Parallel Front Propagation in Medical Image Segmentation



Varduhi Yeghiazaryan
Keble College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2017

To my father

Acknowledgements

I must acknowledge my supervisor, Dr Irina Voiculescu, for her unceasing patience, invaluable advice, wisdom, and commitment. She imparted to me immense knowledge and helped me hone my skills in research.

I would like to thank my college advisor, Dr Stephen Cameron, for guiding and encouraging me with his support and profound insights throughout my doctoral studies at Keble College, University of Oxford.

I am grateful to Dr Zoë Traill of the Churchill Hospital, Oxford, for providing abdominal CT scan data and for spending considerable time drawing gold standard contours with the *millipede* tool. I would like to thank Dr Hemant Pandit, Dr Elise Pegg, and Dr Mona Alinejad of the Nuffield Department of Orthopaedics, Rheumatology and Musculoskeletal Sciences, University of Oxford, for the knee MRI images and their manual segmentation.

I would like to express my gratitude to Dr Stuart Golodetz for all the information he provided about *millipede* and medical image data. Many thanks to Dr Marija Marčan for the conversations about medical image processing.

I appreciate the contributions of Avraham Sherman and Mokrane Gaci to the experiments with human rankings of segmentations using the synthetic fractal images. Also, thanks to Avraham for the 3D printed kidney model.

I am most grateful to the Department of Computer Science, University of Oxford, for the Teaching Assistantship Grant that enabled me to pursue a doctoral degree while gaining important teaching experience. I extend my sincere gratitude to everyone in the department.

I am eternally indebted to my parents and my sister for their sacrifices, endless support, and love. I could not have completed this journey without them. My greetings and thanks to all my friends who have always inspired and supported me from Oxford, Yerevan, and all other corners of the world.

Abstract

Newly emerging medical image modalities, large high-dimensional images with resolution and quality improved over time, introduce new challenges for traditionally sequential segmentation procedures. We contribute to three aspects of medical image segmentation, and we provide a new, more reliable way to evaluate the results of segmentation.

The watershed transform is a popular image partitioning procedure from mathematical morphology, used in many applications of computer vision. Our first contribution is to automate an organ segmentation procedure. The algorithm is sequential and is based on the construction of an image partition forest—a watershed-based hierarchical partitioning of a 3D image—followed by the fast marching method for hypersurface front propagation.

Secondly, we parallelise the watershed procedure. Our algorithm constructs paths of steepest descent and reduces these paths into direct pointers to catchment basin minima in logarithmic time, also crucially incorporating successful resolution of plateaux. Three GPU implementation variants and their parameters are analysed through experiments on 2D and 3D images.

Thirdly, we propose the fast dashing parallel method for hypersurface front propagation in a partitioned image. This new parallel procedure solves the eikonal equation at voxels on partitioning region boundaries and 'dashes' with constant arrival time inside partitioning regions. Experiments on 2D and 3D data show that the GPU implementation of fast dashing executes faster than traditional eikonal solvers and produces qualitative results that fit better for medical image segmentation.

Our novel way to evaluate segmentation results relies on a new family of metrics, with hybrid characteristics. These metrics quantify the similarity or difference of segmented regions by considering their average overlap in fixed-size neighbourhoods of points on the boundaries of those regions. Our metrics are more sensitive to combinations of segmentation error types than other metrics in the existing literature. We compare the metric performance on collections of segmentation results sourced from carefully compiled 2D synthetic data and 3D medical images. We show that our metrics: (1) penalise errors successfully, especially those around region boundaries; (2) avoid overly inflated scores; and (3) score segmentation results over a wider range of values.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Goals 2

1.3 Contribution 3

2 Hardware and Data **7**

2.1 Image types 7

2.1.1 Digital raster images 7

2.1.2 Gradient magnitude images 9

2.2 Medical images 9

2.2.1 CT 9

2.2.2 Micro-CT 10

2.2.3 MRI 10

2.2.4 Windowing 11

2.2.5 Representation 12

2.2.6 Scanner hardware 12

2.2.7 Ethics 13

2.3 Input and output image formats 14

2.4 Computing hardware 15

2.5 File formats 15

3 Literature Review **17**

3.1 Medical image segmentation 17

3.1.1 The task 17

3.1.2 Segmentation approaches 18

3.2 Watershed transform 20

3.2.1 Description 20

3.2.2 Problem of over-partitioning 22

3.2.3 Plateaux problems 23

3.2.4 Algorithms 23

3.2.5 Parallel watershed 26

3.3 Waterfall transform 27

3.3.1 Description 27

3.3.2 Algorithms 27

3.4 Front propagation with eikonal equation 28

3.4.1 Fast marching basics 29

3.4.2 Other eikonal solvers 34

3.4.3 Use in medical image segmentation 35

3.5 Validation 35

3.5.1 Important factors 36

3.5.2 Classification of evaluation method types 38

3.5.3 Evaluation frameworks 40

3.5.4 Evaluation in practice 41

3.6 Existing evaluation methods 43

3.6.1 Overlap-based methods 43

3.6.2 Size-based methods 45

3.6.3 Boundary-distance-based methods 45

3.6.4 Advantages and disadvantages of existing metrics 46

4 Automated Sequential 3D Segmentation Based on Hierarchical Partitioning **49**

4.1 Introduction 49

4.2 Image volume partitioning 50

4.3 Feature labelling with fast marching and morphological closing 52

4.4 Automated segmentation 54

4.5 Results and validation 55

4.5.1 Evaluation on main dataset 55

4.5.2 Evaluation of special cases 57

4.5.3 Inter- and intra-observer variability 59

4.5.4 Remarks 60

4.6 Summary 61

5 Boundary Overlap Metrics for Segmentation Evaluation **63**

5.1 Introduction 63

5.2 Novel evaluation methods based on boundary overlap 65

5.2.1 Definitions 65

5.2.2 Parameterisation 70

5.2.3 Implementation and complexity 72

5.3 Experimental results 73

5.3.1 Synthetic data 73

5.3.2 Real scan data 82

5.3.3 Further tests 84

5.4 Comparison of boundary overlap methods 91

5.5 Summary 96

6 Path Reducing Watershed for the GPU **97**

6.1 Introduction 97

6.2 Path reducing watershed 98
6.2.1 Path reducing parallel watershed 98
6.2.2 GPU synchronous implementation 105
6.2.3 Block-asynchronous and balanced implementations 107
6.2.4 Implementation details 112
6.3 Results 113
6.3.1 Qualitative results 114
6.3.2 Synchronous vs. block-asynchronous 116
6.3.3 Comparison against the state-of-the-art 119
6.3.4 Parameters 121
6.3.5 Large 3D images 126
6.4 Hierarchical segmentation with PRW 129
6.4.1 Parallel waterfall transform with PRW 130
6.4.2 Parallel waterfall results 135
6.5 Summary 142

7 Fast Dashing Method for the GPU **145**

7.1 Introduction 145
7.2 Fast dashing method 146
7.2.1 Fast dashing parallel method 146
7.2.2 GPU synchronous implementation 149
7.2.3 Block-asynchronous implementation 152
7.2.4 Implementation details 155
7.3 Simple dashing method 156
7.4 Fast dashing method results 158
7.4.1 Synchronous vs. block-asynchronous 158
7.4.2 Parameters 161
7.4.3 Large 3D images 166

7.5 Simple dashing method results . . . 169
7.5.1 Synchronous vs. block-asynchronous . . . 169
7.5.2 Parameters . . . 172
7.6 Qualitative results . . . 174
7.6.1 Visual comparison . . . 176
7.6.2 Roughly segmented objects . . . 183
7.7 Comparison against traditional eikonal solvers . . . 185
7.8 Summary . . . 192

8 Results **195**
8.1 Experimental setting . . . 195
8.2 Performance . . . 198
8.3 The state-of-the-art context . . . 204
8.4 Discussion . . . 209

9 Conclusions **211**
9.1 Concluding remarks . . . 211
9.2 Future work . . . 213
9.2.1 Evaluation methods . . . 213
9.2.2 Watershed and waterfall . . . 214
9.2.3 Extended class of fast dashing methods . . . 214
9.2.4 Fully automated GPU-based segmentation . . . 215

A Knee MRI Segmentation: Qualitative Results **217**

B Software **233**
B.1 OxMedIS . . . 233
B.2 Evaluation methods . . . 234
B.3 Partitioning CUDA GPU programmes . . . 234

B.4 Front propagation CUDA GPU programmes 235
B.5 Other tools 236

C Publications **237**

C.1 Journal 237
C.2 Conference 237
C.3 Internal 238
C.4 Other presentations 239
C.5 Posters 240

Bibliography **241**

List of Figures

2.1 Colour (left), greyscale (middle), and binary (right) variants of the same image. 8
2.2 The greyscale image from Figure 2.1 and its gradient magnitude image. 8
2.3 The appearance of knee MRI sagittal scans with different sequences: T2-weighted (left) and PD-weighted (right). 10
2.4 The effect of windowing: a CT image slice without any windowing (left) and the same slice with windowing 40/400 (right). The image is visually more intelligible and the greyscale preserves detail in the image for the targeted tissue types. 11

3.1 The popularity trends of different segmentation approaches during the last 40 years 19
3.2 A sample input abdominal CT axial slice image at reduced resolution 64×64, its gradient magnitude image (top row), and the topographic terrain map visualisations of both images (bottom row). 21
3.3 The immersion (flooding) concept of the watershed transform. *Reproduced from [48].* 24
3.4 The concept of watershed transform by topographical distance (also called rainfalling). *Reproduced from [48].* 24

3.5 Two candidates for an initial front for fast marching within the anatomical structure. (a) A suitable initial front for a kidney. When developing outwards it approaches both the inner and outer borders of the kidney, thus shaping the organ. (b) An unsuitable candidate for a kidney. If expanding, it only approaches the outer border. If contracting, on the contrary, the inner boundary alone is shaped. 31

3.6 Fast marching with speed terms $F(\vec{x}) = \frac{1}{1+|\nabla I(\vec{x})|}$ (left) and $F(\vec{x}) = e^{-|\nabla I(\vec{x})|}$ (right). Closed white contours correspond to several level sets of $T(\vec{x})$ with a fixed level step Δt. Different coloured regions represent the positions for which $i \times \Delta t < T(\vec{x}) \leq (i+1) \times \Delta t, i \geq 0$. 34

3.7 (a) and (b) get the same size-based score; (a) and (c) get the same overlap-based score; (d) and (e) get the same boundary-distance-based score. 46

4.1 Four of the six layers in a 3D IPF, representing image partitions in decreasing detail level. Layers 2–5 in (a)–(d), respectively; finest layers 0–1 not shown. 53

4.2 Image partitioning and seed extraction from the resulting IPF 53

4.3 The generic structure of the segmentation procedure 54

4.4 The average ± standard deviation values of the considered metrics for the main dataset of kidneys A–I. Similarity metrics are in the first graph; difference metrics are in the second. 56

4.5 An extreme case of a tumourous left kidney (not part of the validation set) on which our algorithm yields only 30% Dice similarity 57

4.6 Machine segmented kidney (kidney M) with big tumour vs. its manually labelled ground truth which excludes the tumorous tissue 58

4.7 3D abdominal feature labelling by extrapolating the presented algorithm to other abdominal organs 61

5.1 Some of the fixed-size neighbourhoods of the points on the boundaries of the segmented regions. Directed Boundary Overlap considers only red (or only blue) neighbourhoods. Symmetric Boundary Overlap considers both red and blue neighbourhoods. 66

5.2 A simplified segmentation case with undefined local Precision in some of the boundary neighbourhoods. (a) shows ground truth pixels I–IV; (b) shows machine segmentation pixel V. The highlighted Moore neighbourhood of pixel I contains no machine segmentation pixels, hence the Precision metric is undefined there. 70

5.3 Results for synthetic rectangle images, comparing the segmented images against the ground truth, with all the considered metrics. Please note that DSC, TPVF, Prec scores are identical for all four images and their plots are not easily traced (similarity graph on the left). Likewise, DBD_G and DBD_M plots are almost identical with SBD. 76

5.4 Results for synthetic disc, ellipse, and star images, comparing the segmented images against the ground truths, with all the considered metrics. Please note that some overlap-based (boundary-distance-based) scores are equal in the left graphs (right graphs) for a few images and their plots are not easily traced. 77

5.5 Results for synthetic fractals. Fractals at steps 3, 6, and 9 of the recursion considered as ground truth, respectively. 81

5.6 Results for medical kidney scans, i.e. 3D regions in 3D images. Results for eleven kidneys (left or right) from nine 3D medical CT scans, between 1 and 38 slices each. Please note that some overlap-based scores are identical in the left graph for a few images and their plots are not easily traced. 82

5.7 (a) the ground truth and the automated segmentation of k10; (b) the ground truth and the automated segmentation of k11. In both cases the kidney is slightly undersegmented. 83
5.8 The ground truth and the automated segmentation of k1 captured from three different slices in (a)–(c). Both undersegmentation and oversegmentation can be observed in different slices. Random errors similar to our simulated salt-and-pepper are (just about) visible inside (c). 84
5.9 The ranking of the fractal images with different metrics with the ground truth set at f6. The images are sorted in the order of human ranking in the first graph; the second graph preserves the order of fractal recursion. 85
5.10 Comparison of SBD results for rectangle, ellipse, and fractal (ground truth at step 6) images and medical kidney 3D images using Moore neighbourhoods of radius 1 to 5. Please note that some SBD scores with different neighbourhood radii are equal or have very close values for a few images, and their plots are not easily traced. 86
5.11 Different boundary overlap metrics (their directional and symmetric variants) compared on the synthetic fractal images with ground truth f6. The first five graphs correspond to Boundary Dice, Jaccard, TPVF, TNVF, and Prec metrics. The sixth graph combines the symmetric variants of all five metrics. 92
5.12 Different boundary overlap metrics (their directional and symmetric variants) compared on the real kidney 3D images. The first five graphs correspond to Boundary Dice, Jaccard, TPVF, TNVF, and Prec metrics. The sixth graph combines the symmetric variants of all five metrics. 93

5.13 The performance of the conventional metrics against the proposed family of Symmetric Boundary Overlap metrics on the examples of the synthetic fractal images with ground truth at f6 and the kidney 3D images (excluding k12–k13). 95

6.1 Simple 2D neighbourhoods: Moore (left) and von Neumann (right). The inclusion of the central grid point in the neighbourhood is task dependent. 99

6.2 A simple 6×2 example with middle greyscale input image values detailing the procedure of path reducing watershed with von Neumann neighbourhood. Pixels with a change are highlighted in red; considered but not changed pixels are green. 102

6.3 The reduction of a path of length 22 with a reduction rate 2 in only $\lceil \log_3 22 \rceil = 3$ iterations. 104

6.4 A few shapes for minimal plateaux with state 3 pixels highlighted in grey. Von Neumann neighbourhood in the first row; Moore neighbourhood in the second. 105

6.5 The structure of shared memory allocated for each thread block. For a block of size 4×4, memory of size 6×6 is allocated. The threads on the boundary of the block are responsible for copying the adjacent extra values into the shared memory. 109

6.6 An image of size 128×128 (left); correct watershed line produced with $\mathrm{PRW}_{\mathrm{sync}}$ (middle); watershed line of $\mathrm{PRW}_{\mathrm{async}}$ tracking the 16×16 block boundaries (right). Example inspired by a similar one in [98]. . 110

6.7 $\mathrm{PRW}_{\mathrm{bal}}$ results on three sample images: Lena, an abdominal CT slice, and a knee MRI slice. The original greyscale images are in the first row; gradient magnitude images after minimal Gaussian blurring in the second row; the original images overlaid with watershed lines in white in the third row. 115

6.8 The execution times (average of ten runs per image) of the three variants of our watershed compared on the BSDS500 images. For clarity of presentation the images are sorted in increasing order of the $\mathrm{PRW}_{\mathrm{sync}}$ execution times. 116

6.9 The execution times and the number of global iterations of the non-minimal plateau resolution (algorithm step S2) of the three variants of our watershed compared on the BSDS500 images. 118

6.10 Images 1, 250, and 500, and the corresponding $\mathrm{PRW}_{\mathrm{bal}}$ results. Converted-to-greyscale images in the first row; overlays with the watershed lines in white in the second row. (Gradient magnitude was not constructed before running the watershed.) 118

6.11 Image 498 and the corresponding $\mathrm{PRW}_{\mathrm{sync}}$, $\mathrm{PRW}_{\mathrm{async}}$ and $\mathrm{PRW}_{\mathrm{bal}}$ results, depicted as overlays with the watershed lines in white. Apparent differences in the location of the watershed lines can be observed on the left and at the top of the image. 119

6.12 The comparison of the execution times (average of ten runs per image) of the DW watershed and our $\mathrm{PRW}_{\mathrm{bal}}$ implementation on the BSDS500 images, excluding six images on which the DW implementation deadlocks. The sums of the average execution times over the 494 images are 1,257.8308 ms and 895.298 ms respectively. Thus our algorithm shows an execution time improvement of more than 28.8%. 120

6.13 The comparison of the execution times of our $\mathrm{PRW}_{\mathrm{bal}}$ with different 2D block sizes on the BSDS500 images, two alternative visualisations. The first version encodes the runtime for each block width–height pair within a colourbar scale. The second version interpolates these runtimes into a surface (shown from two different viewpoints). The best performance of < 910 ms is achieved with $16\times8, 16\times10, 16\times12, 16\times14, 16\times16$ block sizes. 122

6.14 The comparison of the overall execution times (sum of average step S3 runtimes) and the number of global iterations of the path reduction (algorithm step S3) of $\mathrm{PRW}_{\mathrm{bal}}$ for different reduction rate values on the BSDS500 images. 124

6.15 The comparison of the execution times (average step S3 runtimes) and the number of global iterations of the path reduction (algorithm step S3) of $\mathrm{PRW}_{\mathrm{bal}}$ for different reduction rate values on the 3D test images. 127

6.16 The composition of the $\mathrm{PRW}_{\mathrm{bal}}$ execution times—data copy into the device, steps S1–S4, data copy from the device—on the 3D test images. 129

6.17 The watershed transform of an input 1D image followed by three recursive applications of waterfall—the image reconstruction and the watershed transform. The image boundaries and watershed lines are red. 133

6.18 Hierarchical segmentation of (gradient magnitude of minimally smoothed) sample images Lena, abdominal CT slice, and knee MRI slice for $NL = 6$. 136

6.19 The gradient magnitude of minimally smoothed images 1, 250 and 500 from BSDS500 followed by hierarchical segmentation with $NL = 7$. . 138

6.20 The layer generation execution times (lines 4–20 in Algorithm 8) for $NL = 7$ iterations of the hierarchical segmentation on the 3D test images. 139

6.21 The composition of layer generation execution times (lines 4–20 in Algorithm 8) for segmentation layers 1–3 of the hierarchical segmentation with $NL = 7$ on the 3D test images. Note, scale of values is different in each plot. 140

6.22 The composition of layer generation execution times (lines 4–20 in Algorithm 8) for segmentation layers 4–7 of the hierarchical segmentation with $NL = 7$ on the 3D test images. Note, scale of values is different in each plot. 141

7.1 The execution times (in ms) of the synchronous and block-asynchronous fast dashing method implementations on BSDS500. Two orderings of the 500 images illustrated: increasing order of PRW_{sync}, consistent with the results in the previous chapter; increasing order of $\text{FDM}_{\text{async}}$, preferred in this chapter for various FDM results. 159

7.2 The number of global iterations during the propagation step of FDM_{sync} and $\text{FDM}_{\text{async}}$ on BSDS500. 160

7.3 The execution times of $\text{FDM}_{\text{async}}$ with four speed options. 162

7.4 The execution times (in ms) of $\text{FDM}_{\text{async}}$ on BSDS500 with the partitioning label image varying among first seven layers of hierarchical segmentation with PRW_{bal}. Since the first layer is the watershed result, the first plot reported fully relies on the default experimental settings of FDM. The next six plots correspond to six waterfall layers of increasing region sizes and, as a result, faster $\text{FDM}_{\text{async}}$ execution times. 164

7.5 The overall execution times (sums of the 500 per image averages) of $\mathrm{FDM}_{\mathrm{async}}$ for different block sizes, two visualisations of the results. The first version encodes the runtime for each block width–height pair within a colourbar scale. The second version interpolates these runtimes into a surface (shown from two different viewpoints). 165

7.6 The execution times (in ms) of the synchronous and block-asynchronous simple dashing method implementations on BSDS500. Two orderings of the 500 images illustrated: increasing order of $\mathrm{FDM}_{\mathrm{async}}$, consistent with the results in the previous section; increasing order of $\mathrm{SDM}_{\mathrm{async}}$, preferred in this section for various SDM results. 170

7.7 The execution times of $\mathrm{SDM}_{\mathrm{async}}$ with four speed options. 173

7.8 The execution times (in ms) of $\mathrm{SDM}_{\mathrm{async}}$ on BSDS500 with the partitioning label image varying among first seven layers of hierarchical segmentation with $\mathrm{PRW}_{\mathrm{bal}}$. Since the first layer is the watershed result, the first plot reported fully relies on the default experimental settings of SDM. The next six plots correspond to six waterfall layers of increasing region sizes and, as a result, faster $\mathrm{SDM}_{\mathrm{async}}$ execution times. 175

7.9 FDM and SDM results on three sample images: Lena, abdominal CT slice, and knee MRI slice. The original greyscale images are in the first row; gradient magnitude images constructed after minimal Gaussian blurring in the second row; the original images overlaid with watershed lines in the third row. FDM and SDM default settings results on gradient magnitude images in the fourth and fifth rows, respectively. . 177

7.10 The effect of different FDM speeds on the arrival time images. Images 1, 250, and 500 (PRW_{sync} ordering) from BSDS500; corresponding gradient magnitude images constructed after minimal smoothing; original images overlaid with watershed lines. FDM arrival time images with speeds $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} in rows 4–7, respectively. 178

7.11 The effect of different SDM speeds on the arrival time images. SDM arrival time images with speeds $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} in rows 1–4, respectively. Original, gradient magnitude, and watershed overlaid images in Figure 7.10. 179

7.12 The effect of different hierarchical segmentation layers on FDM arrival time images. FDM results based on watershed partitioning (layer 1) in the first row; waterfall (layers 2–6) used for FDM in rows 2–6, respectively. Original, gradient magnitude, and watershed overlaid images in Figure 7.9. 180

7.13 The effect of different hierarchical segmentation layers on SDM arrival time images. SDM results based on watershed partitioning (layer 1) in the first row; waterfall (layers 2–6) used for SDM in rows 2–6, respectively. Original, gradient magnitude, and watershed overlaid images in Figure 7.9. 181

7.14 Objects segmented with a rough semi-automatic procedure. Original greyscale images in the first row; gradient magnitude images in the second row; overlay of L3 partitioning in the third row. Segmented objects in the fourth and fifth rows. 184

7.14 The BSDS500 execution times (in ms) of the fast marching and fast updating methods with different propagation speeds, compared against the fast dashing method with the same speeds using hierarchical partitioning layers 1–3. 187

