Chapter 7

Concluding Remarks

7.1 Summary

This thesis deals with efficient algorithms for morphological image processing. Each chapter deals with some algorithm from the field of mathematical morphology. Reviews of existing algorithms, as well as new algorithms or improvements to existing algorithms are given. In most chapters, the focus is on the design and implementation of parallel algorithms. The interest for parallel algorithms is mainly driven by the growing demand of real-time 2D image processing applications, and algorithms for the interactive processing of 3D volume data sets. Parallel algorithms for the processing of 3D volume data sets are especially interesting in the field of medical imaging, where the processing of large MRI and CT-data sets has become daily practice.

In chapter 2 a general algorithm for computing the distance transform of a binary image in linear time is presented. A distance transform computes a grey scale image of which each pixel is assigned its distance to the nearest foreground pixel in the binary image. The presented algorithm is general in the sense that it can be used for different metrics. The metrics discussed in chapter 2 are the Manhattan or City-block metric ($L_1$-metric), the chessboard metric ($L_\infty$-metric), and the Euclidean metric ($L_2$-metric).

The algorithm can be summarized as follows. In a first phase each column $C_x$ (defined by points $(x,y)$ with $x$ fixed) is separately scanned. For each point $(x,y)$ on $C_x$, the distance $G(x,y)$ of $(x,y)$ to the nearest foreground pixel in the same column of the binary image is determined. In a second phase each row $R_y$ (defined by points $(x,y)$ with $y$ fixed) is separately scanned, and for each point $(x,y)$ on $R_y$ the minimum of $(x-x')^2 + G(x',y)^2$ for the $L_2$-norm, $|x-x'| + G(x',y)$ for the $L_1$-norm, and $|x-x'| \max G(x',y)$ for the $L_\infty$-norm is determined, where $(x',y)$ ranges over row $R_y$. Both phases consist of two scans, a forward and a backward scan. The time complexity of the algorithm is linear in the number of pixels. Since the computation per row (column) is independent of the computation of other rows (columns), the algorithm is easily parallelized for concurrent execution on shared memory computers. A barrier is used as a synchronization point between the two phases. The algorithm turns out to be very efficient, and scales almost perfectly on shared memory architectures.

The algorithm is easily extended to $d$-dimensional ($d > 2$) distance transforms by separating the algorithm into $d$ phases, each solving a one-dimensional problem.

In chapter 3 the design is described of a parallel algorithm for the determination of connected
where the nodes represent the connected components of the image thresholded at various levels. The tree is a rooted tree, of which the regional extrema are the leaves. Each node $C$ in the tree has a parent pointer to another node, which represents the connected component $C$ is contained in when the data set is thresholded at a lower level. Salembier uses a recursive flooding algorithm, to construct the tree. He uses a *hierarchical queue* for the flooding process, which is basically a queue of queues. This queue structure is used for flooding and for processing neighboring components of a regional extremum in order of grey-value. In the nodes of the tree attributes, like area, can be stored. Filtering the image boils down to following parent pointers in the tree, until a node is reached where the filter criterion is satisfied. The computational cost of the Max-tree algorithm is dominated by the flood filling process which is inherently linear in both the number of pixels and in the connectivity. Filtering the Max-tree requires visiting all nodes in the tree, so it is also $O(N)$. Computing the output image is also linear in the number of pixels. One somewhat peculiar feature of the computational complexity of the Max-tree algorithm is its dependency on the number of unoccupied grey levels between parent and child nodes, since a linear search is necessary to find the next occupied grey level.

The key difference between the algorithm based on union-find and the methods of Vincent and Salembier et al. is that multiple extrema are processed simultaneously, rather than sequentially. The algorithm is based on the idea to compute connected components using union-find (see chap. 3), while using Salembier’s idea of constructing a tree. The tree is replaced by a forest of smaller trees. Each tree is extended until the root of the tree satisfies the filter criterion.

