 GB of memory and a quad-core Intel i7-740QM processor. The graphics card was not used: only tree construction and filtering were benchmarked, volume reconstruction and visualization were not. The traditional edge strength measure was used for these benchmarks. The alternative edge strength measures described in section 4.3.1 slow down tree construction. This effect is small but noticeable for the *maxima* measure, and significant for the *averages* measure. Most likely, this is because the edge strength computation is run very often, and involves floating point computations in the case of the *averages* measure. All benchmarks were done using 6-connectivity, even though the max-tree uses 26-connectivity by default. This was done because the α-partition tree only supports 6-connectivity at this time, and using different connectivities would have skewed the results.

6.1 Performance

Compared to the max-tree, the performance of the α-partition tree is lacking in every respect. As detailed in table 6.1, building the α-partition tree is many times slower than building the max-tree and uses approximately three times as much memory. The filter time, while not as good as the max-tree's, is acceptable for interactive visualization. As outlined in section 7.1, there is clearly room for optimization in the tree construction phase, both in terms of speed and memory usage. The filtering phase can also be sped up.

6.2 Quality

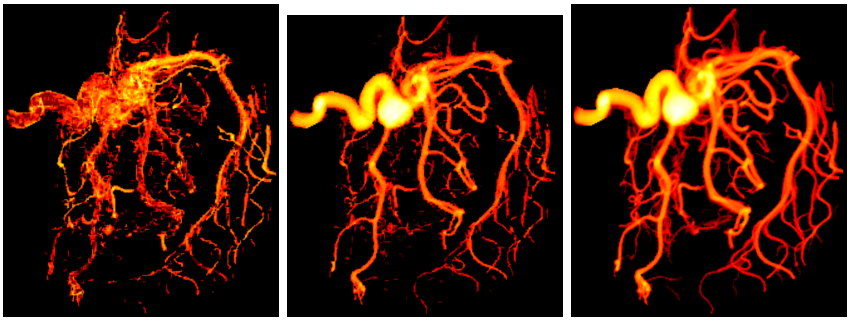
In figure 6.1, the results of filtering the *vessels* dataset are compared. The max-tree clearly performs best, dropping fewer vessels and leaving less noise. The traditional edge strength measure suffers greatly from fragmentation, with

Dataset	*Vessels*	*Piggy bank*	*fullHead*
Number of voxels	16,777,216	35,127,296	11,534,336
Dynamic range	8 bits	12 bits	12 bits
Build time (s)	10.5 (4.5)	69.2 (18.2)	27.7 (6.1)
Filter time (s)	0.40 (0.01)	1.75 (0.98)	0.72 (0.13)
Number of nodes	16,826,409 (54,454)	39,505,389 (667,972)	15,011,722 (927,170)
Number of leaf levels	130,482	17,894,351	8,395,678
Peak memory usage (GB)	0.95 (0.40)	3.6 (1.0)	1.3 (0.44)

Table 6.1: Salience tree benchmarks compared to the max-tree. Max-tree numbers are in parentheses. All timings are five-run averages.

filtering cutting small holes out of the vessels, and leaving small bits and pieces of background noise. The *averages* edge strength measure almost completely eliminates the fragmentation of the vessels, but the background fragments remain. Some of these fragments are very persistent and are preserved even when some the threshold is raised and some of the larger vessels are filtered out. The results of the *maxima* measure are very similar to those of the *averages* measure, but the fragmentation of small blood vessels and background artefacts is slightly worse when using the *maxima* measure.

To investigate how the max-tree and the α-partition tree handle volumes with noisier backgrounds, I modified the *vessels* dataset by adding a gradient. The gradient runs in the x direction and has slope 1, and is added to the original voxel value. That is to say, the x coordinate of each voxel is added to the original value of that voxel. Moreover, to ensure the gradient is only applied to background voxels, this addition is only done if the voxel's original gray value does not exceed 20. With this gradient applied, the performance of both methods suffers: in figure 6.2, fewer blood vessels are shown than in figure 6.1. However, the α-partition tree seems to suffer slightly less than the max-tree: the difference between the two methods' filter outputs has become smaller. The *averages* edge strength measure clearly works better than the traditional or *maxima* measures: the latter two eliminate all but a handful of vessels and leave planes with a single gray value in the middle of the volume. The *averages* measure and the max-tree also leave one or two of these planes, but they are located on the sides of the volume, away from the vessels. Direct filtering was used for this volume because subtractive filtering eliminated many vessels.

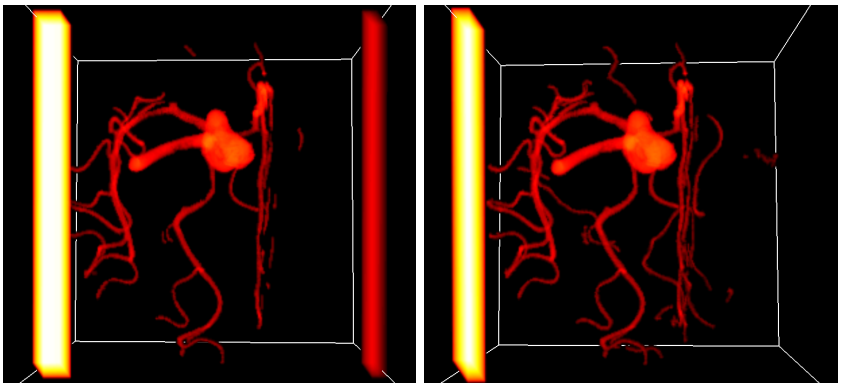


(a) Salience tree with traditional edge strength measure, $\lambda = 0.4$

(b) Salience tree with *averages* edge strength measure, $\lambda = 0.4$

(c) Max-tree, $\lambda = 1.2$

Figure 6.1: The *vessels* dataset filtered with the inertia attribute, using subtractive filtering.