7.15 FMM arrival time images for Lena, abdominal CT slice, and knee MRI slice sample images using default parameters. Original and gradient magnitude in Figure 7.9. 190

7.16 The effect of different FMM speeds on the arrival time images. FMM arrival time images with speeds $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} in rows 1–4, respectively. Original and gradient magnitude in Figure 7.10. 191

7.17 Objects segmented with the rough semi-automatic procedure, replacing FDM with FMM. Original greyscale images and gradient magnitude images in Figure 7.14. 192

8.1 The problems of the ground truth regions used (seen in columns 1 and 4), illustrated for a single knee MRI slice. Original greyscale slice is in the middle column. A slice of the ground truth and machine segmentation masks for fem2 (tib2) in first and second (fourth and fifth) columns. First row shows initial results, second row—updated results. 199

8.2 Single-slice femur segmentation results for M and N image slices from SKI10. Original greyscale slices are in the first and fourth columns. The ground truth (machine segmentation) masks in second and fifth (third and sixth) columns. 205

8.2 Single-slice femur segmentation results for P–W image slices from SKI10. Original greyscale slices are in the first and fourth columns. The ground truth (machine segmentation) masks in second and fifth (third and sixth) columns. 206

A.1 The femur and tibia segmentation results for fem3 and tib3, first four slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem3 (tib3) in first and second (fourth and fifth) columns. 218
A.1 The femur and tibia segmentation results for fem3 and tib3, last seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem3 (tib3) in first and second (fourth and fifth) columns. 219
A.2 The femur and tibia segmentation results for fem5 and tib5, first seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns. 220
A.2 The femur and tibia segmentation results for fem5 and tib5, second seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns. 221
A.2 The femur and tibia segmentation results for fem5 and tib5, third seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns. 222
A.2 The femur and tibia segmentation results for fem5 and tib5, fourth, and last, seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns. 223

A.3 The femur and tibia segmentation results for fem7 and tib7, first seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns. 224
A.3 The femur and tibia segmentation results for fem7 and tib7, second seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns. 225
A.3 The femur and tibia segmentation results for fem7 and tib7, third seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns. 226
A.3 The femur and tibia segmentation results for fem7 and tib7, fourth seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns. 227
A.3 The femur and tibia segmentation results for fem7 and tib7, last four slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns. 228
A.4 The femur and tibia segmentation results for fem9 and tib9, first slice. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns. 228

A.4 The femur and tibia segmentation results for fem9 and tib9, next seven slices (2–8). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns. 229
A.4 The femur and tibia segmentation results for fem9 and tib9, next seven slices (9–15). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns. 230
A.4 The femur and tibia segmentation results for fem9 and tib9, next seven slices (16–22). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns. 231
A.4 The femur and tibia segmentation results for fem9 and tib9, last seven slices (23–29). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns. 232

List of Tables

2.1 Different types of files and corresponding format extensions used in our work 16

3.1 (1) Overlap-based: Dice Similarity Coefficient (DSC) and Symmetric Volume Difference (SVD); Jaccard Similarity Coefficient (JSC) and Volumetric Overlap Error (VOE); True Positive (TPVF), True Negative (TNVF), False Positive (FPVF), and False Negative (FNVF) Volume Fractions; Precision (Prec) and Recall (Rec). (2) Size-based: Relative Volume Difference (RVD). (3) Boundary-distance-based: Hausdorff Distance (HD) and Average Symmetric Surface Distance (ASSD). 42

4.1 Match of our segmentation against gold standards per kidney with several similarity or difference metrics on the main dataset (A–I) and the special cases (J–M). Metrics DSC–RVD are measured in %; HD and ASSD—in mm. 57

4.2 Comparison of automated segmentation validation results against inter-observer variability for a kidney volume independently segmented manually by three operators (Op1, Op2, Op3) and automatically with our algorithm (Alg). First three rows validate manual results against each other; last three rows validate the labelling by our algorithm against manual results. Metrics DSC–RVD are measured in %; HD and ASSD—in mm. 59

4.3 Comparison of automated segmentation validation results to intra-observer variability for a kidney volume repeatedly (with a lapse of several months) segmented manually by two operators and automatically with our algorithm. First three rows validate intra-operator results and our algorithm for one operator (Op2 in the other experiment) against each other; last three rows present the results for the other operator (Op3 in the other experiment). Metrics DSC–RVD are measured in %; HD and ASSD—in mm. 60

5.1 Proposed boundary-overlap-based similarity metrics: directional and symmetric variants. 71

5.2 Synthetic dataset: ground truth images (left column) along with images for segmented regions incorporating size, location, shape, simulated salt-and-pepper errors in the segmentation, or combinations. The text refers to these images with a combination of one of the letters ‘r’, ‘d’, ‘e’, or ‘s’, and a number: r0 corresponds to the first—and ground truth—rectangle image; d12 is the last disc image, etc. 74

5.3 Sequence of synthetic fractal images with new isosceles triangles added at each recursive step (0–12). The three images at steps 3, 6, and 9—used as ground truth—are also presented in the left column. The text refers to these images with a combination of the letter 'f' and a number: f6 is the image at recursive step 6; f12 is the image at recursive step 12, i.e. the last image. 75

5.4 Value ranges and average values of SBD results over 13 star images using Moore neighbourhoods of radii 1–5 87

5.5 A small collection of simulated machine segmentation (M) and ground truth (G) image pairs with monotonically increasing region size based on ellipse and rectangle shapes. The image resolution is set at 512×512. 88

5.6 The impact of region size variation on metric scores using simulated segmentation images with ellipse and rectangle shapes. 89

5.7 The impact of image resolution variation on metric scores using simulated segmentation images with ellipse shape. A single pair of ellipse G and M images is considered at resolutions from 128×128 to 2048×2048. 90

6.1 The comparison of the execution times (in ms) of our PRW_{bal} with different 3D width×height×depth block sizes on the three 3D test images. The best performance for each image is highlighted. 125

6.2 The execution times (in ms, average of ten runs per image) of the three variants of our watershed compared on the 3D test images. The speedups of the block-asynchronous implementations over the synchronous procedure are reported in the last two rows. 128

6.3 The numbers of catchment basins in different layers of the hierarchical segmentation for a few 2D test images. Layer 7 is only required for one image out of six. 135

7.1 The execution times (in sec, average of ten runs per image) of the two variants of fast dashing compared on the 3D test images. The speedups of the block-asynchronous implementation over the synchronous procedure are reported in the last row. 166

7.2 The comparison of the execution times (in sec) of our $\mathrm{FDM}_{\mathrm{async}}$ with different 3D width×height×depth block sizes on the three 3D test images. The best performance for each image is highlighted. 167

7.3 The execution times (in sec, average of ten runs per image) of the two variants of simple dashing compared on the 3D test images. The speedups of the block-asynchronous implementation over the synchronous procedure are reported in the middle row. The last two rows include a comparison of $\mathrm{FDM}_{\mathrm{async}}$ and $\mathrm{SDM}_{\mathrm{async}}$. 171

7.4 The sums of the runtimes (in sec) of FMM, $\mathrm{FUM}_{\mathrm{async}}$, and $\mathrm{FDM}_{\mathrm{async}}$ (L1–L3) over BSDS500 with the four speed terms considered. 189

7.5 The execution times (in sec) of FMM, $\mathrm{FUM}_{\mathrm{async}}$, and $\mathrm{FDM}_{\mathrm{async}}$ with default parameter settings on the 3D test images. The smallest runtime for each 3D test image is highlighted. 190

8.1 Match of our segmentation against gold standards per femur with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm. 201

8.2 Match of our updated ('cropped') segmentation against gold standards per femur with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm. 201

8.3 Match of our segmentation against gold standards per tibia with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm. 202

8.4 Match of our updated ('cropped') segmentation against (modified to exclude fibula) gold standards per tibia with several similarity or difference metrics; tib1, tib2, tib3, tib7 affected. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm. 202

8.5 Match of our segmentation against gold standards per femur slice from SKI10 with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm. 205

8.6 Average femur bone test segmentation scores of the overall top ten performing submissions to SKI10 as of December 2017 and the reference inter-operator variability used in the scoring system. 208

Glossary

RR reduction rate. xiii, 103, 112, 116, 121, 123, 125–128

ASSD Average Symmetric Surface Distance. xxi, xxii, xxiv, xxv, 42, 45, 56, 57, 59, 60, 78–80, 83, 89, 90, 201, 202, 205, 207, 208, 234

BSDS500 Berkeley Segmentation Data Set and Benchmarks 500. xii–xvi, xxiv, 113, 114, 116, 118, 120–125, 137, 138, 158–161, 163, 164, 166, 169–172, 174–176, 178, 183, 187–189

CPU central processing unit. 15, 35, 121, 189

CT computed tomography. i, vii, ix, xi, xiii, xv, xvii, 3, 4, 9–12, 17, 21, 31, 33, 40, 50, 53, 55, 59, 61, 82, 83, 114, 115, 126, 135, 136, 139, 176, 177, 182, 190, 211, 233

DBD Directed Boundary Dice. ix, 66, 76, 79, 83

DBJ Directed Boundary Jaccard. 67

DBP Directed Boundary Precision. 68

DBTN Directed Boundary True Negative Fraction. 68

DBTP Directed Boundary True Positive Fraction. 68

DICOM Digital Imaging and Communication in Medicine. 12, 16, 50, 126, 196

DSC Dice Similarity Coefficient. viii–x, xxi, xxii, xxiv, xxv, 42–44, 56–60, 66, 68, 71, 76, 78–80, 83–85, 87, 89, 90, 92, 93, 198, 201–203, 205, 207

FDM fast dashing method. iv, v, xiv–xvii, xxiv, 4, 5, 146, 149, 150, 152–172, 174, 176–178, 180, 182, 183, 185, 187–190, 192, 193, 195–197, 199, 204, 208–210, 212, 214–217, 235

FIM fast iterative method. 188

FMM fast marching method. ii, iii, viii, xvi, xvii, xxiv, 29–35, 49, 50, 52–55, 58, 185, 187–192, 211, 235

FNVF False Negative Volume Fraction. xxi, 42, 44

FOZ first overflow zone. 27

FPVF False Positive Volume Fraction. xxi, 42, 44

FUM fast updating method. xvi, xxiv, 187–190, 235

GPU graphics processing unit. iii–vi, 2–4, 15, 19, 20, 26, 35, 97, 98, 105, 119, 121, 126, 143, 145, 149, 152, 157, 168, 174, 188, 189, 192, 195, 196, 212–215, 234, 235, 237, 238

GUI graphical user interface. 208, 216

HD Hausdorff Distance. xxi, xxii, xxiv, xxv, 42, 45, 56, 57, 59, 60, 78–80, 83, 84, 89, 90, 198, 201–203, 205, 207, 234

HU Hounsfield units. 9–11, 50, 51

IPF image partition forest. viii, 16, 28, 50–53, 55, 58, 139, 143, 203, 211, 214

JSC Jaccard Similarity Coefficient. x, xxi, 42, 44, 56, 67, 68, 71, 78–80, 83, 89, 90, 92, 93

mm millimetres. xxi, xxii, xxiv, xxv, 13, 45, 56, 57, 59, 60, 84, 91, 198, 199, 201–203, 205, 207, 208, 210

MRI magnetic resonance imaging. i, v, vii, xi, xiii, xv, xvii, 5, 9, 10, 12, 13, 17, 31, 51, 62, 114, 115, 135, 136, 176, 177, 182, 190, 197–200, 203, 204, 212, 215, 217

ms milliseconds. xii–xvi, xxiii, 116, 117, 120–123, 125, 128, 129, 142, 159, 160, 164, 170, 171, 174, 175, 187

PBM portable bitmap format. 16

PET positron emission tomography. 9

PGM portable graymap format. 16, 120

Prec Precision. ix, x, xxi, 42, 44, 56, 68–71, 75, 76, 78–80, 82, 83, 85, 88–90, 92–94

PRW path reducing watershed. iv, xi–xvi, xxiii, 4, 5, 98, 99, 101, 105–130, 132, 135, 143, 155, 158, 159, 164, 166, 174–176, 178, 182, 183, 193, 195–197, 204, 208, 212, 214, 215, 234, 235

RAVD Relative Absolute Volume Difference. 45

Rec Recall. xxi, 42, 44

RMSD Root-Mean-Square Symmetric Surface Distance. 207

RVD Relative Volume Difference. xxi, xxii, xxiv, xxv, 42, 45, 56–60, 75, 78–80, 83–85, 89, 90, 201, 202, 205

SBD Symmetric Boundary Dice. ix, x, xxiii–xxv, 66, 67, 70, 72, 73, 75, 76, 78–80, 82–87, 89–91, 94, 96, 198, 201–203, 205, 207, 213

SBJ Symmetric Boundary Jaccard. 68, 94

SBP Symmetric Boundary Precision. xxiv, xxv, 69, 70, 94, 201, 202, 205

SBTN Symmetric Boundary True Negative Fraction. 69, 91, 94

SBTP Symmetric Boundary True Positive Fraction. 69, 94

SDM simple dashing method. iv, v, xv, xvi, xxiv, 4, 156, 157, 169–177, 179, 181, 182, 192, 193, 212, 214, 215, 235

sec seconds. xxiv, 88, 128, 142, 143, 160, 163, 166, 167, 171, 172, 174, 189, 190, 208, 212

SM streaming multiprocessor. 168, 169

SVD Symmetric Volume Difference. xxi, 42–44

SVO Symmetric Volume Overlap. 43, 44

TNVF True Negative Volume Fraction. x, xxi, 42, 44, 56, 68, 71, 75, 78–80, 82, 85, 88–90, 92–94

TPVF True Positive Volume Fraction. ix, x, xxi, 42, 44, 56, 68, 71, 75, 76, 78–80, 82, 83, 85, 89, 90, 92–94, 207

VOE Volumetric Overlap Error. xxi, 42, 44

Chapter 1

Introduction

1.1 Motivation

The field of medical image segmentation is an actively evolving one with new segmentation approaches and applications emerging simultaneously with novel image types and improved quality. The main task of segmentation is to label and *accurately* separate the region of a specific anatomical structure of interest from the rest of the image. Given the natural variability in size and shape of different anatomical structures and commonly noisy imaging data, this is not an easy or straightforward task.

Currently, a vast amount of medical image segmentation approaches rely on learning techniques and expect large datasets of training data in order to perform well. The training data usually has the form of an extensive set of manually segmented 2D or 3D images. However, the production of accurate manual segmentations is a challenging task that requires many user hours. Thus, there is scope for efficient and accurate segmentation tools that can be used as automated segmentation procedures or can be utilised as part of the manual segmentation to speed up the process.

The importance of segmentation *efficiency* is also dictated by the scan resolutions improving over time and, hence, increasing image sizes. Medical scan sequences

of a dozen of megavoxels have become commonplace in recent years. A sequential algorithm that successfully segmented an average 3D image a few years ago may not be applicable on new larger images because of time and memory constraints[1].

Not of less importance is the *simplicity* of segmentation methods. On the one hand, clinicians do not usually like machine learning results which cannot be explained. In contrast, an easy to understand segmentation approach which allows to grasp the rationale behind the machine's decisions can be trusted more easily.

On the other hand, machine learning approaches like deep networks have gained massive popularity for medical image analysis in recent years. Deep learning models, e.g. U-Net and ResNet, have triumphed in well known medical image segmentation challenges. Meanwhile, fully automatic machine learning methods rely on training data and simple approaches that provide a reasonable initial segmentation can be incorporated to facilitate the production of training data.

1.2 Goals

This thesis presents medical image segmentation techniques that combine accuracy with efficiency, while maintaining simplicity. In order to achieve that, we consider graphics processing unit (GPU) hardware and possibilities of developing massively parallel general computing algorithms to be executed on such devices. In recent years, with the advent of affordable and programmable GPU devices and with the development of easy-to-use GPU programming frameworks and toolkits, this trend is gaining more attention.

The simplicity of any proposed techniques is also important to guarantee applicability to different types of data. Thus, we intend to maintain the proposed methods as data-independent as possible. In order to achieve this, we minimise the use of

[1]Our own algorithm presented in Chapter 4 works on images of 6–7 megavoxels, but fails if we double the size of the image, i.e. around 12 megavoxels.

domain-specific knowledge in our methods, where possible.

An additional aspect of interest is the validation of medical image segmentation results. We survey the current state of this field and make a contribution in the form of novel evaluation methods. This is done to show that these methods, once adopted by the research community, will make the segmentation evaluation process more informative and assess the segmentation quality in a fair manner.

1.3 Contribution

The original contributions made by this thesis are as follows:

1. a fully automated sequential 3D procedure for the segmentation of kidneys from abdominal computed tomography (CT) scans;
2. a family of boundary-overlap-based metrics for the evaluation of segmentation results against ground truth;
3. an efficient parallel watershed algorithm designed for GPU implementation;
4. a parallel waterfall algorithm for hierarchical partitioning designed for GPU implementation;
5. a front propagation method for partitioned domains designed for GPU implementation and its simplified variant;
6. detailed analysis and comparison of the implementation types of the proposed GPU algorithms and their parameters;
7. a simple semi-automatic medical image segmentation procedure based on these GPU algorithms.

The remainder of this thesis is organised as follows.

Chapter 2 specifies the imaging and computing hardware used; describes the image types and formats; briefly introduces relevant medical image modalities.

Chapter 3 provides a review of the existing relevant literature. Watershed and waterfall transforms are described, main algorithms implementing these are referenced. Next, front propagation using an eikonal equation is described along with algorithms and parameters. Finally, the task of medical image segmentation validation is presented, followed by a list of well-known existing evaluation methods.

Chapter 4 describes the proposed automated sequential 3D procedure for the segmentation of kidneys using hierarchical partitioning and front propagation. The procedure is validated on a small dataset of abdominal CT scans.

Chapter 5 presents novel segmentation evaluation metrics based on the concept of boundary overlap. These metrics are compared against well-known existing metrics using a set of synthetic 2D images and real 3D scan data.

Chapter 6 introduces a novel parallel watershed algorithm, called path reducing watershed (PRW), for implementation on GPU. Three different implementations are presented and compared on a large set of images. Furthermore, a hierarchical segmentation framework based on waterfall transform using PRW is also introduced and its efficiency estimated.

Chapter 7 introduces the fast dashing method (FDM) for front propagation in a partitioned domain, proposed for GPU implementation. A further simple dashing method (SDM) approach is described for propagation with more relaxed and nonexpensive rules. A large dataset is used to estimate the efficiency of these methods and the impact of their parameters.

Chapter 8 presents experimental semi-automatic segmentation results based on PRW and FDM.

Chapter 9 includes concluding remarks and briefly describes possible future work.

Appendix A visualises (input greyscale images and segmentation masks) some of the 3D knee magnetic resonance imaging (MRI) segmentation results.

Appendix B summarises the different pieces of software written in the context of this thesis.

Appendix C lists my current publications relevant to this thesis.

Chapter 2

Hardware and Data

2.1 Image types

2.1.1 Digital raster images

Digital raster images are data structures, representing rectangular grids of image value elements. These elements are called *pixels* for 2D images and *voxels* for 3D images. Depending on the set of possible values that a pixel[1] can take, images can be classified as

- binary
- greyscale
- colour

Each pixel in a *binary* image gets one of the two possible values—black and white. In *greyscale* images, the pixels get a value from a X-value range $\{0, 1, \ldots, X-2, X-1\}$, which define different intensities or shades of grey. Black, or the weakest intensity, is at the lower end of the range, i.e. 0. And the other end of the range, i.e. $X-1$,

[1]The term *pixel* can be used as a common term to refer to image value elements in both 2D and 3D images.



Figure 2.1: Colour (left), greyscale (middle), and binary (right) variants of the same image.



Figure 2.2: The greyscale image from Figure 2.1 and its gradient magnitude image.

corresponds to white. In *colour* images, a pixel comprises a few colour channels and gets an intensity value for each of those channels. The most common representation of a pixel in a colour image is a triplet $\{r, g, b\}$, where $r, g, b \in \{0, \ldots, X-1\}$ are the intensities of the red, green, and blue colour channels. Figure 2.1 illustrates how the binary, greyscale, and colour images differ on the example of the same image.

In an uncompressed representation of the image, a fixed amount of memory is required per pixel. In the case of binary images, a bit is sufficient to store and differentiate the values for black and white. Common representations of greyscale images include 8-bit and less frequently 16-bit pixels with corresponding 256- and 65536-value ranges. Depending on the image format and the available graphics hardware, the colour images may have different representations, including, for instance, 24-bit pixels with 8 bits per colour channel or even 8-bit pixels with 2–3 bits per colour channel.

2.1.2 Gradient magnitude images

A gradient magnitude image is a new greyscale image, the size of which equals the size of the input greyscale image and each pixel is the magnitude of the colour gradient vector of the corresponding pixel in the input image. Figure 2.2 illustrates the gradient magnitude image constructed for the greyscale image from Figure 2.1. Lighter shades of grey in the gradient magnitude image correspond to pixels with high contrast or local variability. These in turn usually correspond to boundaries of objects depicted in the image. A lot of computer vision and medical image segmentation techniques rely on gradient magnitude images in order to identify object boundaries.

2.2 Medical images

Medical imaging covers different types of image representations, called *modalities*, which describe the image acquisition techniques and the interpretation of the resulting images. Well-known image modalities include CT, MRI, positron emission tomography (PET), ultrasound, etc. Our research revolves around CT and MRI.

2.2.1 CT

CT images are X-ray based stacks of 2D slices, which visualise consecutive thin body sections and make up a 3D image volume. After Sir Godfrey Hounsfield—the scholar introducing the first CT machine—the value scale used in the images is called Hounsfield units (HU). This value is determined by the formula

$$H = \frac{\mu_{tissue} - \mu_{water}}{\mu_{water} - \mu_{air}} \times 1{,}000,$$

where μ is the radiodensity of the material. For instance, the HU value for water is 0, while $[100, 300]$ is the interval for soft tissue. The scale for human body scans usually

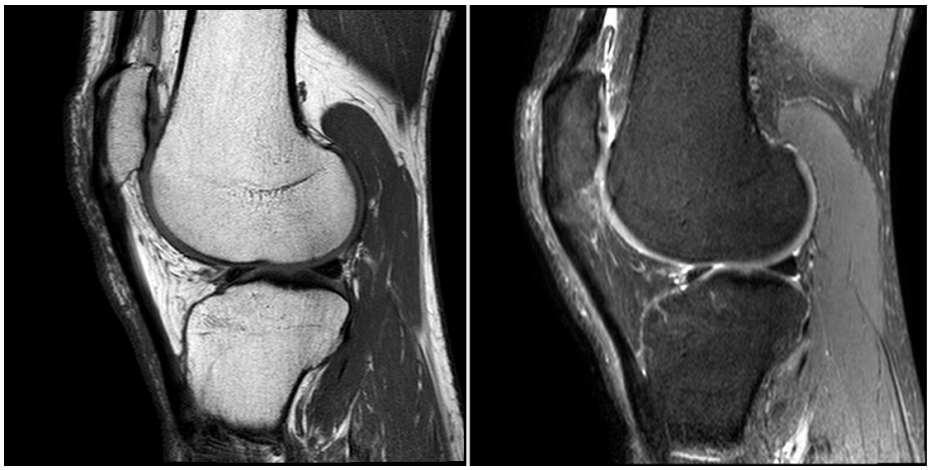


Figure 2.3: The appearance of knee MRI sagittal scans with different sequences: T2-weighted (left) and PD-weighted (right).

spans from −1,000 for air up to around 3,000 for bone.