The union-find algorithm outperforms the other methods on almost all natural and synthetic images tested. The running time of the Max-tree algorithm is independent of $\lambda$, while the union-find algorithm shows a very weak dependence. Still, from practical experiments with 2D images, the Max-tree algorithm is typically slower by a factor of 2. The advantage of the union-find algorithm is even greater for the processing of 3-D data sets. This is probably due to its smaller memory requirements, especially given the limited bandwidth of the memory bus of personal computers. In the case of a $128 \times 128 \times 62$ volume, the total memory use of the Max-tree method was 57.5 MB, the priority-queue method needed 45 MB, whereas the union-find algorithm required only 25 MB. Chapter 4 briefly discusses the extension to *attribute openings* and *pattern spectra*. It has been demonstrated that it is possible to compute an area pattern spectrum in the time needed for a single area opening.

Although the attribute opening (closing) of Meijster and Wilkinson is faster in the case in which one wants to filter a data set with a given parameter $\lambda$, the Max-tree algorithm does have its merits. It is very useful in the case one wants to filter a data set interactively, by first computing the tree structure, and supplying the user with some graphical user interface to choose the parameter $\lambda$. Filtering the tree is a fast operation, while constructing the tree can be done off-line or as a pre-processing step. In view of this application, a parallel algorithm for constructing and filtering Max-trees is discussed in chapter 5.

In chapter 6 an extensive review is given of various existing definitions of the watershed transform based on immersion or on shortest paths with respect to a topographical distance function. The watershed transform can be classified as a region-based segmentation approach. The idea is to look upon a grey-scale image as a landscape, where grey levels denote altitude. An intuitive description of the watershed transform goes as follows. Imagine the landscape being
immersed in a lake, with holes pierced in local minima. Basins (also called ‘catchment basins’) will fill up with water starting at these local minima, and, at points where water coming from different basins would meet, dams are built. When the water level has reached the highest peak in the landscape, the process is stopped. As a result, the landscape is partitioned into regions or basins separated by dams, called watershed lines or simply watersheds.

The main sequential algorithms for computing the watershed transform according to both definitions are described. Emphasis is put on the fact that watershed algorithms found in the literature often do not adhere to their definition, or are the implementation of an algorithm without a proper specification.

Strategies for parallel implementation are discussed, distinguishing between distributed memory and shared memory architectures. The watershed algorithm by immersion is hard to parallelize because of its inherently sequential nature. A parallel implementation of this algorithm can be based upon a transformation to a components graph. The distance-based definition allows various parallel implementations. The main ones are based on (ordered) queues, repeated raster scanning, a modified UNION-FIND algorithm, or a combination of these. The main conclusion to be drawn from this review is that, despite all the techniques and architectures used, there is always a stage in the watershed transform which remains a global operation, and therefore in the case of parallel implementation at most modest speedups are to be expected.

7.2 Perspectives

This thesis has presented and reviewed only a small selection of algorithms for commonly used morphological operators. Clearly, there are many algorithms for morphological operators that were not discussed. Some of them are closely related with the ones discussed in this thesis. For example, it would be interesting to investigate if it is possible to extend the distance transform algorithm (chapter 2) for binary images into one that can accept grey-scale images as its input. The idea is to compute for each pixel the distance to the nearest pixel with a lower grey-value, i.e. the lower distance as discussed in chapter 6. Apart from its own merit, such an algorithm would be beneficial when we want to compute the watershed according to Meyer’s definition.

The connected components algorithm presented in chapter 3 could possibly be improved by using a computation scheme that resembles the distance transform algorithm. In the case of 2D images, it is possible to scan each row of the image independently, and construct small disjoint set trees per scan-line. These small trees are clearly completely compressed, i.e. each pixel points directly to its parent. Obviously, this processing phase takes linear time, and is trivially parallelized. In a second phase, neighboring scan-lines are merged, using the information coded in the trees obtained from the previous scan. These trees represent a partition of a scan-line in segments, in which the grey-value is constant. We can use this information to skip vertical edges between pixels $p$ and $q$ which both lie in the interior of such a segment. In a way, we could regard this technique as connected component labeling on run-length encoded images. In the case of 2D images this could be worthwhile, especially in images with large (background) components. It is not clear how to extend this idea to 3D data sets.