(a) Salience tree with *averages* edge strength measure

(b) Max-tree

Figure 6.2: The *vessels* dataset with an added gradient, filtered with the inertia attribute thresholded at $\lambda = 1.2$, using direct filtering. The volume is viewed from a different angle than in figure 6.1 because of the bright plane on the left.

Chapter 7

Future work

It is evident from the previous chapter that the α-partition tree is slower than the max-tree, uses more memory, and produces results of lower quality. This chapter discusses strategies that could be used to improve the speed, memory usage and result quality.

7.1 Possible optimizations

The current implementation is quite inefficient in terms of both memory usage and speed. The excessive memory usage is mainly caused by the large number of nodes in the tree and the large size of each node in memory. There is also a significant amount of memory that is wasted because the size of the tree is not known in advance, so the algorithm allocates memory for the maximum number of nodes, and because at least half of the tree consists of single-voxel leaf nodes.

A possible approach to reducing memory usage is to initially store the structure of the tree only, then allocate and populate node objects once the structure is built. The structure could be recorded in an array containing parent indices that index into that same array. This takes a minimal amount of memory, and allocating that parent array for the worst-case tree size does not result in much memory waste.

Building on that, the single-voxel leaf nodes could be made implicit. All of the data in such a node object, except for the parent link, can easily be derived from the voxel value. These leaf nodes comprise at least half of the nodes in the tree in theory; in the benchmarks in table 6.1, leaf nodes make up 75 to 99.7 percent of the tree. Not storing these leaf nodes explicitly would therefore significantly reduce memory usage. It would improve speed, as well: populating all of these leaf node objects with data from the volume takes time. For the *piggy bank* dataset, with 35,127,296 voxels, this population step took over 2 seconds in the benchmarks.

Making leaf nodes implicit could foster speed improvements as well. The current filtering algorithm wastes precious time writing its output values to leaf nodes objects first, then copying them to the output volume one by one. Optimizing this by writing the output values that would normally be written to leaf node objects directly to the output volume instead is already possible in the current implementation, but moving to implicit leaf nodes would force this.

7.2 Quality improvements

One explanation for the quality of the results is that the datasets used have properties that the max-tree was designed for. The max-tree makes certain assumptions about its input which, while impairing its ability to deal with input where these assumptions do not hold, enable it to be more effective when they do hold. The α-partition tree suffered less from the background gradient added to the *vessels* volume, and while I failed to find volumes for which the α-partition tree works better than the max-tree, they might exist. Additionally, filtering of color and multi-channel data could be attempted.

Improving the quality of the results might be done by using a higher connectivity, like 26-connectivity. However, using 26-connectivity will slow down the generation of the α-partition tree by more than a factor four, because the edge queue will be correspondingly larger. It is also not likely to cause spectacular improvements, and relative to the max-tree the quality might not improve at all, as the max-tree's output quality will also benefit from using 26-connectivity.

The most fruitful way forward seems to be the careful choosing of edge strength measures. As I have shown, using a different edge strength measure can have a significant impact on the quality of the results. While the edge strength measures I have used are very generic, assumptions about the properties of the input could be built in by crafting an edge strength measure that favors these properties. For instance, for volumes with bright structures on a dark background, one might define edge strength as the relative difference between voxels as opposed to absolute difference. This would be expected to emphasize contrast between a structure and its surroundings over contrast between voxels inside a structure. For multi-channel data where the structures of interest are bright in one channel and dark in the other, one might use a distance measure like $|(a_2 - a_1) - (b_2 - b_1)|$ to emphasize contrast between such structures and the background. The possibilities are endless.

Bibliography

[1] Georgios K. Ouzounis and Pierre Soille. Pattern spectra from partition pyramids and hierarchies. In *ISMM*, pages 108–119, 2011.

[2] P. Salembier and L. Garrido. Binary partition tree as an efficient representation for image processing, segmentation and information retrieval. *IEEE Trans. Image Proc.*, 9(4):561–576, April, 2000.

[3] P. Salembier, A. Oliveras, and L. Garrido. Anti-extensive connected operators for image and sequence processing. *IEEE Trans. Image Proc.*, 7:555–570, 1998.

[4] Jasper B. Smit and Michael H. F. Wilkinson. Vector-attribute filters based on salience trees for object detection in color images.

[5] M. A. Westenberg, J. B. T. M. Roerdink, and M. H. F. Wilkinson. Volumetric attribute filtering and interactive visualization using the max-tree representation. *IEEE Trans. Image Proc.*, 16:2943–2952, 2007.

[6] Lee Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24:367–376, 1990.