2.2.2 Micro-CT

Micro-CT is an imaging technique to produce X-ray based images at very high resolutions in the micrometre range. Just like ordinary CT, it relies on the HU value scale. In addition to medical applications, micro-CT can be applied to examine the interior of various substances, from food to fossils to small electronic components.

2.2.3 MRI

MRI technique is based on magnetic fields, radio waves, and field gradients, and generates 3D images consisting of 2D slices. Different types, or sequences, of MRI exist, based on the particular settings of the scanner, and these highly affect the tissue appearance in the images. Figure 2.3 illustrates how knee MRI sagittal[2] scans can have different appearances if acquired with varying imaging sequences. The femur and the tibia bones are visible located centrally in the top and bottom halves of both images. In the first, T2-weighted slice, these bones appear as large light grey regions.

[2]Sagittal scans use a plane for image slices that divides the body into left and right halves. Coronal scans use the front–back division; axial scans use the top–down division.

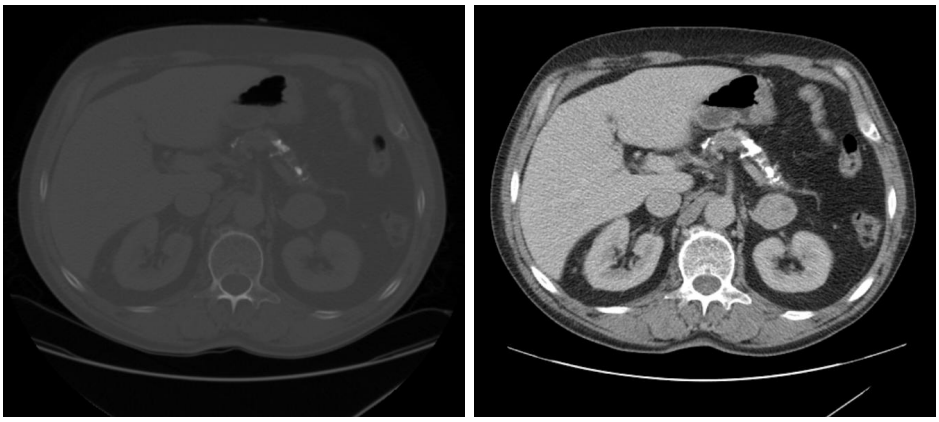


Figure 2.4: The effect of windowing: a CT image slice without any windowing (left) and the same slice with windowing 40/400 (right). The image is visually more intelligible and the greyscale preserves detail in the image for the targeted tissue types.

In the second, PD-weighted slice, the bones are much darker, while the cartilage along the touching bone boundaries is brighter.

2.2.4 Windowing

The range of HU values for CT images is very large and the raw images are not suitable for displaying. A common technique for overcoming the problem and at the same time preserving the needed information is called *windowing*. The idea is to project the relevant interval of Hounsfield values on the greyscale 0–255 and use the resulting image. The procedure is

1. choose an interval $[L - W/2, L + W/2] \subset [-1{,}000, 3{,}000]$,
2. project every HU value less than $L - W/2$ to 0 and every value more than $L + W/2$ to 255,
3. for any value within $[L - W/2, L + W/2]$, project them proportionally:

$$p' = \left[255 \times \frac{p - (L - W/2)}{W}\right] = \left[255 \times \frac{p - L}{W} + 127.5\right].$$

L, *window level*, is the centre of the interval, W, *window width*, is the length

of the interval. Depending on the body part being studied and the type of tissue examined, the windowing interval can vary. In particular, for consideration of the abdominal soft tissue organs the standard interval is $L = 40, W = 400$, which is used throughout all of our experiments in Chapter 4. The difference between a raw and a windowed image is demonstrated in Figure 2.4. This is an abdominal CT slice, where the vertebra bone is visible in the lower middle part of the image, the kidneys are to either side of it, and the liver spans a big portion of the left half of the image.

Note, that the windowing technique is not limited to CT or micro-CT images. MRI images are in practice also windowed for visualisation. However, since the greyscale appearance is affected by the MRI sequence and does not assume a unique interpretation, there are no general predefined windowing parameters for these. A well-fitted windowing range is usually established for each image experimentally.

2.2.5 Representation

Digital Imaging and Communication in Medicine (DICOM) is the international standard for medical images and related information. In addition to the actual 2D image, a DICOM file contains some metadata about the image in the form of pairs of tags and corresponding values. Pixel values are in practice scaled from their original ranges to be stored in DICOM files on disc, and rescaled back to original value range when interpreted in memory. Common representations include signed or unsigned 8-bit or 16-bit integer numbers.

2.2.6 Scanner hardware

The abdominal CT image volumes, used in Chapter 4, were kindly provided by medical practitioners working in the Churchill Hospital, Manor Hospital, and John Radcliffe Hospital, Oxford. The data included CT scans of the patients who underwent treatment in those hospitals. The scanners used were LightSpeed Ultra and Light-

Speed QX/i by GE MEDICAL SYSTEMS and Sensation 16 by SIEMENS. The scans are axial, i.e. the scan plane divides the body in a top–down manner. The pixel intra-slice spacing is isotropic and ranges between 0.68 millimetres (mm) and 0.78 mm. The slice thickness is 5 mm. The dataset contains nine abdominal CT sequences, all of which are acquired with contrast agent administration.

The MRI dataset we used was obtained from the Botnar Research Centre (part of the Nuffield Department of Orthopaedics, Rheumatology and Musculoskeletal Sciences at Oxford). The data consists of nine knee MRI sequences taken with a Philips Achieva TX 3.0T MRI system, with a T1W contrast enhanced Turbo Spin Echo (TSE) sequence, with no fat suppression. The scans are sagittal, meaning that the slices are taken in the plane that divides the body into right and left sides. The pixel spacing is around 0.3 mm by 0.3 mm, and the slice thickness is 2.5 mm for eight and 3 mm for one sequence (hence, the data is anisotropic). The scans were taken at the Botnar Research Centre in the Nuffield Orthopaedic Centre using the standard sequence employed in clinical practice within the UK's National Health Service. All scans were right knees of females with an average age of 49 (range: 31–63 years). Patients had been diagnosed with either a meniscal tear or patello-femoral arthritis.

Finally, individual slices from ten different knee MRI sequences were obtained from the SKI10 grand challenge [51], a world-wide competition in knee segmentation associated with MICCAI, a top international imaging conference. All images were acquired in the sagittal plane with a pixel spacing of 0.4 mm by 0.4 mm and a slice thickness of 1 mm. Further characteristics of the SKI10 data can be found in [51, 96].

2.2.7 Ethics

All scans were anonymised, were subject to ethical approval for use in research, and were covered by the University's ethics protocols.

A crucial aspect to consider while working with real-world data is to correctly deal

with ethical issues. In this case, we had to make sure that our data does not carry any information which can potentially be used illegally. For that reason, the data passed several phases of anonymisation. Any information about the identity of the patients, such as their names, age, birth dates, etc., or the names of the physicians were removed from the scans.

2.3 Input and output image formats

At the different stages of processing, the images have different formats in terms of pixel values. Depending on the requirements of the current implementation, pixel value ranges may be converted (for instance, from colour to greyscale) and/or scaled (for instance, from 16-bit to 8-bit numbers) to a required range as a pre-processing step.

In Chapters 4, 6, and 7 the following formats are used:

- *input*: greyscale range 0–255;
- *label*: extended greyscale range, where each level is used as label for one region, i.e. pixels in the same partitioning region have equal values;
- *arrival time*: greyscale range 0–255 or 0–65535, linearly mapped from real-valued arrival time arrays;
- *segmentation*: binary with the two values corresponding to the foreground and the background.

The *input* image can be formed by applying windowing to a medical scan or by direct conversion of an image into an 8-bit raster image format. Gradient magnitude images are also classified as *input*.

The *label* images are obviously **not** intended for visualisation. The chosen format is just used to store the information about the image partitioning and to use it in

further processing. Note, that the range of values for such images can be very large—up to the scale of the number of pixels in the image—if the partitioning contains a plethora of tiny regions.

The *arrival time* images can be used to visualise the front propagation patterns. Also, these can be thresholded to produce *segmentation* images. The *segmentation* images, in their turn, can be used in evaluation to compare machine segmentation to ground truth, as shown in Chapter 5.

2.4 Computing hardware

We have run all our parallel implementation tests on an x86-64 Linux machine with an NVIDIA GeForce GTX 960M graphics card of 640 CUDA cores and 2GB memory. The compute capability[3] of this GPU is 5.0. For consistency reasons, all experiments were carried out using CUDA 8.0.61, NVIDIA driver 375.74. Although new versions of both the CUDA toolkit and the NVIDIA driver became available halfway through our tests, we maintained the original versions to enable fair comparison between different parts of our results.

The central processing unit (CPU) is an Intel Core i7-6700HQ at 2.60GHz with 8 threads and 4 physical cores. The memory and the swap are 16GB each.

Every sequential CPU and parallel GPU execution time reported in Chapters 6 and 7 is an average over ten runs.

2.5 File formats

Table 2.1 summarises the different types of files and corresponding format extensions used to store input data, intermediate information, and final results.

[3]The compute capability identifies the features supported by the GPU hardware.

Table 2.1: Different types of files and corresponding format extensions used in our work

Type	Format	Description
medical scan slice	dcm	DICOM, contains a single 2D image slice along with metadata in the format of (*tag*, *value*) pairs
volume choice file	vcf	small text file describing a chosen medical image volume and its subvolume of interest for experiments, originally introduced for *millipede* [43]
image partition forest	ipf	large text file with a full description of an image partition forest (IPF) data structure constructed for a chosen image subvolume [133]
multi-feature selection	mfs	mid-size text file describing multiple labelled regions in the form of sets of nodes in the IPF data structure [43]
single label mask	pbm3	mid-to-large text file that mimics portable bitmap format (PBM) and describes the mask of a single label in a 3D image volume with known voxel spacing
synthetic 2D mask	pbm	PBM, foreground–background mask for synthetic 2D segmentation stored as a digital binary image
greyscale image	pgm	portable graymap format (PGM), *input, label, arrival time*, or any other greyscale image used in the work

Chapter 3

Literature Review

3.1 Medical image segmentation

3.1.1 The task

In a clinical or clinical research context relying on CT or MRI imaging data, the shapes of anatomical structures of interest are typically found from single slices, using commercial software (such as Mimics [9] or Simpleware [12]). These shapes are segmented from individual images manually: this involves setting a threshold then going through slice by slice and fixing regions which have not been detected correctly. Processing a set of scan slices takes many user hours. The uptake of computer-based segmentation in clinics and in research fields which process large volumes of visual data depends crucially on shortening this user time. Open-source alternatives to commercial software include ITK-SNAP [7] and 3D Slicer [1].

The task of *medical image segmentation* involves the processing of 3D image data to find the exact shape and location of anatomical structures. For a specific organ or tissue of interest, this can be formalised as a process that produces a mask of the whole image, where the foreground pixels correspond to the target organ or tissue, and the background pixels denote everything else. If there are multiple anatomical

structures of interest, a label image can be produced, where each possible pixel value corresponds to one of the anatomical structures of interest or to background (to represent anything but the target structures).

This is similar to the general *image segmentation* concept from computer vision. However, there is also a significant difference. The different labelled regions in medical image segmentation correspond to known *semantic*, or meaningful, concepts, like blood vessel, skin, bone, etc. In general image segmentation, on the other hand, the images are divided into multiple regions based on some criteria of intra-region homogeneity and inter-region heterogeneity. For the results of general image segmentation, it is not necessarily possible to find a unique semantic label for each region.

Traditionally, the term *segmentation* is used to refer to both tasks [113, 15]. But this may cause confusion in certain settings. Thus, when a clear separation of terms is required, the task of medical image segmentation will hereafter be referred to as 'labelling', and the task of image segmentation in the general computer vision meaning will be called 'partitioning'. Whenever it is not obvious from the context which task is being discussed, we rely on these alternative terms.

3.1.2 Segmentation approaches

The field of medical image segmentation has attracted a vast amount of research efforts in the last decades. Various segmentation approaches and specific procedures for the labelling of distinct anatomical structures from images of concrete modalities have been proposed, tested, and analysed.

According to several reviews on medical image segmentation algorithms [110, 73, 91, 65], segmentation approaches can be grouped into multiple classes in different ways. One possible classification includes histogram-based (thresholding), edge-based (edge detection), region-based (region growing), morphological (watershed), graph-based (grow cuts), partial-differential-equation-based (level sets, fast march-

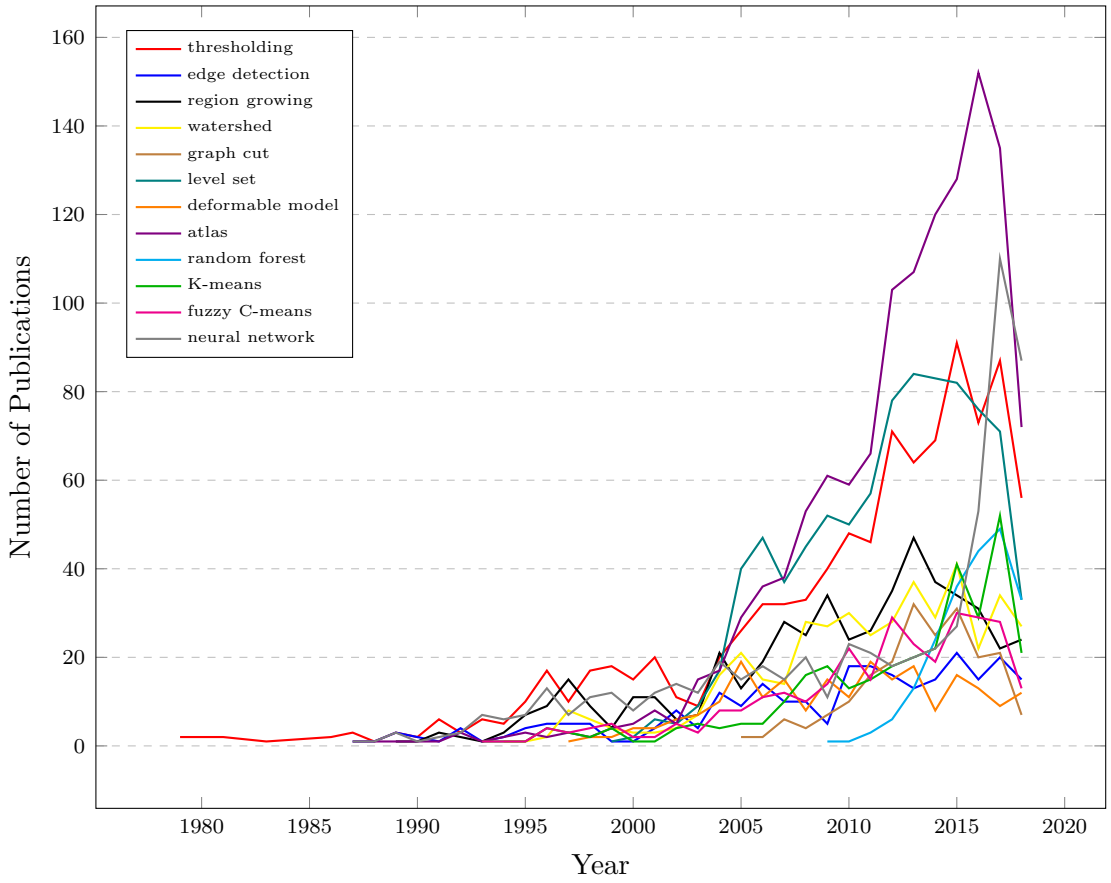


Figure 3.1: The popularity trends of different segmentation approaches during the last 40 years

ing), model-based (active shape models, deformable models), atlas-based, supervised learning (artificial neural networks, random forests), unsupervised learning (K-means, fuzzy C-means, reinforcement learning) methods.

Figure 3.1 illustrates how popular different segmentation approaches have been during the last 40 years. The data was extracted from the PubMed [11] search engine using the pattern `segmentation AND ‘‘approach’’`, where `approach` is a proxy for any of the terms in Figure 3.1. As the data shows, the most popular methods in the last decade have been atlases, level sets, and thresholding. Moreover, neural networks have been gaining large popularity since 2015.

Earlier attempts at parallelisation of established segmentation methods on GPU devices and at novel fully GPU-based medical image segmentation procedures show

promising results [94, 58, 59]. Now good accuracy semi-automatic or fully automatic segmentation can be achieved in significantly faster runtimes compared to sequential approaches, and high-resolution 3D image volumes can be processed in time-restricted situations.

In [113], Smistad et al. review medical image segmentation on GPUs. They identify five key factors, including data parallelism, thread count, branch divergence, memory usage, and synchronisation, for the suitability of different segmentation algorithms for parallel implementation on GPU devices, and discuss commonly used image segmentation methods based on these criteria. *Thresholding, region growing, morphology (with structuring elements), watershed, active contours, level sets, atlas-based, statistical shape models, Markov random fields and graph cuts*, and other segmentation methods are considered in the framework of parallel GPU implementation and ranked.

3.2 Watershed transform

3.2.1 Description

The watershed is a greyscale image transformation from mathematical morphology that partitions the image into multiple small regions based on a topographic terrain map interpretation of the image. The name *watershed* is derived from the corresponding geological term that, as stated in Oxford Dictionaries, denotes 'an area or ridge of land that separates waters flowing to different rivers, basins, or seas'. The watershed transform is commonly used in image processing and is typically applied on gradient magnitude images (see Section 2.1.2), constructed from input greyscale images.

This image can be interpreted as a topographic terrain map, i.e. as a landscape where high greyscale pixels have higher altitude, or elevation, and small greyscale pixels have lower altitude. The watershed transform separates the image into disjoint

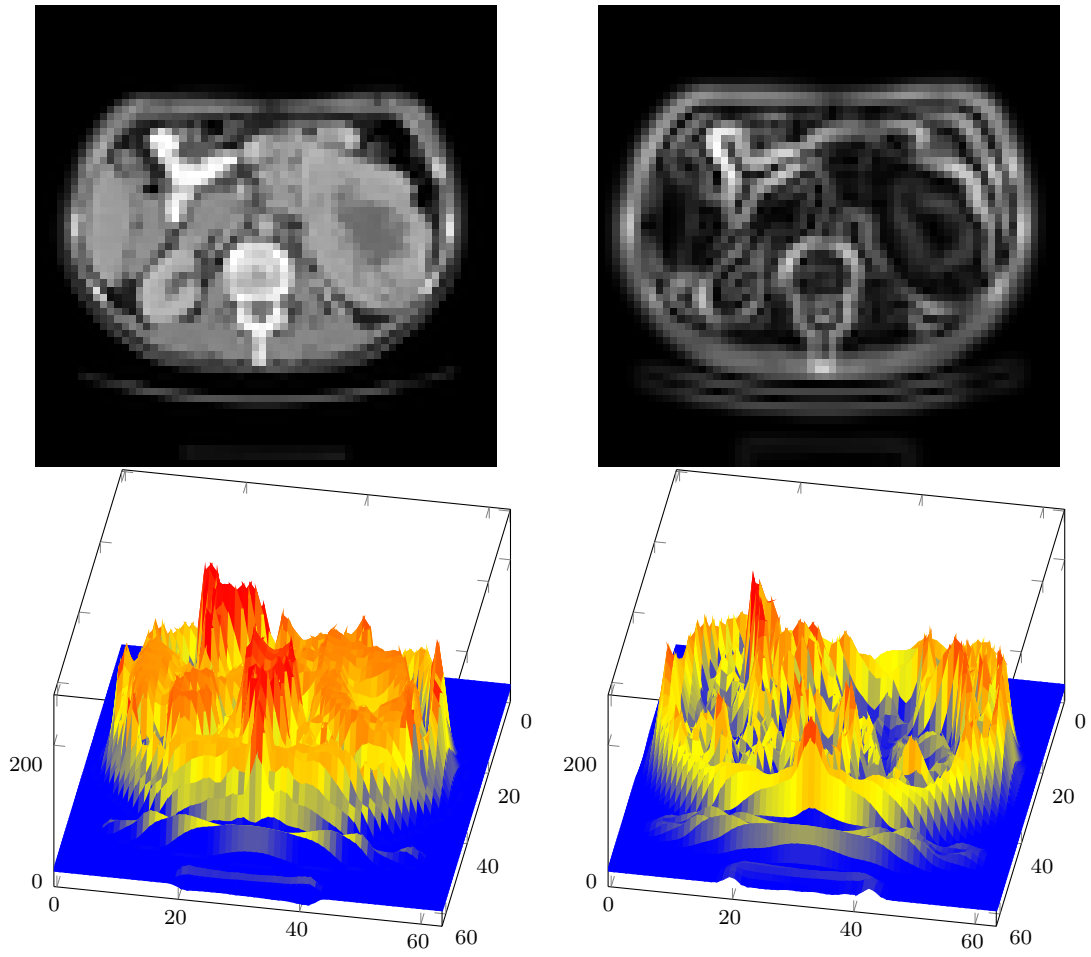


Figure 3.2: A sample input abdominal CT axial slice image at reduced resolution 64×64, its gradient magnitude image (top row), and the topographic terrain map visualisations of both images (bottom row).

regions called *catchment areas or basins*. These regions are formed around regional minima in the topographic terrain. The boundary pixels, separating the different catchment basins are called *watershed lines*, or *watersheds*, and are reminiscent of dams that are erected where different bodies of water filling the terrain would otherwise meet.

Figure 3.2 illustrates a sample input abdominal CT axial slice image at reduced resolution 64×64, the corresponding gradient magnitude image, and visualises both as topographic terrain maps, where blue corresponds to the lowest altitude and red denotes the highest altitude. When the watershed transform is applied to the gradient

magnitude image, the boundaries of the different catchment basins are placed at pixels with high gradient magnitude values. These, in turn, correspond to boundaries of regions of interest—in the particular case, different organs or tissue types. Thus, the watershed transform partitions the image into regions of interest.

3.2.2 Problem of over-partitioning

While the watershed transform is a well-known technique used in many applications of image processing, including edge detection, digital image segmentation, medical image segmentation [101, 112], its main drawback of over-partitioning is inherent to this procedure. It is very sensitive to local greyscale variations in the image and produces too many tiny catchment basins. Traditionally there are three common approaches for overcoming this issue:

- heavy smoothing,
- watershed from markers,
- merging techniques.

Pre-processing smoothing techniques [23], like the Gaussian blur or the anisotropic diffusion filtering [95], are applied before and/or after gradient magnitude image construction to eliminate insignificant local variation. As a result, some of the local minima are smoothed out and the number of catchment basins in the watershed result of the smoothed image goes down. Although smoothing significantly reduces the number of catchment basins, this is usually not enough. Hence it is commonly combined with either watershed from markers or merging techniques.