The generality of the attribute opening (closing) algorithms of chapters 4 and 5 yields a
perfect and general setup to search for new attributes that can be used to segment and filter large data sets. At this moment we are busy to incorporate the algorithm in a radiotherapy application, that supplies the user with a 2D and 3D view on 3D CT and MRI data sets.

It would be interesting to investigate whether it is possible to use the Max-tree approach for pre-computing a number of texture images for faster interactive visualization for certain ranges of the filter parameter $\lambda$. It might also be a good idea to compute textures predictively based on the current value of the filter parameter chosen by the user.

As far as the watershed is concerned, we only discussed algorithms for computing the watershed of an image directly. This is hardly ever useful, since this yields a severe over-segmentation. It would be interesting to investigate if it is possible to construct a concurrent algorithm with which a user can interactively select markers (user selected local minima), such that this over-segmentation is reduced.

Finally, it would be a good idea to incorporate the algorithms presented in this thesis in well known image processing and visualization libraries.
components in binary and grey-scale images. Points of an image are regarded connected if they are neighbors and have the same grey-value. A set of points \( C \) is a connected component if for each point in \( C \) a connected path of points in \( C \) exists to any other point in the set, and no more points can be added to \( C \) while maintaining this property.

A concurrent version of Tarjan’s Union-Find algorithm for determining equivalence classes in a graph is used. Although we are interested in the application to images, we present the algorithm for undirected graphs. The design goes through four stages. We first give a version of Tarjan’s sequential algorithm. Distribution of the vertices of the graph over a number of processes leads to a message passing algorithm. This design is then mapped to a shared memory architecture by means of mutual exclusion and synchronization. Finally, the mutual exclusion and synchronization are implemented by means of POSIX thread primitives. The resulting algorithm is a concurrent one in which the amount of communication is decided at runtime. As a consequence, processes must not stop when they finished some task, since other processes may send a new one. This yields a termination detection problem, which is solved by counting the number of tasks that still need to be done.

We finally describe the application to the determination of connected components in images. Graphs have no dimension, so the algorithm is applicable to images of any size and dimension. Since images are usually more or less constant locally, we sketch an optimization that can significantly reduce the number of communications needed. Experiments show that distribution is quite effective, and we observe a speedup that is often almost linear in the number of processors.

In chapter 4 a review is given of algorithms for connected set openings and closings, and in particular attribute openings and closings. The two best known algorithms for these image filters are the pixel priority-queue method introduced by Vincent for area operators [99] and extended by Breen and Jones to attribute operators [19], and the Max-tree method of Salembier and co-workers [83] which uses hierarchical queues. These two methods are compared to an algorithm developed by Meijster and Wilkinson [57, 58], which is based on the union-find algorithm [88]. The focus is on the area opening since this is one of the simplest cases.

An area opening lowers the grey-value of each pixel that belongs to a connected component that has an area smaller than some threshold \( \lambda \) until the component has an area at least \( \lambda \). In the binary case, area closings fill all background components with an area smaller than \( \lambda \). The area closing is the dual of the area opening. An area opening (closing) can either remove image components completely, or leave them intact, but never alter their shape.

For the determination of connected components, the algorithm of Vincent is based on a region growing (flooding) algorithm that uses a priority queue. Each regional extremum in the image is processed sequentially, and is expanded with neighboring pixels, until it either satisfies the area criterion, or a pixel with higher grey-value is encountered. In the latter case, the processing of the component is stopped, since it will be re-flooded later during the processing of some other regional extremum. If \( N \) is the number of pixels in the image, the computational complexity of the algorithm is \( O(N\lambda \log \lambda) \), which becomes \( O(N^2 \log N) \) if \( \lambda = N \). The latter is the case when computing pattern spectra (see section 4.7). Experiments show that the running time of the algorithm depends strongly on the parameter \( \lambda \), but also on the number of regional extrema.

The algorithm of Salembier of consists of two phases. First a tree, called a Max-tree, is constructed from the image, after which the tree can be used for filtering. A Max-Tree is a tree