Watershed from markers [88, 22, 101] expects markers for the significant catchment basins in the image before running the watershed transform. Then the labels from these markers are flooded into the whole image terrain producing larger catchment basins, the number of which equals the number of the markers. These markers

can be defined by the user or they can be determined automatically: with morphological operators like opening and closing or with the application of domain-specific knowledge.

The alternative approach of post-processing the watershed transform eliminates non-significant catchment basins by merging them. A number of merging techniques using statistical information, clustering, catchment basin depth, or other features have been proposed [101]. Another common merging technique is hierarchical segmentation achieved with the *waterfall transform* [19, 48]. A further discussion of the waterfall transform follows in the next section.

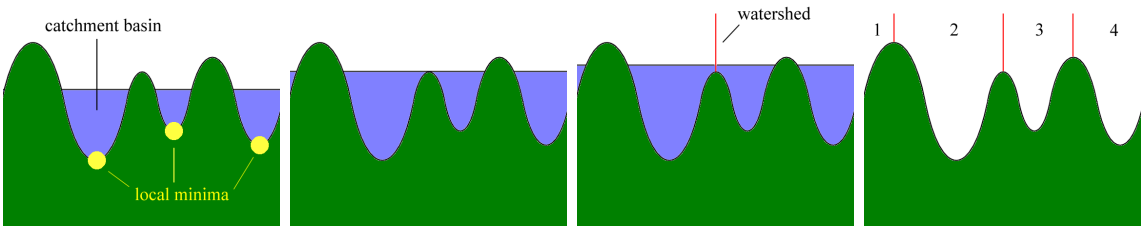
3.2.3 Plateaux problems

An important pitfall for various watershed algorithms are the plateaux—connected components of constant greyscale value in the image. Some algorithms can execute only on lower-complete images, i.e. images without non-minimal plateaux. Thus, a lower-completion transform needs to be applied on an image before the application of watershed [100]. Other watershed algorithms incorporate the resolution of plateaux but this step is usually one of the most inefficient parts of these algorithms.

3.2.4 Algorithms

A number of different definitions and corresponding algorithms exist for the watershed transform. A detailed discussion of watershed formal definitions and a survey of existing algorithms is given by Roerdink and Meijster [100]. They consider two main classes of watershed algorithms based on the two main definitions of watershed: by immersion [122] and by topographical distance [89].

Watershed by immersion (flooding) algorithms rely on the concept of gradually immersing the terrain pierced at the regional minima into water. Progressively, the level of water rises covering catchment basins. The points where the bodies of water



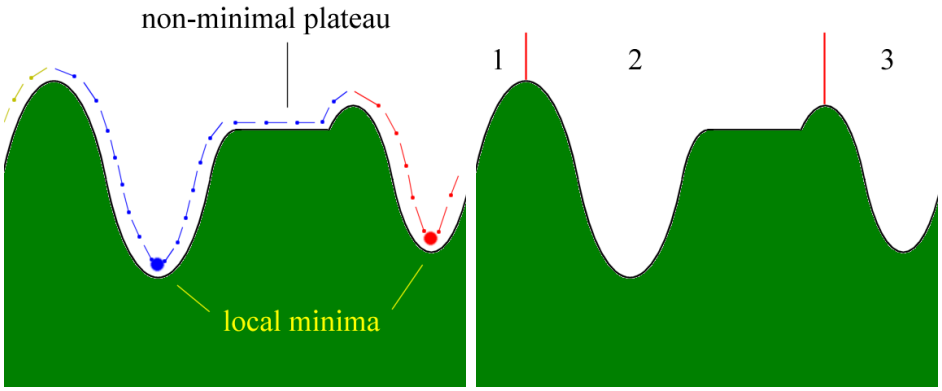
(a) beginning the flooding

(b) two catchment basins meet

(c) building a watershed at the join point

(d) the division of the landscape into catchment basins

Figure 3.3: The immersion (flooding) concept of the watershed transform. *Reproduced from [48].*



(a) finding a path of steepest descent from each point

(b) the division of the landscape into catchment basins

Figure 3.4: The concept of watershed transform by topographical distance (also called rainfalling). *Reproduced from [48].*

from different sources meet are labelled as watersheds separating the corresponding catchment basins.

Watershed by topographical distance uses the idea that, from any point in the terrain, a drop of water would flow to the 'nearest' minimum. For each position, in order to identify the catchment basin that it belongs to, the direction of maximal, or steepest, descent needs to be followed. If there are more than one such paths leading to different minima, then the source position belongs to a watershed rather than any catchment basin.

Figures 3.3 and 3.4 illustrate the concepts of watershed by immersion and by

topographical distance. These illustrations are reproduced from [48].

The formal definitions for watershed by immersion and by topographical distance assume a separate class of watershed pixels. This means that a group of pixels separating the different catchment basins should be part of the watershed and marked with a special label, distinct from the labels of all the catchment basins. Some watershed algorithms, however, produce a full partitioning of the image into catchment basins by propagating to each watershed pixel the label of one of its neighbouring catchment basins, see [100]. Watershed based on a local condition [24] modifies the definition of watershed by topographical distance to classify each pixel as part of a catchment basin. More definition types for watershed include image foresting transform [72] based on a max-arc path cost function and the watershed cut [34].

The survey by Körbes and Lotufo [61] groups the watershed algorithms based on two algorithm paradigms. Breadth-first watersheds and depth-first watersheds are separated with the main criterion of the visitation order of pixels. These paradigms are not tied to a specific definition of watershed: different watershed algorithms with the same algorithm paradigm may be based on different watershed definitions.

Well-known breadth-first watersheds include immersion by Vincent and Soille [122], hierarchical queue by Beucher and Meyer [22], shortest-path and hill climbing by Meyer [89], image foresting transform by Lotufo and Falcão [72], etc. Depth-first watersheds list connected components by Bieniek and Moga [24], union-find by Meijster and Roerdink [86], toboggan by Mortensen and Barrett [90], chain code by Sun, Yang, and Ren [114], shortest paths by Osma-Ruiz et al. [93], watershed cut by Cousty et al. [34]. An up-to-date review of more recent watershed algorithms is provided by Romero-Zaliz and Reinoso-Gordo [101].

Sometimes in recent publications [101], the terms *by immersion* and *by rainfall* are used to classify watershed algorithms (as opposed to watershed definitions) and roughly correspond to the classes of breadth-first and depth-first watersheds, respec-

tively.

3.2.5 Parallel watershed

Roerdink and Meijster [100] also provide a detailed review of early algorithms for parallel watershed on distributed memory and shared memory architectures. Recent watersheds with distributed memory parallelism [115, 130], with shared memory parallelism [126, 120], and with GPU implementations [57, 63, 54, 97] are briefly reviewed in [101].

The advent of affordable programmable GPU devices sparked a new interest in parallelisation of existing, mostly sequential, image processing algorithms, including the watershed transform. The reason for this is the massive parallelism that GPU devices offer and the straightforward analogy that can be drawn between the GPU threads of execution and the grid structure of an image. Thus, a sequence of new watershed algorithms were proposed, which modelled the watershed transform in terms of parallel application of the same procedure on all the pixels or voxels in a 2D or 3D image, respectively.

Pioneering papers like [94, 57, 59, 127] mainly developed watershed procedures using the flooding pixel order, i.e. breadth-first watersheds, where regions are initialised in regional minima and expanded to cover the whole image, meeting at the watersheds. More recent work suggests hybrid [123, 62] or fully parallel depth-first watershed algorithms [124, 125, 63, 97, 98]. For each pixel in the image, a path of steepest descent is constructed and can be followed down to the nearest minimum. Labels of minima are then propagated uphill, grouping pixels with the same label into a region around the minimum with that same label.

3.3 Waterfall transform

3.3.1 Description

The waterfall transform is a hierarchical segmentation approach that is based on the watershed transform, and can be classified as a merging technique to address the over-partitioning problem of the watershed. The main idea of waterfall is that catchment basins in the watershed result can be transformed into new plateaux and the watershed can be re-applied on the resulting new image.

The transformation finds the lowest pass point (called first overflow zone (FOZ)) on the watershed line around each catchment basin and elevates the parts of the catchment basin that are below that pass level (called *lower catchment basin*) up to that same pass level. The resulting image contains a reduced number of regional minima. Hence, watershed applied on this new image produces a lower number of catchment basins, compared to the original watershed application. The repetitive application of the image transformation and the watershed produces a series of partitionings of the image with a monotonically decreasing number of catchment basins, until a single region corresponds to the whole image.

3.3.2 Algorithms

The waterfall transform was first introduced by Beucher [18, 19]. In addition to the waterfall that can be directly applied on an image, a number of graph-based waterfall algorithms were proposed for fast sequential implementation [84, 48]. These algorithms rely on a weighted graph representation. Here catchment basins and/or watershed lines are represented as nodes. Edge or node weights are set to heights of lowest pass points along watershed lines separating adjacent catchment basins. After the construction of the graph, a watershed transform is applied on this graph, or its *minimum spanning tree*, rather than the original grid. Thus, the watershed result

groups nodes in this graph, i.e. regions in the original watershed image.

The main advantage of the waterfall transform is that it does not take any parameters. However, this approach also has some shortcomings which are discussed in detail in [21]. The first defect is the difficulty of finding a 'good' level of hierarchy that partitions the image well. The second major defect is called the "waterfall transformation short-sightedness" when important contours are removed early from the hierarchy, while other similar contours are preserved (see [21] for more details). In order to mitigate the problems of the classical waterfall, a few modifications are proposed to produce *enhanced waterfall* transformations. The enhanced waterfall algorithms, named *standard algorithm* and *P algorithm*, re-introduce at each level of the hierarchical segmentation selected contours into the watershed results based on specific rules. The main goal of the enhancement is to mitigate the effect of "waterfall transformation short-sightedness".

Later, in [20], the term 'enhanced waterfall' is reserved for a third modification (in addition to the standard and P algorithms) of the original classical waterfall procedure. Also, this paper shows how the standard and P algorithms, the two modifications of waterfall from [21], can be generalised by replacing the hidden parameter with an explicit one, called gain, and be extended. Note, that all three modifications and their extensions are applied directly on images, as opposed to the graph-based algorithms [84, 48] for classical waterfall.

One important application of the classical waterfall transform is the construction of *IPFs* (see [43, 47]).

3.4 Front propagation with eikonal equation

The problem of accurate approximation of a moving curve or surface on a plane, or generally any dimensionality space, arises in many applications. The *level set methods*,

introduced by Osher and Sethian [92, 108], represent the moving interface through a level set of a higher dimensionality function. These methods are used to track the development of interfaces that at each point change in their normal direction with speed F. When the speed has a constant sign everywhere, the front propagation can be approximated with a faster algorithm, called *fast marching method (FMM).*

3.4.1 Fast marching basics

The FMM is an efficient iterative algorithm, introduced by Sethian [104, 107], for numerical approximation of the development of fronts propagating through $\mathbb{R}^n$ space (see also [118]).

A propagating front is a closed hypersurface, each point of which moves in the direction of the surface normal with some speed F. Let $S(t) \subset \mathbb{R}^n, t \in [0, \infty)$ be the propagating interface in space $\mathbb{R}^n$. If the speed function $F = F(\vec{x})$ does not change sign, the equation of the evolution of the front can be represented in terms of the arrival time function $T : \mathbb{R}^n \to \mathbb{R}$, such that

$$T(\vec{x}) = t \iff \vec{x} \in S(t)$$

as the eikonal equation (see, for instance, [108]):

$$|\nabla T| = \frac{1}{F} \tag{3.1}$$

Fast marching discretises the eikonal equation (3.1) with upwind approximation schemes, in $\mathbb{R}^3$:

$$\left[\begin{array}{r} \max\left(\frac{T_{ijk}-T_{i-1jk}}{\Delta x}, -\frac{T_{i+1jk}-T_{ijk}}{\Delta x}, 0\right)^2 \\ +\max\left(\frac{T_{ijk}-T_{ij-1k}}{\Delta y}, -\frac{T_{ij+1k}-T_{ijk}}{\Delta y}, 0\right)^2 \\ +\max\left(\frac{T_{ijk}-T_{ijk-1}}{\Delta z}, -\frac{T_{ijk+1}-T_{ijk}}{\Delta z}, 0\right)^2 \end{array} \right]^{1/2} = \frac{1}{F_{ijk}} \tag{3.2}$$

where $\Delta x, \Delta y, \Delta z$ are the spacing of the discretisation lattice, T_{ijk} and F_{ijk} the values of these functions on the points of the lattice. When T_{ijk} is the unknown, equation (3.2) becomes a quadratic equation and is easily solved at any position. Using the fact that the front crosses each point only once, an efficient Dijkstra-like $O(N \log N)$ algorithm, where N is the total number of points in the lattice, is devised to determine the values T_{ijk} in the order the front propagates (details in [104, 105, 106, 107, 108, 109]). The FMM relies on a *priority queue* interface, typically implemented with a *binary heap* data structure, to sort the points in the increasing order of arrival times. An $O(N)$ complexity implementation of fast marching can be found in [132, 99], that uses an *untidy priority queue*, i.e. bucket sort, thus eliminating the $\log N$ factor from the implementation complexity.

Seeds

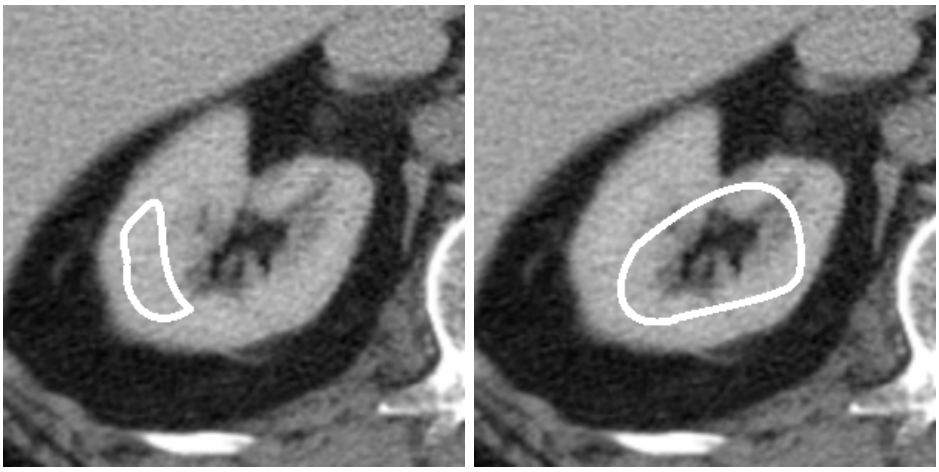
Seeds are points which initialise the front (we denote the set of these as $S(0) \in \mathbb{R}^n$). There can be more than one seed for a given front. In current practice for medical imaging, the choice of seeds is left to the user [79, 75, 74, 77, 108]. In other cases, heavily data specific approaches are used for automated seed extraction [26, 27] in multi-stage segmentation.

The set of seed points is the zero level set of function $S(\vec{x})$ before the front expansion begins. This means that the time function $T(\vec{x})$ has to be equal to 0 at, and only at, these points:

$$T(\vec{x}) = 0 \iff \vec{x} \in S(0)$$

The requirements [108] for the set of seed points in the fast marching method are

- all the seed points are within the target region to be identified, or
- the seed points comprise a closed front which encloses the target region (or target regions) to be identified.



(a) **(b)**

Figure 3.5: Two candidates for an initial front for fast marching within the anatomical structure. (a) A suitable initial front for a kidney. When developing outwards it approaches both the inner and outer borders of the kidney, thus shaping the organ. (b) An unsuitable candidate for a kidney. If expanding, it only approaches the outer border. If contracting, on the contrary, the inner boundary alone is shaped.

In the first case an expanding wave is generated beginning from the seed points to grow until the boundary of the target region. In the second case, the propagating front has a negative speed term and it develops inwards.

However, for the purpose of identifying anatomical structures in medical images it is more reasonable to define seed points within target regions and propagate outwards. This is because some physiological structures are not simply connected in the CT or MRI image space. This means that identifying such structures with the front propagating inwards in fast marching would require several runs of the method. Also, some elaborate stopping criteria would be needed to correctly locate the different borders of structures.

The procedure is much simpler in case of using a front propagating outwards, using an initial curve which encloses only points which belong to the structure. This approach is illustrated in Figure 3.5 on the example of a kidney. When such a front expands, it approaches both the inner and outer boundaries of the kidney, and it shapes only the area of the kidney (excluding the renal pelvis). In contrast, a curve,

which encloses a part of (or the whole) renal pelvis, like in Figure 3.5(b) approaches only one of the inner and outer borders. If the front has a negative speed term and develops inwards, then only the inner border of the kidney will be approached. If, on the other hand, the speed term is positive, the front will develop outwards and approach only the outer border of the kidney.

Even a single seed point can sometimes be sufficient for the method to identify the region successfully. This is just the degenerate case of an initial front like the one in Figure 3.5(a). Thus, an initial curve can be replaced with a set of points (or a single point) within the region of interest.

Speed functions

Fast marching for segmentation of anatomical structures places important requirements on its speed term, i.e. the rate of development of the front interface. The speed function has to converge to zero rapidly in the vicinity of the region boundary to be identified. This is devised to make sure that the expanding interface does not go beyond the border of the targeted object but rather shapes it precisely.

The regions' boundaries usually correspond to noticeable changes in the image and hence are characterized by high values of the gradient magnitude function. In contrast, the area within the organ is commonly less heterogeneous. This means that the same gradient magnitude function tends to have smaller values within the anatomical structure. As a consequence, the speed term has to be chosen according to these properties of the gradient image. The speed must have relatively high values at points with small gradient magnitude and vice versa.

In [74], Malladi and Sethian use the speed term

$$F(\vec{x}) = \frac{1}{1 + |\nabla G_\sigma * I(\vec{x})|}$$

for shape recovery on medical images, where $G_\sigma * I(\vec{x})$ is the original image smoothed with Gaussian filter and $|\nabla \cdot|$ is the gradient magnitude [108]. While in [75] the same authors suggest

$$F(\vec{x}) = e^{-\alpha|\nabla G_\sigma * I(\vec{x})|}, \quad \alpha > 0,$$

as a speed term for the task, without discussing the relative merits of these choices.

In [26] and [27] two other speed terms are suggested but again there is no comparative analysis.

We show in [135] a series of systematic experiments with the speed term expressions like in equations (3.3) and (3.4). These are carried out on raw images and on images pre-processed with windowing and/or smoothing.

$$F(\vec{x}) = \frac{1}{\left(1 + \left(\frac{|\nabla I(\vec{x})|}{C}\right)^n\right)^m}, \quad C > 0, \quad n, m \in \mathbb{N} \tag{3.3}$$

$$F(\vec{x}) = e^{-C|\nabla I(\vec{x})|}, \quad C > 0 \tag{3.4}$$

In this paper we illustrated the effects by running the fast marching algorithm on a single CT slice. We always started the method at the same point on the right kidney and compared the development of the interface visually. The more precisely the right kidney was shaped the more successful we considered the speed term expression (in CT axial slices, the right kidney is on the left of the image).

Figure 3.6 illustrates the propagation stages of a front initialised inside a liver in an abdominal CT slice. The comparison of the left and right images displays how the speed term affects the shape of the evolving surface and the identification of region boundaries.

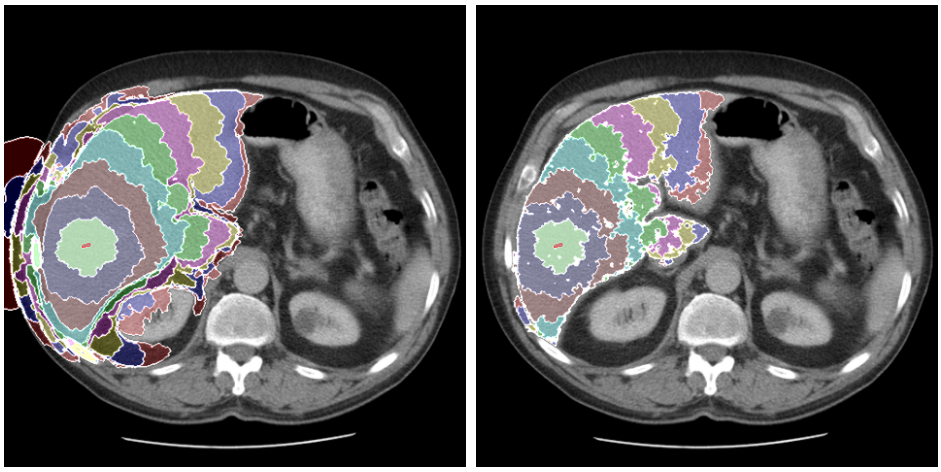


Figure 3.6: Fast marching with speed terms $F(\vec{x}) = \frac{1}{1+|\nabla I(\vec{x})|}$ (left) and $F(\vec{x}) = e^{-|\nabla I(\vec{x})|}$ (right). Closed white contours correspond to several level sets of $T(\vec{x})$ with a fixed level step Δt. Different coloured regions represent the positions for which $i \times \Delta t < T(\vec{x}) \leq (i+1) \times \Delta t, i \geq 0$.

Stopping criteria

Stopping criteria describe when and how the development of the propagating front is stopped. Existing approaches to stopping criteria include ceasing the development of the front at a fixed time [77] or histogram-based methods [26]. The development of the front in fast marching slows down at points with high gradient magnitude values (in particular, in the neighbourhood of organ boundaries), if a speed term similar to the ones described above is used. It is possible to devise procedures (see Section 4.3 for an example) to capture the moment of slow development of the front and to cease the expansion process there.

3.4.2 Other eikonal solvers

Following the introduction of the fast marching method, a number of other algorithms have been proposed for the solution of the eikonal equation on rectangular grids, triangulated surfaces, or other domains. Only the most well-known eikonal solvers are mentioned here.

The fast sweeping method was introduced by Zhao [143, 144] to solve the eikonal

equation by combining Gauss-Seidel iterations with alternating sweeping orderings. This algorithm offers an optimal complexity of $O(N)$ for N grid points.

The fast iterative method was introduced by Jeong and Whitaker [55, 56] to maintain an 'active list' of positions where the solution of the eikonal equation needs to be updated on the wavefront and to iteratively improve these approximations until convergence across the whole grid.

A number of efforts have been made in the existing literature to parallelise these eikonal solvers or to improve the efficiency of existing parallel solvers for CPUs or GPUs, see, for instance, [37, 53].

3.4.3 Use in medical image segmentation

Hypersurface front propagation methods, like level sets and FMM, have been previously used for medical image analysis. Malladi et al. [80, 78, 79, 81, 76] use level set methods to identify regions in 2D and 3D images. Fast marching for region labelling in medical images was investigated in a number of papers [75, 74, 77, 26, 27, 39].

3.5 Validation

An important aspect of the development of image segmentation algorithms is the availability of mechanisms to evaluate them. This is necessary in order to estimate how fit a segmentation approach is for the specific task, validate its performance on data and compare it against other approaches. In a clinical context, the accuracy and high quality of any segmentation results need to be validated before these can be used for implant preparation, operation planning, etc.

Medical image segmentation is very interesting in this context since there is no established evaluation framework. A commonly accepted approach is to compare the segmentation output to some reference results (usually produced with manual seg-

mentation) using similarity or difference metrics. While most authors rely on such methods there is still no agreement concerning the metrics used. This is partly because there are no metrics that reflect all the important features (accurate delineation of region boundaries, correct labelling of internal inhomogeneities and structures, domain-specific criteria, etc.) of a desirable segmentation and the existing metrics do not discriminate different segmentation results in an acceptable way.

3.5.1 Important factors

The evaluation of medical image segmentation algorithms is impacted by a number of factors and conventions specific to this task. The variability of medical image data quality is one of them and manifests itself in noise level, artefacts, partial volume effects, limited resolution. Poor quality images are not only difficult to segment with automated algorithms but also reference labelling images are not always possible to construct from such data in an unambiguous way.

Contrast-enhanced images

Often medical contrast substances are administered prior to image acquisition to enhance the image quality. Depending on the imaging modality and timing of the acquisition this may cause major organs to have a distinct appearance in the images and the segmentation algorithms to perform differently on contrast-enhanced images. For instance, minor blood vessels in a liver which form a part of the organ can be made to appear highlighted in contrast-enhanced images. Whether and how the inclusion or exclusion of these from the identified liver region should be penalised is an issue for the segmentation evaluation to address. However, current evaluation methods being used do not tackle it in any way. For instance, if the common evaluation approach of comparing the segmentation result to a reference region is exercised, then the choice of the reference region significantly affects the evaluation of the segmented region.

If the two regions agree on the exclusion of the blood vessels and the segmentation is scored as ideal, then the same segmentation may be largely penalised in case the reference region includes the blood vessels.

Pathological formations

Another issue for the tasks of medical image segmentation and its evaluation is the presence of pathological tissue in the image. These include cysts, tumours and other lesions and change the anatomical structures in the image by impacting the intensities, size, spatial extent, shape and other features of organs. They impose the greatest challenge on medical image segmentation algorithms with most approaches showing worst results on images with pathological organs (see Campadelli et al. [27], Section 8: proposed method obtains a lower score on image set with cancerous livers).

An important problem over which there still seems to be no consensus is whether, in which cases, and to what extent pathological formations should be labelled as part of the considered regions representing anatomical structures. For instance, the MICCAI 2007 liver segmentation grand challenge [52] (see Section 3.5.3 for more details) assumed tumours or cysts should be segmented as part of liver; as a result the segmentation approaches excluding the lesion region performed poorly and received lower scores. Other literature suggesting a similar approach includes [33, 36], while some papers take the opposite perspective [59].

Intra- and inter-observer variability

A detailed discussion of requirements and format of image segmentation evaluation frameworks can be found in the paper by Udupa et al. [119]. In practice, most image segmentation algorithms applied to medical image data are evaluated by computing region similarity or difference metrics for machine labelled regions and results of manual segmentation of the same images by trained operators. Since absolutely

accurate reference labellings are unavailable for this type of data, expert radiologists produce labellings of the data which are used as ground truth. Such data provides high level of reliability but is still affected by intra- and inter-observer variability (see [25] for reported results). Thus the number of available expert labellings crucially affects the evaluation procedures and results. For example, if labellings by a few experts are available, then an automatic segmentation method can be considered to perform as well as human experts as soon as the method's accuracy is within the range of inter-observer variability. A related topic of research is how to measure intra- and inter-observer variability and how many observers and observations are required to pin down good values for these variabilities.

3.5.2 Classification of evaluation method types

During the past couple of decades several authors conducted surveys on existing image segmentation evaluation methods and introduced classifications and comparisons of those. In a series of papers [140, 141, 142] Zhang discusses an extensive set of evaluation methods, categorises them based on the approaches used to assess image segmentation, compares these categories and reports an empirical comparison of specific evaluation methods. Evaluation methods are grouped into *analytical methods*, which assess segmentation algorithms by considering their principles, *empirical goodness methods* analysing properties of segmented images, and *empirical discrepancy methods*, which compare segmented images to *reference images* (gold standard, ground truth).

The empirical methods are further categorised by the quality measures they use to assess the segmentation algorithm performance for output images. The measure groups for empirical goodness methods include intra-region uniformity, inter-region contrast, region shape, moderate number of regions. The empirical discrepancy methods employ measures of number of mis-segmented voxels, position of mis-

segmented voxels, number of objects in the image, feature values of segmented objects, miscellaneous quantities (region consistency, grey level difference, symmetric divergence) [142]. Also, a group of evaluation methods with measures, like amount of editing operations, visual inspection or correlation between original image and bi-level image are considered special cases, not fitting in any of the groups.

The reviewed methods are compared using criteria of generality, subjective/objective, qualitative/quantitative, complexity, consideration of segmentation applications and requirement for reference images. Additionally, the performance of several empirical (goodness and discrepancy) methods is compared on a series of simple synthetic images segmented with thresholding [140] with discrepancy methods showing better sensitivity to threshold changes.

Zhang, Fritts and Goldman [139] group image segmentation evaluation methods into five categories: *subjective evaluation*, when output images are assessed by human domain experts, *system level evaluation*, which analyses the segmentation algorithm by considering results of systems using the segmentation, *analytical methods* (similar to [140]), *supervised methods* (corresponding to empirical discrepancy) and *unsupervised methods* (empirical goodness). They present a detailed discussion of a range of *unsupervised* evaluation methods, analysing the various metrics used and their combination methods. A number of experiments are performed on synthetic and real images for representative evaluation methods to compare two segmentation results. The accuracy of these methods is reported as the percentage of cases when the segmentation results preferred by the evaluation method and by human judges match. It is concluded from the findings that although the unsupervised evaluation methods perform well at comparing different parameterisations of a single segmentation algorithm, they are not successful in cases of machine segmentation versus different machine segmentation, or machine segmentation versus human segmentation comparisons.

The authors suggest using machine learning to combine simple unsupervised evaluation methods [138]. This approach outperforms all the previous methods on the suggested experiments showing promising results.

3.5.3 Evaluation frameworks

A special algorithm for validating image segmentation, *simultaneous truth and performance level estimation (STAPLE)*, is introduced by Warfield, Zou, and Wells [128]. This is an expectation-maximisation approach for comparison of several (human or machine) segmentations. It gives a probabilistic estimate of the true segmentation and performance scores for the considered segmentation approaches. However, this evaluation method is designed to analyse different segmentation algorithms simultaneously and is not readily applicable to evaluating a single segmentation approach (unless several reference segmentations are available).

Crum et al. [35] redefine established overlap metrics (JSC and DSC, see Section 3.6.1 for definitions) for fuzzy segmentation results and generalise these metrics to consider multiple segmentation labels and images. Thus, a single total fuzzy overlap result can be generated for a series of images fuzzily segmented using several labels with two different segmentation algorithms. A corresponding distance metric is also introduced. Such generalisations turn out to be useful when an overall performance score is sought across different labels and/or images. However, they still suffer from the shortcomings of the basic metrics being redefined.

As part of MICCAI 2007 conference a competition and workshop called "3D Segmentation in the Clinic: A Grand Challenge" was organised [52] which analysed and compared the performance of several state-of-the-art automatic and semi-automatic liver segmentation algorithms for CT images. For evaluation, a scoring system was used which first calculates 5 different segmentation error metrics, calibrates the results to bring the scores to a unique scale, and finally takes the average of the results

in this scale to get a single score for a segmentation approach (see [52] for details).

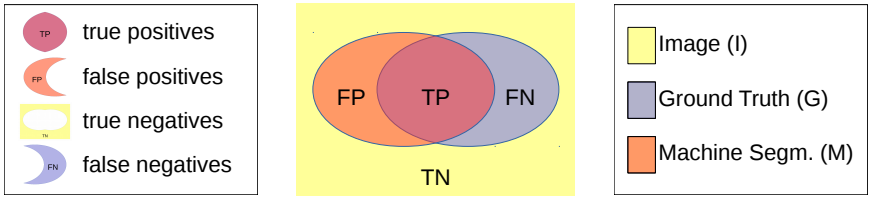
Cárdenes et al. [28] define new image segmentation evaluation metrics by introducing position and intensity values of misclassified voxels into the established overlap-based methods. Two other proposed metrics rely on the connectivity of segmented regions and limiting overlap-based methods (JSC, in this case) to only region boundaries respectively. Finally, the authors combine these metrics into an aggregated multimodal similarity metric. In a series of experiments on real and synthetic data they reveal that the new metrics show more variability and sensitivity when comparing segmentation algorithms than the classic metrics like JSC.

3.5.4 Evaluation in practice

One of the key aspects of a systematic classification of these metrics is experimental data and examples. Crucially, none of the aforementioned surveys illustrate their findings on concrete examples.

While most medical image segmentation papers present some quantitative evaluation for suggested algorithms, there are also works which rely solely on qualitative analysis of the results [42, 32, 121, 66]. This is mainly due to the difficulty of acquiring high quality ground truth segmentations for medical images (see Chapter 8 for further discussion). Manual segmentation in a slice-by-slice manner by an expert radiologist is considered the common source of gold standard segmentations. But this process is extremely time consuming: an operator may spend several hours segmenting a 3D image volume of a few dozen slices. In addition, manual segmentation is exposed to intra- and inter-observer variability.

Table 3.1: (1) Overlap-based: Dice Similarity Coefficient (DSC) and Symmetric Volume Difference (SVD); Jaccard Similarity Coefficient (JSC) and Volumetric Overlap Error (VOE); True Positive (TPVF), True Negative (TNVF), False Positive (FPVF), and False Negative (FNVF) Volume Fractions; Precision (Prec) and Recall (Rec). (2) Size-based: Relative Volume Difference (RVD). (3) Boundary-distance-based: Hausdorff Distance (HD) and Average Symmetric Surface Distance (ASSD).



Similarity metric	**Difference metric**
$\text{DSC} = \dfrac{2 \times \text{TP}}{\text{FP TP} + \text{TP FN}} = \dfrac{2\,\lvert M \cap G \rvert}{\lvert M \rvert + \lvert G \rvert}$	$\text{SVD} = 1 - \text{DSC}$
$\text{JSC} = \dfrac{\text{DSC}}{2 - \text{DSC}} = \dfrac{\text{TP}}{\text{FP TP FN}} = \dfrac{\lvert M \cap G \rvert}{\lvert M \cup G \rvert}$	$\text{VOE} = 1 - \text{JSC}$
$\text{TPVF (Rec)} = \dfrac{\text{TP}}{\text{TP FN}} = \dfrac{\lvert M \cap G \rvert}{\lvert G \rvert}$	$\text{FNVF} = 1 - \text{TPVF} = \dfrac{\text{FN}}{\text{TP FN}} = \dfrac{\lvert G \setminus M \rvert}{\lvert G \rvert}$
$\text{TNVF} = \dfrac{\text{TN}}{\text{TN}} = \dfrac{\lvert I \rvert - \lvert M \cup G \rvert}{\lvert I \rvert - \lvert G \rvert}$	$\text{FPVF} = 1 - \text{TNVF} = \dfrac{\text{FP}}{\text{TN}} = \dfrac{\lvert M \setminus G \rvert}{\lvert I \rvert - \lvert G \rvert}$
$\text{Prec} = \dfrac{\text{TP}}{\text{FP TP}} = \dfrac{\lvert M \cap G \rvert}{\lvert M \rvert}$	
	$\text{RVD} = \left\lvert \dfrac{\text{FP TP} - \text{TP FN}}{\text{TP FN}} \right\rvert = \left\lvert \dfrac{\lvert M \rvert - \lvert G \rvert}{\lvert G \rvert} \right\rvert$
	$\text{HD} = \max \left\{ \max_{x \in \partial G} \mathbf{d}(x, \partial M), \max_{y \in \partial M} \mathbf{d}(y, \partial G) \right\}$
	$\text{ASSD} = \dfrac{\sum_{x \in \partial G} \mathbf{d}(x, \partial M) + \sum_{y \in \partial M} \mathbf{d}(y, \partial G)}{\lvert \partial G \rvert + \lvert \partial M \rvert}$

3.6 Existing evaluation methods

Because there is no agreed framework for evaluating the results of segmentation, authors choose their own metrics. In many cases, a metric is chosen because it was used in a prior reference paper with which a newly designed method is being compared. Whilst this enables direct comparison, it also generates a vicious circle which perpetuates the use of less than ideal metrics. Based on our own literature survey [134], although new evaluation methods have been proposed [60], the most popular methods in use in the segmentation literature are still metrics with simple definitions and/or simple descriptions. Their definitions are presented in Table 3.1.

Throughout this thesis we use the term 'metric' to refer to an evaluation measure of a segmentation result. Strictly speaking, a metric should be symmetric with respect to its operands, whereas some of the discussed similarity or difference criteria listed below are asymmetric. Existing literature uses both 'measure' and 'metric' interchangeably [142, 119, 28], but 'metric' is intuitive to a wider audience.

The term 'ground truth' describes the reference regions against which segmentations are compared. These reference regions are either generated synthetically or are produced manually by trained human operators. The term 'machine segmentation' refers to results obtained automatically, semi-automatically, or even manually.

3.6.1 Overlap-based methods

Dice Similarity Coefficient (DSC) and Symmetric Volume Difference (SVD)

Originally introduced by Dice [38] for ecological studies, DSC (also known as F1 score) is one of the evaluation metrics most frequently used in medical image segmentation [69, 129, 70]. SVD is a Dice-based error metric which gives the symmetric difference of the segmentation result and the reference shape [103, 64]. The term Symmetric Volume Overlap (SVO) is introduced by Campadelli et al. [25] for

SVO $= 1 -$ SVD, and is essentially the same as Dice, and used as their main metric.

Jaccard Similarity Coefficient (JSC) and Volumetric Overlap Error (VOE)

JSC, used in Liu et al. [71] (presented as Similarity Ratio), is defined as the intersection between the machine segmentation and the ground truth regions over their union. It is related to DSC, as shown in Table 3.1. VOE is the corresponding error metric [102, 30].

Volume Fractions

Four metrics—True Positive Volume Fraction (TPVF), True Negative Volume Fraction (TNVF), False Positive Volume Fraction (FPVF), and False Negative Volume Fraction (FNVF)—are borrowed from statistical decision theory metrics (Sensitivity and Specificity) [119]. Only two of the proposed metrics should be used together (e.g. TPVF and FPVF, but not TPVF and FNVF) because of the dependence relationships. Some authors [102, 71] give an alternative definition for FPVF by normalizing the number of false positive voxels over the ground truth rather than the rest of the image. The TPVF–TNVF pair is used by some authors [50, 40, 17], while others [31, 29, 30] compute the TPVF–FPVF pair.

Precision (Prec) and Recall (Rec)

Precision normalizes the volume of the correctly segmented region over the volume of the result of the segmentation. Recall (same as TPVF) normalizes the size of the correctly segmented region over the ground truth. Precision does not account for undersegmentation errors, while oversegmentation is not reflected in Recall. This pair is used in [27, 129]. It is also used extensively in computer vision.

3.6.2 Size-based methods

Relative Volume Difference (RVD)

RVD measures the absolute size difference of the regions, as a fraction of the size of the reference. The metric is sometimes also called Relative Absolute Volume Difference (RAVD). It is used commonly alongside other metrics [68, 69, 59, 102, 30, 52].

3.6.3 Boundary-distance-based methods

A distance metric for a voxel x from a set of voxels A is defined as: $\mathbf{d}(x, A) = \min_{y \in A} d(x, y)$, where $d(x, y)$ is the Euclidean distance between individual voxels incorporating the real spatial resolution of the image. A number of metrics are based on this definition of distance and quantify the dissimilarity (measured in absolute length, such as mm) of the machine segmentation from the ground truth.

Hausdorff Distance (HD)

Let the directed distance between two sets of points be defined as the maximum distance from a point in the first set to a nearest point in the other one [41]. The symmetric Hausdorff metric for the boundaries of the segmented regions, also referred to as Maximum Symmetric Surface Distance [52], is the maximum of the two directed distances between the boundaries of the segmented regions. For each voxel on the boundary of machine segmentation there is guaranteed to be a voxel of the ground truth boundary in a distance of at most HD, and vice versa [102, 30, 33, 71].

Average Symmetric Surface Distance (ASSD)

The Average Symmetric Surface Distance is the average of all the distances from points on the boundary of machine segmented region to the boundary of the ground truth, and vice versa. See [52, 102, 30, 33, 137, 64] for numeric results with this

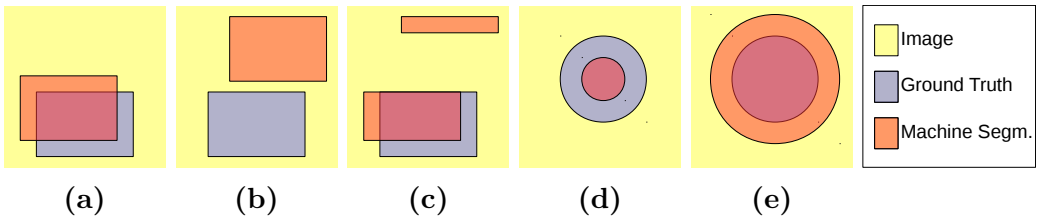


Figure 3.7: (a) and (b) get the same size-based score; (a) and (c) get the same overlap-based score; (d) and (e) get the same boundary-distance-based score.

metric.

3.6.4 Advantages and disadvantages of existing metrics

Individual similarity or difference metrics fail to capture all the aspects of the segmented regions. Size-based metrics rely only on the difference in size between the segmentation and the ground truth. The best attainable score can be achieved even when the segmentation and the ground truth are disjoint. Overlap-based methods account only for the number of correctly classified or misclassified voxels without taking into account their spatial distribution. A segmentation which has 'leaked' (i.e. it features false positives adjacent to the area of true positives, Figure 3.7(a)) is scored the same as a leak-free one with a separate disconnected region of false positives, the same size as the leakage area (Figure 3.7(c)). Similarly, the conventional distance-based methods take into account only the distance from each point on one boundary to the other boundary. These metrics are oblivious to the absolute size of the regions involved.

Figure 3.7 illustrates different types of segmentation errors which nevertheless receive equal similarity or difference scores from size-based methods, overlap-based methods, or boundary-distance-based methods. In Figures 3.7(a) and (b) the ground truth and machine segmentation are of equal size (in the number of pixels), hence their size-based scores will be the same (best attainable difference score of 0). In Figures 3.7(a) and (c) we can see that the two images have the same pixel counts

in the true positive, true negative, false positive, and false negative classes. Hence the overlap-based scores for these images are the same. This is despite the obvious differences in the spatial distributions of the regions.

Figures 3.7(d) and (e) show discs representing machine segmentations having radii of 0.5 and 1.5 times the radius of the ground truth. In each of (d) and (e) the distance between the two boundaries is the same and equal to a half of the radius of the ground truth disc. Therefore, boundary-distance-based metrics fail to differentiate between case (d) and case (e). These examples are revisited in Section 5.3.1.

Chapter 4

Automated Sequential 3D Segmentation Based on Hierarchical Partitioning

4.1 Introduction

In this chapter we refer to *segmentation* as the process of identifying and labelling a 3D region of semantic importance (such as a kidney). In doing so, we pre-process the data in order to group voxels with similar characteristics into regions—a process known in computer vision as (2D) *image segmentation*. In order to avoid ambiguity, we will refer to the vision segmentation technique as *partitioning* the image, and to the medical segmentation technique as *labelling* the features of interest.

Current practice for medical imaging [79, 75, 74, 77, 108] leaves the initialisation of level sets or FMM to the user. Even commercial software for processing scan data expects the user to hand-seed any similar methods, and this on a slice-by-slice basis. In other cases, heavily data-specific methods are used for automated seed (initial region) choice [26, 27]. We present a new approach for fully automatic labelling and

reconstruction of features, such as organs, which combines FMM with a hierarchical image volume partitioning method.

Following the previous work of our colleagues on decision-support systems for renal segmentation [44, 46], we illustrate novel unsupervised organ segmentation [136] through the use of an automated initial labelling approximation algorithm followed by a hypersurface front propagation method. The initial approximation stage relies on a pre-computed image partition forest obtained directly from CT scan data.

4.2 Image volume partitioning

We illustrate our algorithms with abdominal CT scans in DICOM format, each volume typically comprising up to 50 axial slices, 512×512 voxels each. Our labelling approach is easily generalised to any dimensionality. We choose always to run it on 3D data volumes, preserving the original resolution. We pre-process the raw data with windowing [117], which maps the original radiodensity-based range of HU to (256) greyscale values preserving granularity detail in a given HU subrange of interest. After this, edge-preserving smoothing with several iterations of anisotropic diffusion filtering [95] is applied. The pre-processing is followed by a partitioning step, where the 3D data volume is organised into a tree-based representation of adjacency graphs. The nodes of the trees are 3D regions of voxels, and the parental relationship is that of inclusion. Essentially, the data is partitioned into a hierarchy of inter-connected regions and subregions. This data structure is called an image partition forest (IPF). Our colleagues have described elsewhere [45, 48] how they use a watershed and waterfall transform–based approach [19] to construct the IPF in a bottom-up manner. In an IPF, layer 0 corresponds to the image, layer 1 is the watershed result, and layers 2–5 of decreasing granularity are constructed with the waterfall transform. The process stops at layer 5 since the number of regions decreases significantly from one

layer to the next and the production of further layers is unnecessary for the following steps.

The most crucial aspect of automating the segmentation is the way in which we use the IPF to find a rough approximation of a kidney, and seed from it a hypersurface front propagation method which helps localise the kidney boundaries more accurately.

Since the typical HU range (and hence windowed greyscale) for kidneys is known from radiology studies [117], we can choose as kidney candidates those regions in the IPF which are within the desired mean greyscale range. Furthermore, filtering regions, which correspond roughly to kidneys, can also be based on anatomical knowledge. The IPF candidate regions should be in a high layer of the IPF (layers 3–5); in an axial slice, the right kidney should lie on the left of the spine[1] and anatomically close to it; also, the right kidney should span a reasonable number of voxels (depending on the number of slices in the image). A similar filtering procedure, employing symmetry, has been devised for left kidneys. Other abdominal major organs can also be approximated in the IPF with analogous approaches relying only on established anatomical knowledge.

Algorithm 1 illustrates the exact filtering procedure of 3D regions in the IPF for right kidney approximation candidates. This procedure assumes windowing centred at 40 with width 400 and 512×512 in-slice resolution. The values are shown here as hard-wired, but are easily parameterisable to allow for different tissue type, different scan parameters, or even different scan type (e.g. MRI). Our colleagues illustrate in a separate paper [82] how one can arrive at specific windowing parameters for a given scan type.

Figure 4.1 illustrates a 2D projection through a 3D IPF, and its different levels of coarseness. The bottom-most level is made up of the raw image voxels (not shown); it is followed by a layer of over-segmented regions (not shown), followed by increasingly

[1] A simpler filtering procedure is devised and executed for the spine [45], so the filtering of the other labels can be based on the spine.

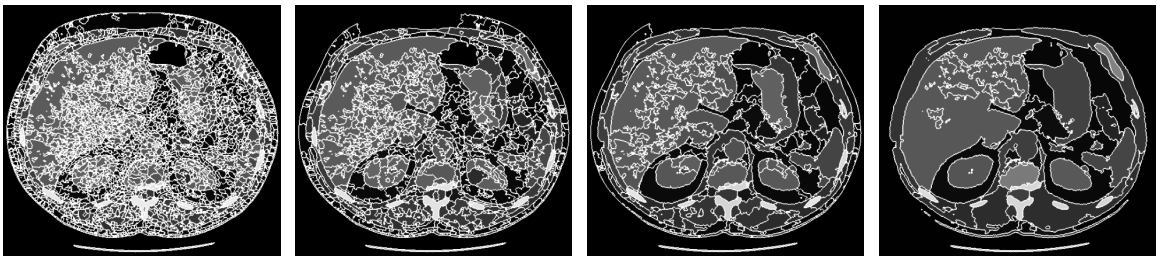
Algorithm 1 The localisation of 3D regions in the IPF for right kidney approximation candidates

$minlayer \leftarrow 3$
$mingreyvalue \leftarrow 150$
$maxgreyvalue \leftarrow 218$
$maxdistancex \leftarrow 180$
$mindistancey \leftarrow 20$
$minaspectratio \leftarrow 0.2$
$maxaspectratio \leftarrow 5$
$minvoxels \leftarrow 1600$
$maxvoxels \leftarrow 6000$
for all $region \leftarrow$ IPF **do**
  if $region.layer \geq minlayer$ **and**
   $mingreyvalue \leq region.meangreyvalue \leq maxgreyvalue$ **and**
   $region.xmax < spinecentroid.x$ **and**
   $region.xmin > spinecentroid.x - maxdistancex$ **and**
   $region.ymax \geq spine.ymin + mindistancey$ **and**
   $minaspectratio \leq region.aspectratioxy \leq maxaspectratio$ **and**
   $minvoxels \times slices \leq region.voxelcount \leq maxvoxels \times slices$ **then**
    $rightkidney \leftarrow rightkidney + region$
  end if
end for

coarse regions that belong together and gradually group themselves around organs and other features.

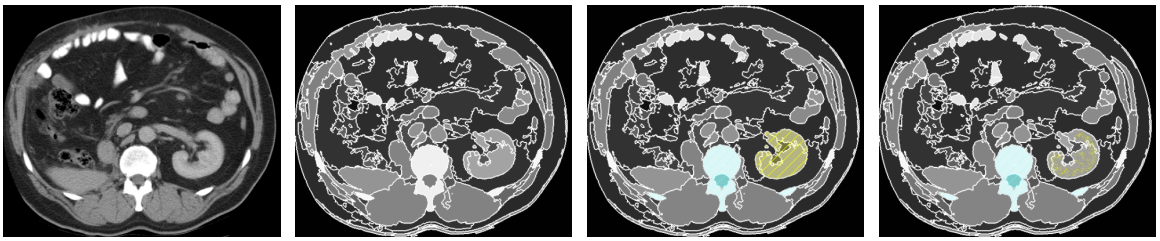
4.3 Feature labelling with fast marching and morphological closing

Our hypersurface front propagation method of choice is fast marching—an efficient iterative algorithm, introduced by Sethian [104, 107], for numerical approximation of the development of fronts propagating through $\mathbb{R}^n$ space ($\mathbb{R}^3$ in this case). An $O(N)$ complexity implementation of fast marching can be found in the literature [132, 99]. In the image segmentation literature [77] this is always seeded manually. Multi-stage approaches to fast marching seeded from previous results have been used [27] but these are very data specific.



(a) **(b)** **(c)** **(d)**

Figure 4.1: Four of the six layers in a 3D IPF, representing image partitions in decreasing detail level. Layers 2–5 in (a)–(d), respectively; finest layers 0–1 not shown.



(a) Single abdominal CT slice. Note that the right kidney has undergone nephrectomy.

(b) Layer of IPF resulting from the segmentation of (a). Projection on a single slice of the 3D structure IPF.

(c) Approximation of the left kidney in the IPF (in yellow). The algorithm did not find a right kidney candidate.

(d) Seed points extracted from regions approximated in (c); only points in healthy-tissue range $[170, 200]$ for kidney.

Figure 4.2: Image partitioning and seed extraction from the resulting IPF

The novelty of our approach is that we seed the fast marching automatically from a subset of the candidate regions output via the IPF. Our take to the seeding process relies on the automatic generation of the IPF structure. Whilst filtering the candidate regions (see Algorithm 1) relies on anatomical structure, this can be hard-coded into the automated segmentation. Only voxels which are in the expected healthy-tissue greyscale range $[170, 200]$ are considered and used as seeds, as shown in Figure 4.2(d).

The candidate regions chosen using the IPF do not approximate the kidneys sufficiently well. Local heterogeneities of high gradient magnitude value are usually missed and most regions have displaced edges. Because of this, these candidate regions may miss necessary pieces of tissue or contain chunks of external tissue. This is why the fast marching is employed. The quality of the outcome is highly dependent on the

speed term and the rate of development of the front. We conducted a range of systematic experiments [135] with several speed terms previously used in the literature for medical image analysis. Based on visual inspection, our best results were achieved with $F(\vec{x}) = e^{-|\nabla I(\vec{x})|}$. Here $|\nabla \cdot|$ is the magnitude of the gradient and $I(\vec{x})$ is the value of the image at location $\vec{x}$ after pre-processing (windowing and smoothing in this case).

Without going into detail, it is easy to see from this speed function that the development of the front slows down at points with high gradient magnitude values (in particular, in the neighbourhood of organ boundaries). We captured such slow development of the front by stopping the growth after the development is less than a threshold percentage of the region's volume in one time step.

The regions output by FMM are then post-processed with several iterations of morphological closing [49] to remove spurious holes and to smooth out the boundaries.

4.4 Automated segmentation

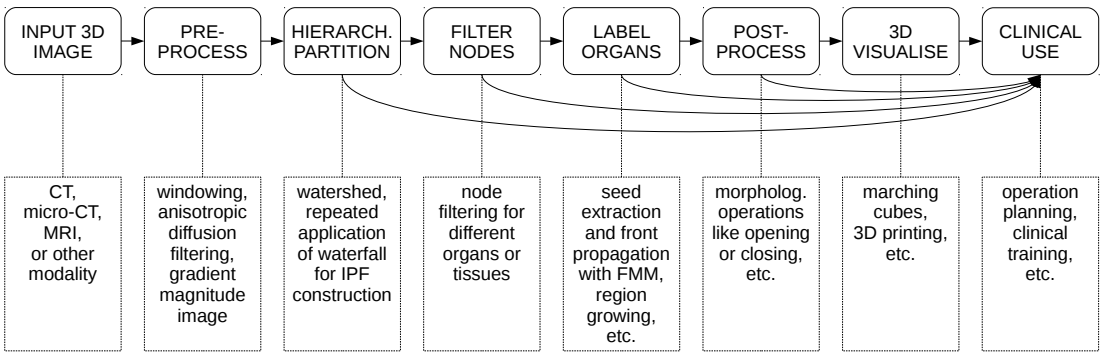


Figure 4.3: The generic structure of the segmentation procedure

Putting all the above elements together, we construct the following *novel automated segmentation procedure* (generic structure illustrated in Figure 4.3):

1. Pre-process the data with windowing and anisotropic diffusion filtering (as

per §4.2)

2. Partition the data and construct the IPF (§4.2)

3. Filter the resulting IPF for candidate kidney regions based on anatomical knowledge (§4.2)

4. Within candidate regions, choose seed points which are in an expected healthy-kidney greyscale range (§4.3)

5. Choose hypersurface front propagation and its parameters, and seed it from Step 4 (§4.3)

6. Advance the front until it makes little or no progress (§4.3)

7. Post-process the labelled regions with several iterations of morphological closing (§4.3).

Steps 2–4 of the automated method are illustrated on a slice in Figure 4.2, and some final results on another slice in Figure 4.7(a).

4.5 Results and validation

We have carried out extensive experiments on 3D volumes of up to 50 CT slices, allowing for both partitioning and fast marching to extend in all three dimensions. We varied the algorithm parameters systematically, and the overall results were consistent with what is summarised here.

4.5.1 Evaluation on main dataset

Our quantitative evaluation is based on 9 different kidneys (denoted A–I) from 7 patients' CT volumes of 5–38 slices each. Most of these volumes come from nephrectomised patients and contain only one kidney. A knowledgeable operator traced

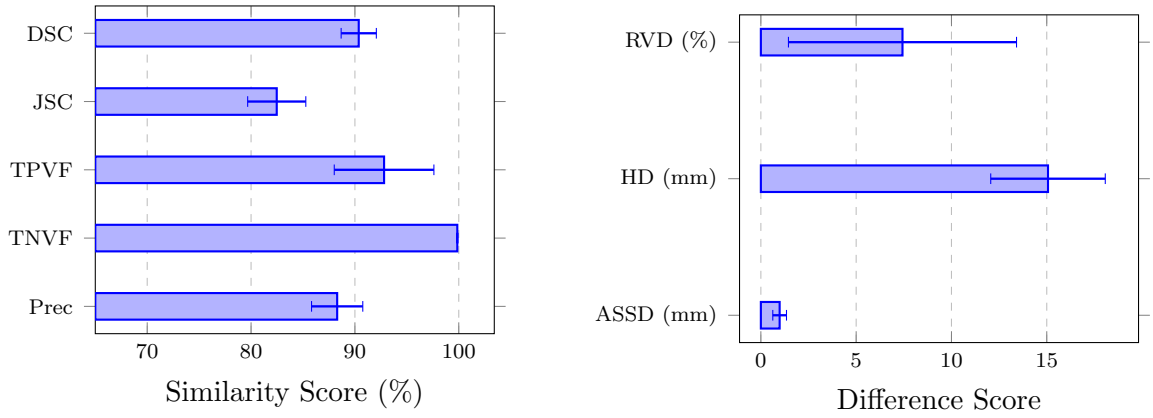


Figure 4.4: The average ± standard deviation values of the considered metrics for the main dataset of kidneys A–I. Similarity metrics are in the first graph; difference metrics are in the second.

contours manually for each organ. Inter-operator variability of 92–94% (Dice coefficient) was estimated by asking three different operators to label a kidney region in a single abdominal volume (details are presented in Section 4.5.3).

Table 4.1 shows the comparison between the manually traced gold standard and the output of our unsupervised segmentation algorithm per kidney. It is quantified using well-known overlap-based metrics—*DSC, JSC, TPVF, TNVF, Prec*; size-based metric—*RVD*; and boundary-distance-based metrics—*HD, ASSD* (for details see Section 3.6). Note, that these segmentation results are revisited later in Sections 5.3 and 5.4, where we analyse the behaviour of our novel segmentation evaluation metrics against conventional metrics.

The DSC evaluation, which ranges between 87.40% and 93.02% on kidneys A–I, is close to the inter-observer variability (92–94%). Average results and standard deviations for these metrics are reported in Figure 4.4. The low variation for most metrics of up to 5.98% and 3 mm indicates consistent performance. The evaluation outcome can only be compared superficially to kidney segmentation results previously reported in the literature [36, 69] due to differences in data quality.

Our dataset contains images of low resolution (5 mm slice thickness, 0.68–0.78 mm pixel resolution), acquired with or without contrast agent administration and covers

Table 4.1: Match of our segmentation against gold standards per kidney with several similarity or difference metrics on the main dataset (A–I) and the special cases (J–M). Metrics DSC–RVD are measured in %; HD and ASSD—in mm.

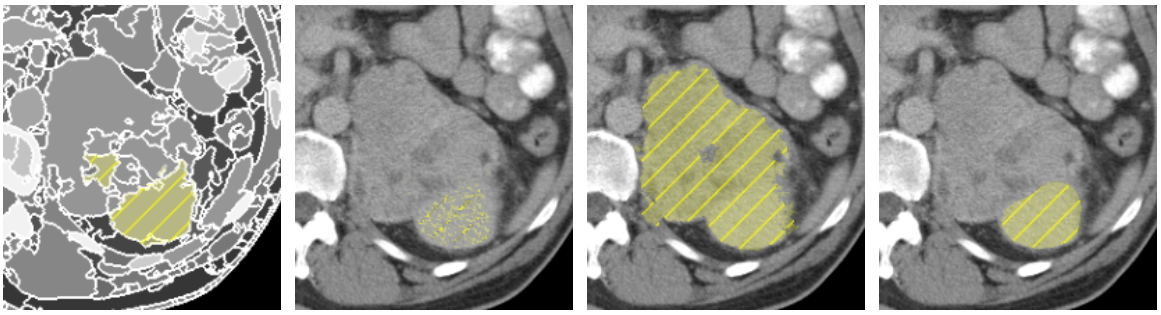
Kidney	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD
A	93.02	86.94	97.55	99.92	88.88	9.75	14.63	0.78
B	91.53	84.38	92.02	99.88	91.04	1.08	15.29	0.81
C	91.39	84.15	97.63	99.73	85.91	13.64	14.62	1.42
D	91.02	83.53	92.34	99.90	89.74	2.90	20.49	0.84
E	90.82	83.19	95.66	99.87	86.46	10.64	15.00	1.16
F	90.36	82.42	99.09	99.75	83.04	19.32	17.22	1.55
G	90.02	81.86	89.78	99.86	90.27	0.55	9.13	0.47
H	87.75	78.18	85.43	99.94	90.20	5.29	16.96	1.28
I	87.40	77.61	85.78	99.89	89.08	3.70	12.24	0.57
J	83.94	72.32	72.38	100.00	99.88	27.54	7.35	0.31
K	83.83	72.17	72.18	100.00	99.96	27.79	5.98	0.29
L	33.87	20.39	93.43	93.57	20.68	351.72	84.53	23.39
M	28.07	16.33	96.72	96.37	16.42	489.05	96.98	27.75

both healthy and diseased kidneys. The large slice thickness and the big number of slices in the volumes considered impact the value range for the boundary-distance-based metrics, causing the relatively higher values for HD. This naturally follows from the measure definition and should be taken into account when comparing different results.

4.5.2 Evaluation of special cases



Figure 4.5: An extreme case of a tumourous left kidney (not part of the validation set) on which our algorithm yields only 30% Dice similarity



(a) Labelling approximation in the IPF includes part of the tumour.

(b) Seeds for FMM have been extracted from the labelling approximation in (a).

(c) FMM growth extended to include the malignant tissue in the kidney labelling.

(d) Manually produced ground truth excludes voxels of the tumour.

Figure 4.6: Machine segmented kidney (kidney M) with big tumour vs. its manually labelled ground truth which excludes the tumorous tissue

We have excluded from the main dataset cases with kidneys severely distorted by large tumours, with necrosis, and shifted within the abdomen, as depicted in Figure 4.5. For such severely diseased organs with prominent malignant tissue, the DSC indicated a lower match of around 30%, but such cases represent only around 15% (two out of thirteen) of the set studied. There is no agreed convention on whether to include tumourous tissue into the segmentation, and the decision is mostly task-dependent. In our case, only healthy tissue had been labelled in the gold standard, yet the machine crossed the relatively weak boundaries between the tumour and the healthy tissue, and went on to label both, arguably mis-segmenting the organ. Figure 4.6 illustrates this.

Kidneys L and M in Table 4.1 belong to this class of heavily misshaped organs. The numeric results for these highlight the phenomenon of disagreement between gold standard and machine segmentation on tumorous tissue. The RVD results of around 352% and 489% emphasise the difference in size of the two regions.

Also not considered in the main dataset were single slice volumes. Results for single slice volumes are reported as kidneys J and K in Table 4.1. The pattern of most disagreements occurring around region boundaries and the larger portion of

Table 4.2: Comparison of automated segmentation validation results against inter-observer variability for a kidney volume independently segmented manually by three operators (Op1, Op2, Op3) and automatically with our algorithm (Alg). First three rows validate manual results against each other; last three rows validate the labelling by our algorithm against manual results. Metrics DSC–RVD are measured in %; HD and ASSD—in mm.

		DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD
Op1	Op2	92.61	86.24	95.27	99.91	90.09	5.75	10.67	0.70
Op1	Op3	93.86	88.43	93.10	99.95	94.63	1.62	15.00	0.69
Op2	Op3	92.53	86.10	89.31	99.97	96.00	6.97	10.50	0.62
Op1	Alg	90.82	83.19	95.66	99.87	86.46	10.64	15.00	1.16
Op2	Alg	90.01	81.84	92.10	99.89	88.02	4.63	12.65	1.12
Op3	Alg	89.79	81.47	95.38	99.86	84.81	12.46	12.99	1.13

boundary voxels in comparison to multi-slice volumes jointly lead to lower overlap similarity and higher distance difference results.

4.5.3 Inter- and intra-observer variability

In order to compare the segmentation algorithm performance to manual segmentation quality we consider its results alongside inter- and intra-observer variability. Table 4.2 summarises our findings on inter-observer variation and its impact on automated segmentation evaluation. We asked three operators to manually segment the right kidney (kidney E in Table 4.1) in a 21 slice CT volume, on which our algorithm showed average performance. There are consistent small score disagreements between the manual labellings with the several metrics, around 92–94% DSC and 10–15 mm HD. These are the result of the low image resolution and subjective image interpretation differences. The automatic segmentation evaluation scores against the three gold standards are only slightly inferior (lower similarity scores and higher difference scores) with approximately 89–91% DSC and 12–15 mm HD.

Our experimental results for intra-observer variation are presented in Table 4.3. We considered six consecutive slices in the same kidney volume (E in Table 4.1), and two operators repeated the segmentation of the kidney portion on those slices after

Table 4.3: Comparison of automated segmentation validation results to intra-observer variability for a kidney volume repeatedly (with a lapse of several months) segmented manually by two operators and automatically with our algorithm. First three rows validate intra-operator results and our algorithm for one operator (Op2 in the other experiment) against each other; last three rows present the results for the other operator (Op3 in the other experiment). Metrics DSC–RVD are measured in %; HD and ASSD—in mm.

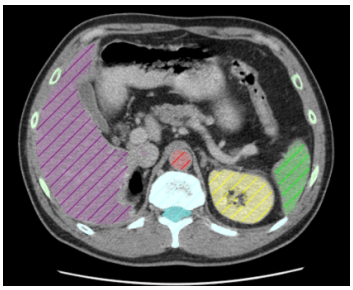
		DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD
Op2	Op2R	93.41	87.63	95.94	99.96	91.01	5.42	5.00	0.31
Op2	Alg	92.36	85.80	95.34	99.96	89.56	6.45	6.68	0.39
Op2R	Alg	92.95	86.83	93.41	99.97	92.50	0.98	6.74	0.32
Op3	Op3R	93.21	87.29	88.11	100.00	98.94	10.94	5.22	0.28
Op3	Alg	92.63	86.27	96.81	99.95	88.80	9.02	8.30	0.40
Op3R	Alg	89.24	80.57	99.24	99.92	81.07	22.41	7.89	0.57

a gap of a few months. Interestingly, the intra-observer scores are at a similar level as the scores for automated segmentation, with a very small advantage for the intra-observer results. Although this experimental comparison is based on limited data, it still showcases the good performance of our automated algorithm.

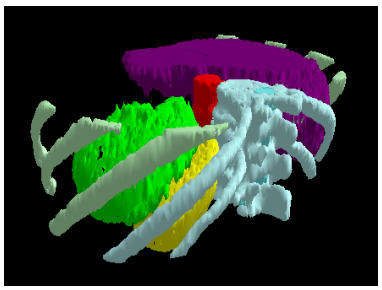
4.5.4 Remarks

The algorithm presented can easily be extrapolated to other body parts by simply altering the filtering procedure for initial organ approximation in Algorithm 1. The results of labelling the main abdominal organs in a 40 slice volume have been piped into multiple material Marching Cubes [131] to extract a triangular surface mesh, automatically yielding the 3D reconstruction shown in Figure 4.7. Once the 3D labelled regions are available, evaluation techniques (such as volume calculations, evaluation of thickness, or rate of growth) become straightforward to compute, as does preparation for 3D printing after mesh extraction(see Figure 4.7(c)). 3D prints of organs can be used by surgeons during operation planning, e.g. give inexperienced surgeons an idea of where to start the incision.

On a laptop with average configuration our automated segmentation algorithm



(a) Final results shown on a single axial CT slice



(b) 3D *marching cubes* reconstruction of labelled features



(c) 3D printed healthy kidney. *Reproduced from [111].*

Figure 4.7: 3D abdominal feature labelling by extrapolating the presented algorithm to other abdominal organs

runs for 10–20 minutes for an image volume of 20 slices and labelling several abdominal organs. We believe that running it on a parallel super-computer would shorten the overall time, but we are keen to maintain our image analysis software tool to lowest configuration parameters so as to make it more readily accessible to clinicians. In addition, the parallelisation of the different steps of this segmentation procedure is not straightforward. Chapters 6 and 7 present our work on the parallelisation of the partitioning step and a parallel alternative to the front propagation step.

4.6 Summary

The novelty of seeding a front propagation method automatically from a carefully crafted hierarchical structure makes our segmentation entirely unsupervised and automated. Our current experiments on kidneys show a match between automated segmentation results and manually labelled gold standard of 90%. This is similar to the state-of-the-art algorithms [36, 69], albeit on different data. We are not aware of open access benchmarking kidney data, nor for the abdomen.

Although our current quantitative results are reported for renal segmentation only, the method is easily extrapolated to other organs and tissue types, already shown in Figure 4.7. Since new medical scanners produce data with increasingly

higher resolution and accuracy, experiments on a wider class of volumes are needed, including on MRI data (which we have started tackling).

All our results have been validated qualitatively by a human domain expert. Whilst the procedure is fully automated, in practice users (particularly clinicians) sometimes prefer to override some of the automatic features of the algorithms. For that consideration, all our software tools provide both automatic and manual parameter setting, providing a light level of supervision if desired. The overall time saver for the user is, in any case, the automation of the overall segmentation, leaving them the much less onerous option of fine tuning the final labelled volume as a post-processing task.

Chapter 5

Boundary Overlap Metrics for Segmentation Evaluation

5.1 Introduction

It is often necessary to evaluate the results of image segmentation (and, indeed, compare contours in general) in an established systematic way. Despite attempts by several authors to classify and analyse methods for evaluating medical image segmentation, this field still lacks a widely accepted evaluation framework.

We propose a new family of boundary overlap metrics, which detect and measure a wider range of segmentation errors than are detected by most existing metrics. Our proposed metrics combine features of both overlap-based metrics and boundary-distance-based metrics in order to assess the similarity or difference of two regions: they do this by analysing the overlap in boundary neighbourhoods. The metrics, therefore, assess the percentage of boundary match between the expected correct shape and the segmentation result.

Empirical discrepancy methods (as per Zhang's classification [140]) compare segmented images to ground truth by exploring the similarity or difference of the labelled

regions. For our synthetic data, the ground truth consists of 2D black and white images of geometric shapes. For medical data, it is a mask which has been hand-drawn onto 2D or 3D medical scan images. Each metric outputs a score which measures the similarity or difference between the given ground truth and a segmented region.

We consider three common classes of such metrics: overlap-, size-, and boundary-distance-based, and examine the most frequently used members of these classes (see Section 3.6 for definitions and further details). These metrics are only sensitive to limited, mostly separate, ranges of segmentation errors; they produce contradictory scores and rankings. Heimann et al. [52] propose a framework for combining results from several metrics to produce a single score. Nevertheless, this remains an open question. More recent reports comparing segmentation algorithms avoid this by using one metric as the main indicator of segmentation quality [87] or by averaging rankings of algorithms obtained with different metrics [6].

An ideal metric should

(a) be able to flag even minute differences between any machine segmentation and its corresponding ground truth,

(b) be able to flag different types of segmentation errors (size, location, shape),

(c) spread its scores over much of its domain (say between 0% and 100%),

(d) be usable as a ranking tool for different segmentation algorithms.

We first construct a dataset of 52 synthetic images to simulate segmentation results with varying deviations from a chosen ground truth. We measure these images with a range of evaluation metrics (including our own) and reveal how well these correlate with specific segmentation errors. We subsequently compare the performance of the same metrics on real 3D scan images, using automatically segmented medical images [136] and hand-drawn ground truth masks. These experimental results show that our metrics fulfil the above goals.

Based on the results illustrated, we propose a new metric from the boundary overlap family which can be used systematically in assessing the results of segmentation, alongside existing metrics. In the future, our metric family can substitute effectively a number of existing metrics or can be adopted as a single general way of evaluation.

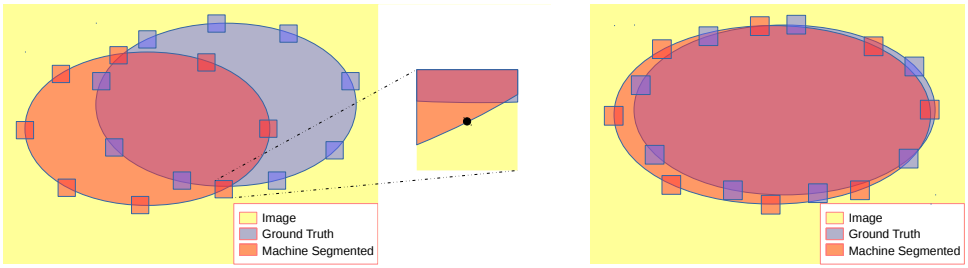
5.2 Novel evaluation methods based on boundary overlap

5.2.1 Definitions

We introduce a novel family of metrics with hybrid features, based on the concept of boundary overlap defined below. For this, we borrow features from both overlap-based metrics and boundary-distance-based metrics.

The boundary ∂A of region A is defined as the set of the pixels of A that have at least one neighbouring pixel outside that A region. A chosen fixed-size neighbourhood of a boundary pixel can be regarded as a smaller image. In such a neighbourhood, subsets S_A and S_B (potentially empty) of two separate regions A and B can be scored using any of the definitions in Section 3.6.1. In order to compare regions A and B overall, we consider each of the points on the boundaries ∂A and ∂B. We look at local neighbourhoods of those boundary pixels (see Figure 5.1(a)), scoring the overlap of S_A and S_B in each of those neighbourhoods, and taking an average of this overlap over all the boundary pixels. That overall average constitutes the boundary overlap metric. Since Section 3.6.1 offers a choice of several overlap metrics, we formally define a whole family of boundary overlap metrics in this way.

Figure 5.1(b) shows a more realistic situation, where the two boundaries are closer to each other. In some of the neighbourhoods the match is near perfect, in some others high overlap scores indicate large intersections of the sub-regions, i.e. a good match



(a) **(b)**

Figure 5.1: Some of the fixed-size neighbourhoods of the points on the boundaries of the segmented regions. Directed Boundary Overlap considers only red (or only blue) neighbourhoods. Symmetric Boundary Overlap considers both red and blue neighbourhoods.

between the boundaries being compared.

The boundary overlap can be either directed or symmetric. Directed Boundary Overlap considers the neighbourhoods of all the points on either the boundary of the machine segmentation or the boundary of the ground truth segmentation. Symmetric Boundary Overlap considers all the points on each boundary. A 100% Symmetric Boundary Overlap score can be interpreted as absolute match between ∂A and ∂B and, hence, between A and B. At the other end of the spectrum, a 0% score means no overlap of ∂A and ∂B and indicates a failed segmentation result.

It is important that any new evaluation metric allow for a succinct explanation in lay terms. Although not a precise description, it would be appropriate to say informally that boundary overlap metrics quantify the overlap percentage along the boundaries of the two regions.

Symmetric Boundary Dice (SBD)

Let N_x be the local neighbourhood of some radius r of a point x: $N_x = \{y, d(x,y) \leq r\}$, where $d(x,y)$ is the distance between points x and y. Let $A(N_x)$ and $B(N_x)$ be the portions of regions A and B in that neighbourhood N_x.

Let us denote the Dice Similarity Coefficient between region portions in neighbourhood N_x by $DSC(N_x) = \frac{2\,|A(N_x) \cap B(N_x)|}{|A(N_x)| + |B(N_x)|}$. Now we can define Directed Boundary Dice

(DBD) as the average of these overlap scores over all points on the first boundary:

$$\mathrm{DBD}(A,\, B) = \frac{\sum\limits_{x \in \partial A} \frac{2\,|A(N_x) \cap B(N_x)|}{|A(N_x)| + |B(N_x)|}}{|\partial A|}$$

The symmetric variant of this metric can be defined—Symmetric Boundary Dice (SBD):

$$\mathrm{SBD}(A,\, B) = \frac{\sum\limits_{x \in \partial A} DSC(N_x) + \sum\limits_{y \in \partial B} DSC(N_y)}{|\partial A| + |\partial B|}$$

According to these definitions, both our metrics have a fixed range of values between 0 and 1, where a higher value indicates a better match between the regions. We can change the separate components in the above definitions, thus getting other metrics in the same family. New metrics can be defined, for instance, by replacing the boundary averaging by another boundary distance feature, such as taking the maximum. Similarly, the overlap metric in the neighbourhood can be replaced by any other metric. Corresponding difference metrics can also be introduced, such as $1 - \mathrm{SBD}$.

Other Boundary Overlap Metrics

We introduce a number of novel boundary overlap metrics based on the conventional overlap metrics discussed in Section 3.6.1.

Directed Boundary Jaccard (DBJ) is defined as the average of Jaccard Similarity Coefficient results in the neighbourhoods of all points on the first boundary:

$$\mathrm{DBJ}(A,\, B) = \frac{\sum\limits_{x \in \partial A} JSC(N_x)}{|\partial A|} = \frac{\sum\limits_{x \in \partial A} \frac{|A(N_x) \cap B(N_x)|}{|A(N_x) \cup B(N_x)|}}{|\partial A|}$$

And the Symmetric Boundary Jaccard (SBJ) metric is defined as

$$\text{SBJ}(A,\, B) = \frac{\sum\limits_{x \in \partial A} JSC(N_x) + \sum\limits_{y \in \partial B} JSC(N_y)}{|\partial A| + |\partial B|}$$

Unlike DSC and JSC, the other overlap-based metrics considered are not inherently symmetric: they differentiate between the ground truth and machine segmented regions (see TPVF, TNVF, Prec definitions in Table 3.1). Thus, we define directed boundary metrics for TPVF, TNVF, and Prec separately on the ground truth boundary (subscript G) and the machine segmentation boundary (subscript M). Technically, these are evaluation measures rather than metrics.

Considering the ground truth boundary, we get Directed Boundary True Positive Fraction (DBTP_G), Directed Boundary True Negative Fraction (DBTN_G), and Directed Boundary Precision (DBP_G) defined respectively as

$$\text{DBTP}_G = \frac{\sum\limits_{x \in \partial G} TPVF(N_x)}{|\partial G|} = \frac{\sum\limits_{x \in \partial G} \frac{|M(N_x) \cap G(N_x)|}{|G(N_x)|}}{|\partial G|}$$

$$\text{DBTN}_G = \frac{\sum\limits_{x \in \partial G} TNVF(N_x)}{|\partial G|} = \frac{\sum\limits_{x \in \partial G} \frac{|N_x| - |M(N_x) \cup G(N_x)|}{|N_x| - |G(N_x)|}}{|\partial G|}$$

$$\text{DBP}_G = \frac{\sum\limits_{x \in \partial G} Prec(N_x)}{|\partial G|} = \frac{\sum\limits_{x \in \partial G} \frac{|M(N_x) \cap G(N_x)|}{|M(N_x)|}}{|\partial G|}$$

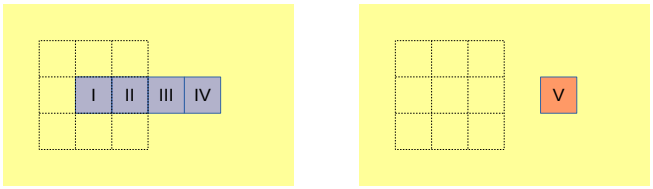
The corresponding directed boundary metrics on the boundary of the region obtained by machine segmentation are

$$\mathrm{DBTP}_M = \frac{\sum_{y\in\partial M} TPVF(N_y)}{|\partial M|} = \frac{\sum_{y\in\partial M} \frac{|M(N_y)\cap G(N_y)|}{|G(N_y)|}}{|\partial M|}$$
$$\mathrm{DBTN}_M = \frac{\sum_{y\in\partial M} TNVF(N_y)}{|\partial M|} = \frac{\sum_{y\in\partial M} \frac{|N_y|-|M(N_y)\cup G(N_y)|}{|N_y|-|G(N_y)|}}{|\partial M|}$$
$$\mathrm{DBP}_M = \frac{\sum_{y\in\partial M} Prec(N_y)}{|\partial M|} = \frac{\sum_{y\in\partial M} \frac{|M(N_y)\cap G(N_y)|}{|M(N_y)|}}{|\partial M|}$$

Finally, the corresponding symmetric boundary metrics—Symmetric Boundary True Positive Fraction (SBTP), Symmetric Boundary True Negative Fraction (SBTN), and Symmetric Boundary Precision (SBP)—relying both on ground truth and machine segmentation boundaries are defined:

$$\mathrm{SBTP} = \frac{\sum_{x\in\partial G} TPVF(N_x) + \sum_{y\in\partial M} TPVF(N_y)}{|\partial G| + |\partial M|}$$
$$\mathrm{SBTN} = \frac{\sum_{x\in\partial G} TNVF(N_x) + \sum_{y\in\partial M} TNVF(N_y)}{|\partial G| + |\partial M|}$$
$$\mathrm{SBP} = \frac{\sum_{x\in\partial G} Prec(N_x) + \sum_{y\in\partial M} Prec(N_y)}{|\partial G| + |\partial M|}$$

Note that the local overlap-based metrics may not be computable in some of the boundary neighbourhoods. For instance, if we consider x on ∂G and $|M(N_x)| = 0$, then $Prec(N_x)$ is unknown (zero divided by zero). Because such cases indicate a mismatch of the boundaries, we want to decrease their score and, hence, we replace such unknowns with zeroes in the above definitions. A simplified example is presented in Figure 5.2. In this example, the ground truth consists of four pixels (I–IV) and the machine segmentation consists of single pixel V, which overlaps with pixel IV of the ground truth. The Precision metric is undefined in the local Moore neighbourhoods of pixels I and II. In the neighbourhoods of pixels III and IV, the Precision score is 1



(a) **(b)**

Figure 5.2: A simplified segmentation case with undefined local Precision in some of the boundary neighbourhoods. (a) shows ground truth pixels I–IV; (b) shows machine segmentation pixel V. The highlighted Moore neighbourhood of pixel I contains no machine segmentation pixels, hence the Precision metric is undefined there.

(both the local intersection $G(N_{IV}) \cap M(N_{IV})$ and the local machine segmentation $M(N_{IV})$ count exactly one pixel, IV). The neighbourhood of pixel V is the same as the neighbourhood of pixel IV, hence that neighbourhood has a Precision score of 1. If we were to exclude the neighbourhoods with undefined Precision from the SBP definition, this machine segmentation would get a best attainable SBP score of $\frac{3\times1}{3} = 1$. When we replace the unknowns with zeroes, the SBP score of this example becomes $\frac{2\times0+3\times1}{5} = 0.6$, a more realistic figure.

Table 5.1 lists the newly defined metrics in the family; an experimental comparison of these new metrics is given in Section 5.4.

5.2.2 Parameterisation

Each boundary overlap metric can be parameterised by the neighbourhood radius; and thereby, increase or decrease the visibility of local overlap. In our current experiments, we consider cube-shaped Moore neighbourhoods of radius 1 (i.e. 9- and 27-neighbourhoods in 2D and 3D respectively), all the way up to radius 5, and we compare experimentally the performance of SBD for each radius. This systematic study follows in Section 5.3.3.

Table 5.1: Proposed boundary-overlap-based similarity metrics: directional and symmetric variants.

Boundary Dice metrics	**Boundary Jaccard metrics**
$\mathrm{DBD}_G = \mathrm{DBD}(G,\, M) = \dfrac{\sum\limits_{x \in \partial G} DSC(N_x)}{\lvert \partial G \rvert}$	$\mathrm{DBJ}_G = \mathrm{DBJ}(G,\, M) = \dfrac{\sum\limits_{x \in \partial G} JSC(N_x)}{\lvert \partial G \rvert}$
$\mathrm{DBD}_M = \mathrm{DBD}(M,\, G) = \dfrac{\sum\limits_{y \in \partial M} DSC(N_y)}{\lvert \partial M \rvert}$	$\mathrm{DBJ}_M = \mathrm{DBJ}(M,\, G) = \dfrac{\sum\limits_{y \in \partial M} JSC(N_y)}{\lvert \partial M \rvert}$
$\mathrm{SBD} = \dfrac{\sum\limits_{x \in \partial G} DSC(N_x) + \sum\limits_{y \in \partial M} DSC(N_y)}{\lvert \partial G \rvert + \lvert \partial M \rvert}$	$\mathrm{SBJ} = \dfrac{\sum\limits_{x \in \partial G} JSC(N_x) + \sum\limits_{y \in \partial M} JSC(N_y)}{\lvert \partial G \rvert + \lvert \partial M \rvert}$
Boundary TPVF metrics	**Boundary TNVF metrics**
$\mathrm{DBTP}_G = \dfrac{\sum\limits_{x \in \partial G} TPVF(N_x)}{\lvert \partial G \rvert}$	$\mathrm{DBTN}_G = \dfrac{\sum\limits_{x \in \partial G} TNVF(N_x)}{\lvert \partial G \rvert}$
$\mathrm{DBTP}_M = \dfrac{\sum\limits_{y \in \partial M} TPVF(N_y)}{\lvert \partial M \rvert}$	$\mathrm{DBTN}_M = \dfrac{\sum\limits_{y \in \partial M} TNVF(N_y)}{\lvert \partial M \rvert}$
$\mathrm{SBTP} = \dfrac{\sum\limits_{x \in \partial G} TPVF(N_x) + \sum\limits_{y \in \partial M} TPVF(N_y)}{\lvert \partial G \rvert + \lvert \partial M \rvert}$	$\mathrm{SBTN} = \dfrac{\sum\limits_{x \in \partial G} TNVF(N_x) + \sum\limits_{y \in \partial M} TNVF(N_y)}{\lvert \partial G \rvert + \lvert \partial M \rvert}$
Boundary Prec metrics	
$\mathrm{DBP}_G = \dfrac{\sum\limits_{x \in \partial G} Prec(N_x)}{\lvert \partial G \rvert}$	
$\mathrm{DBP}_M = \dfrac{\sum\limits_{y \in \partial M} Prec(N_y)}{\lvert \partial M \rvert}$	
$\mathrm{SBP} = \dfrac{\sum\limits_{x \in \partial G} Prec(N_x) + \sum\limits_{y \in \partial M} Prec(N_y)}{\lvert \partial G \rvert + \lvert \partial M \rvert}$	

5.2.3 Implementation and complexity

Algorithm 2 The calculation of SBD in 2D

```
1: procedure SBD(gt, ms, radius)        ▷ The SBD of the gt and ms binary masks
2:     sum ← 0
3:     number ← 0
4:     for all pos ∈ BOUNDARY(gt, radius) + BOUNDARY(ms, radius) do        ▷
   position may be considered twice
5:         gtsub ←SUBARRAY(gt, pos, radius)
6:         mssub ←SUBARRAY(ms, pos, radius)
7:         sum ← sum + DSC(gtsub, mssub)
8:         number ← number + 1
9:     end for
10:    return sum ÷ number
11: end procedure
12: procedure BOUNDARY(mask, radius)   ▷ The boundary of the mask with given
   radius
13:    for all p ∈ positions do          ▷ positions is the set of positions in mask
14:        if mask[p] and ∃q ∈NEIGHB(p, radius), ¬mask[q] then
15:            yield p                                  ▷ include p in the boundary set
16:        end if
17:    end for
18: end procedure
```

Algorithm 2 presents how SBD can be implemented for 2D images. The NEIGHB procedure returns the set of positions in the Moore neighbourhood of point p with *radius*. The BOUNDARY procedure constructs and returns the set of points on the boundary of *mask*, a binary representation of a region. The SUBARRAY procedure constructs a sub-mask with *radius* of the given mask at position *pos*. DSC and SBD are the procedures that calculate the corresponding metrics.

In terms of complexity, boundary overlap metrics allow for straightforward implementation which is linear in region size. From this perspective, they are more similar to overlap metrics and outperform boundary distance metrics, which have to consider all the pairs of points from the two boundaries (although nearly linear implementations exist for these [116]).

5.3 Experimental results

The existing metrics disagree greatly: some metrics are sensitive to specific types of errors while other metrics are not, and vice versa for other images. Depending on the clinical application, different aspects of the evaluation can take priority: sometimes errors in the shape are more important (e.g. cancerous tissue), sometimes the precise location is crucial (e.g. in radiotherapy), other times it is the overall volume that matters more (e.g. in volume evaluation of lung cancer). In an ideal world, the evaluation of any segmentation procedures would be carried out with feature-specific metrics.

5.3.1 Synthetic data

Before we look at medical scan data (in Section 5.3.2), we first investigate SBD using a set of 2D synthetic images carefully constructed so as to demonstrate instances where the existing metrics fail. The collections in Tables 5.2 and 5.3 contain simple geometric shapes (rectangle, disc, ellipse, star) and fractals, in order to enable us to reason about and visualise the behaviour of both old and new comparison metrics. For each shape there are images (left column) to serve as ground truth and a sequence of synthetic images (right columns) to simulate segmentation results. These shapes are constructed to simulate departure from the ground truth in size, location, orientation, or combinations thereof. Another potential departure from the ground truth can be in voxel or group-of-voxels errors; these are simulated by 'salt-and-pepper' pixels in the synthetic segmentations. The salt-and-pepper pixels are obtained by flipping pixels in the synthetic ground truth image with some probability. Disc, ellipse, and star images 9 and 10 are constructed by flipping pixels inside the white region with 20% and 5% probabilities, respectively. Corresponding images 11 and 12 are achieved by flipping all image pixels with 5% and 0.1% probabilities.

Table 5.2: Synthetic dataset: ground truth images (left column) along with images for segmented regions incorporating size, location, shape, simulated salt-and-pepper errors in the segmentation, or combinations. The text refers to these images with a combination of one of the letters 'r', 'd', 'e', or 's', and a number: r0 corresponds to the first—and ground truth—rectangle image; d12 is the last disc image, etc.

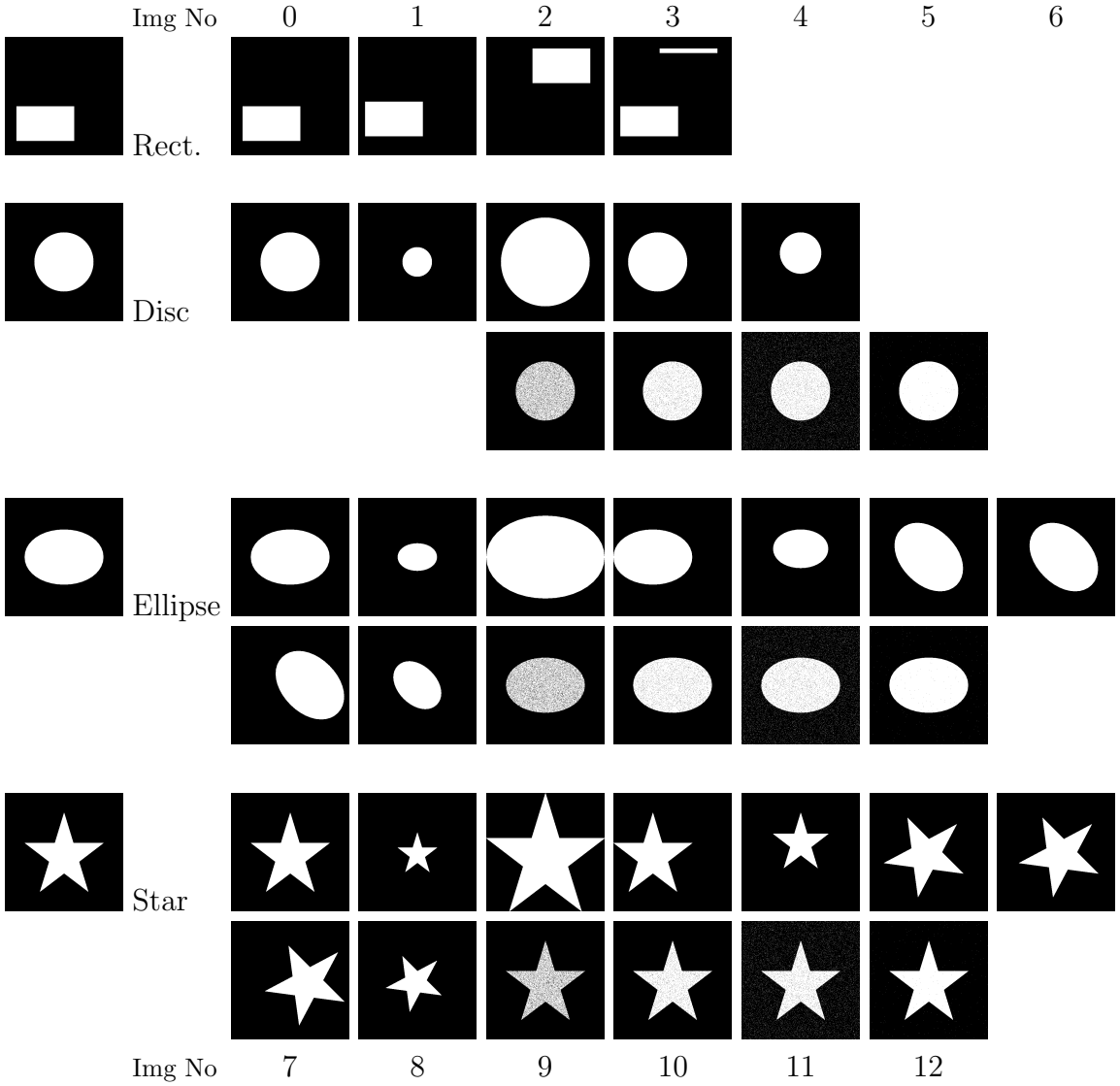
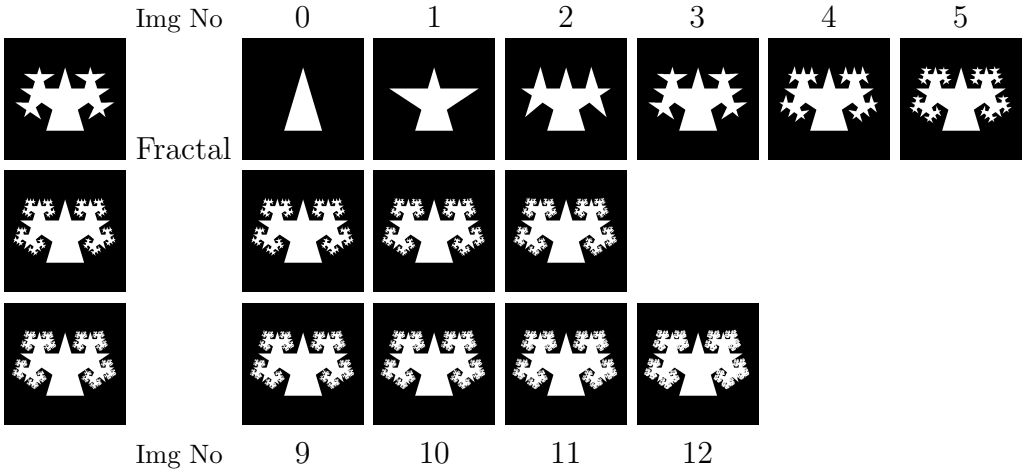


Table 5.3: Sequence of synthetic fractal images with new isosceles triangles added at each recursive step (0–12). The three images at steps 3, 6, and 9—used as ground truth—are also presented in the left column. The text refers to these images with a combination of the letter 'f' and a number: f6 is the image at recursive step 6; f12 is the image at recursive step 12, i.e. the last image.



We proceed to compare the white or near white regions in the right hand columns (representing potential segmentation results) against the white region in the left hand column (representing ground truth).

Simple geometric shapes

Our experiments in Figures 5.3 and 5.4 compare the segmented images against the ground truth using the existing metrics and some of our boundary overlap metrics. Whilst most conventional metrics are unable to differentiate clearly between images in a given sequence, SBD assigns them each a different score. Note that some of the difference metrics have been scaled in the graphs to be brought into similar value ranges. Such differences are the result of varying metric units.

Our results show that SBD generally penalises segmentation errors more than the other metrics, resulting in a wider (and hence easier to use) range of scores. The score assigned to the synthetic images designed with segmentation errors by most existing metrics is frequently the best attainable one. A few examples of such unfair scoring in Figure 5.4 include Prec and TNVF for images e8–e10, RVD for images s5–s7, TPVF

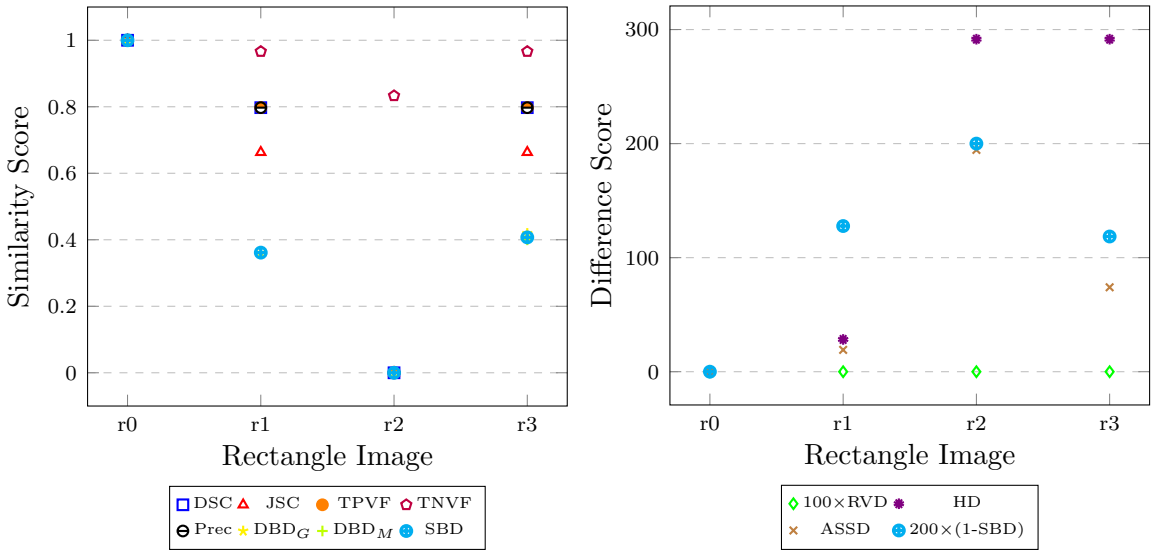


Figure 5.3: Results for synthetic rectangle images, comparing the segmented images against the ground truth, with all the considered metrics. Please note that DSC, TPVF, Prec scores are identical for all four images and their plots are not easily traced (similarity graph on the left). Likewise, DBD_G and DBD_M plots are almost identical with SBD.

for d2 and e2, etc. By contrast, SBD never assigns the best attainable score to an image with an obvious segmentation fault. For images with simulated salt-and-pepper errors in the segmentation (disc, ellipse, and star images 9–12), where other metrics mostly disagree, SBD averages the overoptimistic scores of overlap-based metrics and the overpessimistic scores of boundary-distance-based metrics. In other words, SBD penalises the random false negatives inside and false positives outside of the expected shapes more than overlap metrics and less than boundary distance metrics do.

SBD also assigns a low score to all images featuring some mismatch against the ground truth, thus showing better sensitivity to a larger span of error types. The directional variants of Boundary Dice (DBD_G and DBD_M) show interestingly contradictory results for images where other metrics evidently disagree; these contradictions are neutralised in the symmetric variant (SBD).

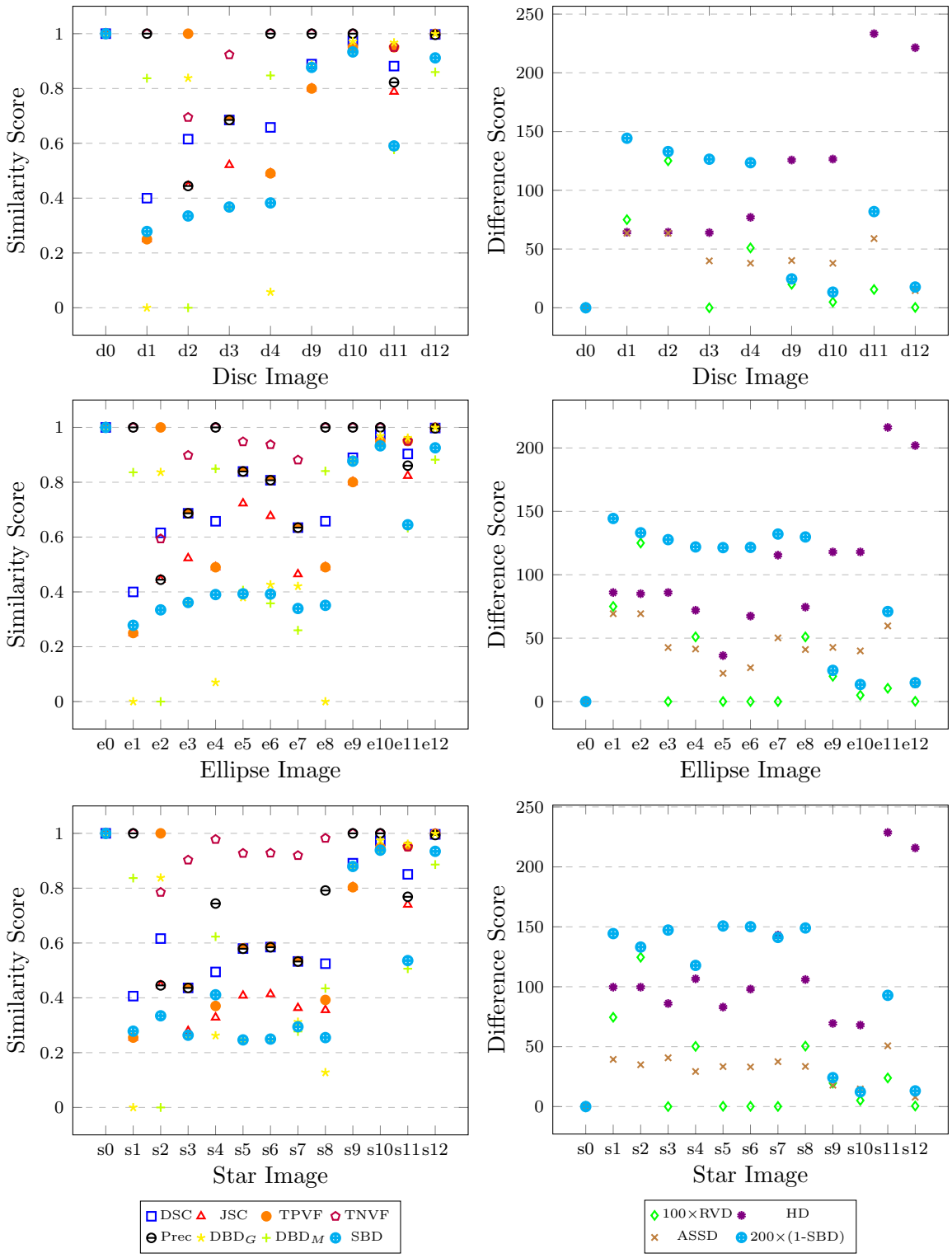


Figure 5.4: Results for synthetic disc, ellipse, and star images, comparing the segmented images against the ground truths, with all the considered metrics. Please note that some overlap-based (boundary-distance-based) scores are equal in the left graphs (right graphs) for a few images and their plots are not easily traced.

Examples for discussion

We revisit the examples from Section 3.6.4 in order to analyse the performance of existing metrics and SBD. Note that the synthetic images for rectangle shape (r0 as ground truth, r1–r3 as segmented, see Table 5.2) correspond to the discussed cases in Figures 3.7(a)–(c); and Figures 3.7(d), (e) match disc images d1 and d2 (d0 is ground truth). Evidently, such extreme segmentation faults are rare in real world situations. We chose to use these so as to simplify the demonstration: we care less about the quality of the segmentation and are only interested in how the metrics react to such segmentation faults.

As expected, the RVD score is zero throughout the rectangle examples (see Figure 5.3) since the sizes of the ground truth and the segmented regions are the same. The overlap-based metrics—DSC, JSC, TPVF, TNVF, Prec—give equal scores to the r1 and r3 images, ignoring the different location errors in these images. HD fails to differentiate between images r2 and r3. The only two metrics that assign different scores to all four images are ASSD and SBD, although they disagree on the ranking of the r1 and r3 images: while ASSD claims r1 to be better, SBD prefers r3 since there is an absolute match of a segment of the boundary in this image with the ground truth.

Let us imagine that we want to rank the three rectangle segmentations (r1–r3) with respect to the ground truth r0. DSC ranks r1 and r3 as a shared first with a score of 80%; r2 is third with 0% (see Figure 5.3 left). HD ranks r1 first with a distance score of 28 units; r2 and r3 are second with an extremely high distance score of 292 units (see Figure 5.3 right) and can be considered as failed segmentation results. Should r3 be considered a relatively good segmentation or should it be considered as a total failure? Evidently, r2, also Figure 3.7(b), is a worse result than r3, also Figure 3.7(c), and this should be reflected in the evaluation scores and the segmentation ranking. It can be argued that for specific tasks, where correct location of boundaries is a

priority, r3 can be considered a better segmentation than r1. This is because most false positives in r3 can be removed automatically by an algorithm which deletes the disconnected rectangle at the top of Figure 3.7(c). However, we acknowledge that the expected ranking of the images can be arguable and task-dependent; hence, the choice of evaluation metric should also be task-dependent.

As discussed in Section 3.6.4 and illustrated in Figures 3.7(d) and (e), the boundary-distance-based metrics (HD, ASSD) score images d1 and d2 identically. Some of the overlap-based metrics assign the best attainable scores to these images: TNVF, Prec to image d1; TPVF to d2. Image d1 is preferred to image d2 by TNVF, Prec, and RVD; while d2 is considered better by DSC, JSC, TPVF, and SBD. It is worth noting that the directional variants of Boundary Dice (DBD_G and DBD_M) successfully highlight the total mismatch of the ground truth and segmented boundaries with $\mathrm{DBD}_G = 0$ for d1 and $\mathrm{DBD}_M = 0$ for d2. SBD gives the two images scores of 0.278 and 0.335, which are sufficiently low to indicate segmentation errors in both cases, and are non-equal to point out that the errors are different.

Images d9–d12, e9–e12, and s9–s12 with simulated salt-and-pepper errors have been constructed to analyse the sensitivity of the considered metrics to random segmentation errors. The overlap-based metrics class d12, e12, s12 as near-perfect segmentation results and rank them higher than d1–d4 and d9–d11, e1–e11, s1–s11, respectively. However, boundary-distance-based metrics like HD rank images d12, e12, s12 as second worst in their corresponding image sequences. The high but not the best attainable similarity scores that SBD assigns to these images indicate three properties: first, the random errors are reflected in the scores; second, the errors in these images affect the segmentation quality less than the errors in images like e1–e8, s1–s8; third, such random errors are easier to eliminate in segmentation post-processing.

Fractals

Additional to the simple shapes, the synthetic data contains a sequence of fractals (Table 5.3). These help reveal how the metrics react to boundary errors, where the boundary consists of increasingly intricate detail. The initial region is an isosceles triangle (image f0). At each step of the recursion, new triangles with the same ratio of side lengths (4:7) are added on the equal sides of each triangle. The images from steps 3, 6, and 9 of the recursion are assigned as ground truth in turn, producing the results reported in Figure 5.5.

The graphs illustrate interesting patterns. Images f0–f2 (f0–f5, f0–f8) are undersegmented, and images f4–f12 (f7–f12, f10–f12) are oversegmented. Note, on the one hand, that the conventional metrics DSC (and JSC), TPVF, HD, ASSD show more variability for the undersegmented images—they fail to differentiate properly the oversegmented images. On the other hand, TNVF, Prec are insensitive to undersegmentation errors. The directional variants of the new metric show similar behaviour to the first and the second group of metrics respectively. RVD performs relatively better, but still fails to capture differences at the level of local neighbourhoods on the region boundary.

By contrast, SBD easily identifies the under- and oversegmentation errors, and reflects those in its scoring. It shows high sensitivity to even small differences between the boundaries of the ground truth and segmented images, and uses a wide scoring range to illustrate those differences. For DSC, HD, or ASSD, the differences in score for images f0–f2 (f0–f5, f0–f8) are significantly higher than for images f4–f12 (f7–f12, f10–f12). Unlike the existing metrics, SBD uses a similar range of values on both sides of the ground truth. These examples showcase how well SBD outperforms all the other metrics, especially for segmentation errors near the boundary of the intended region.

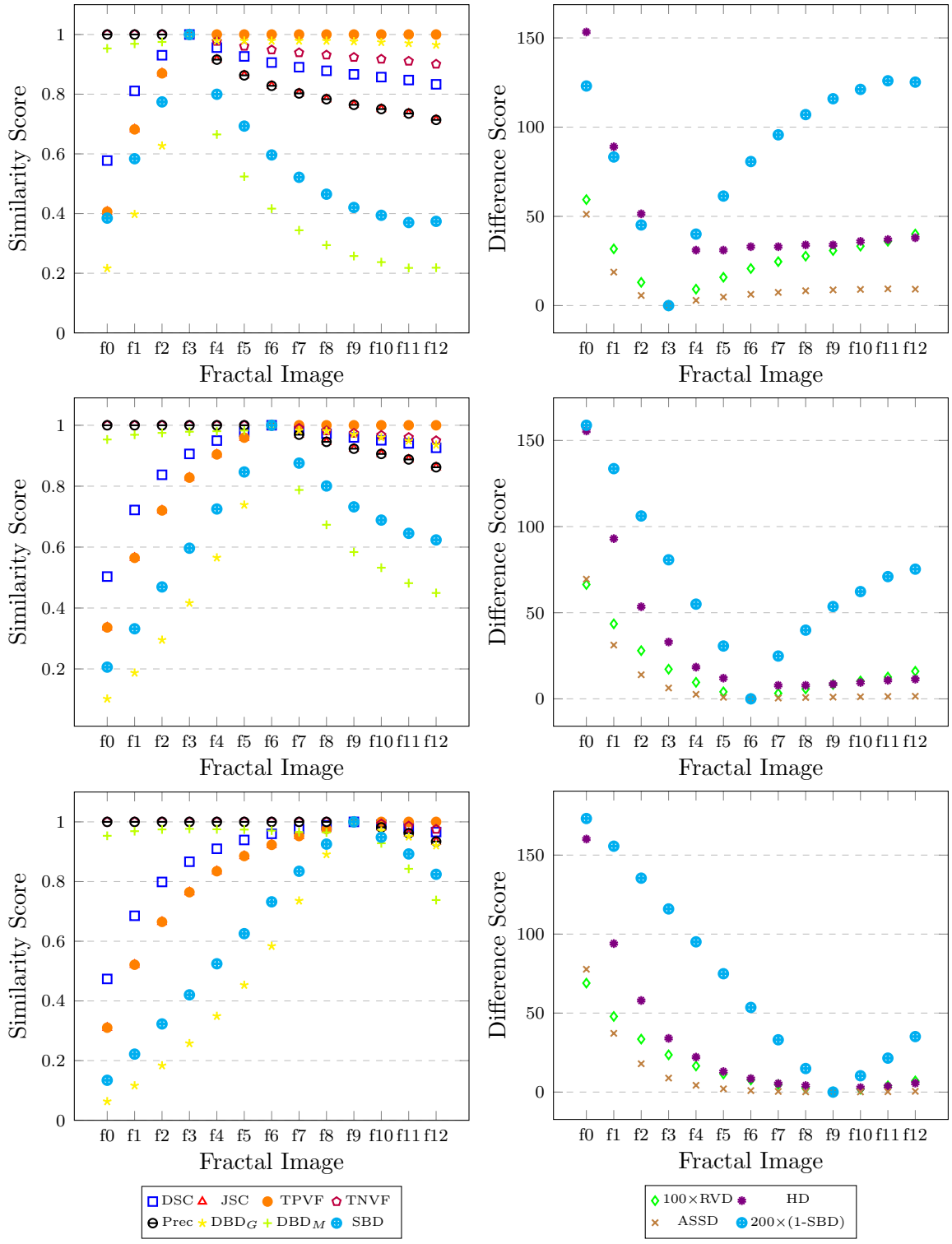


Figure 5.5: Results for synthetic fractals. Fractals at steps 3, 6, and 9 of the recursion considered as ground truth, respectively.

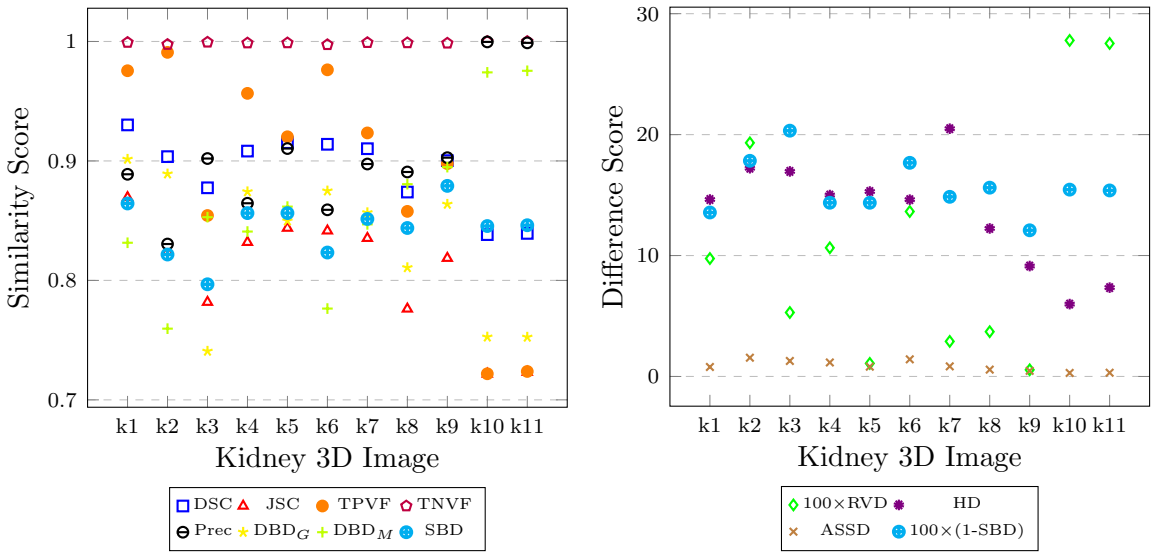
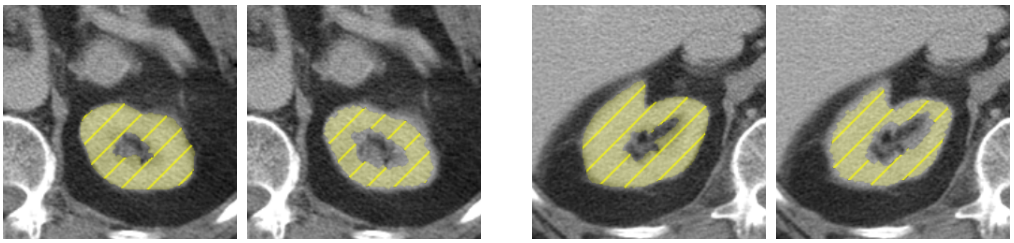


Figure 5.6: Results for medical kidney scans, i.e. 3D regions in 3D images. Results for eleven kidneys (left or right) from nine 3D medical CT scans, between 1 and 38 slices each. Please note that some overlap-based scores are identical in the left graph for a few images and their plots are not easily traced.

5.3.2 Real scan data

We reconsider the results from Section 4.5 of the automated segmentation algorithm (see also [136]) on the nine different CT scan 3D images, between 1 and 38 slices each. We automatically labelled 13 kidneys (left or right), for which we also manually produced ground truth masks. The results of validating the segmented 3D images against the ground truths are presented in Figure 5.6. This time, our main interest is the comparison of the behaviour of the various metrics rather than the quality of the segmentation.

Since the machine segmentation results are produced by a single automated algorithm, errors of a similar nature are expected in all cases. The small ranges of score values produced by the various metrics illustrate this. TNVF values are all above 99%, due to the kidney occupying only a small portion of the 3D image voxels. TPVF and Prec scores are complementary on most images: one of these metrics is only sensitive to undersegmentation and the other one—to oversegmentation. As was the case with 2D synthetic images, SBD tends to penalise errors more harshly,



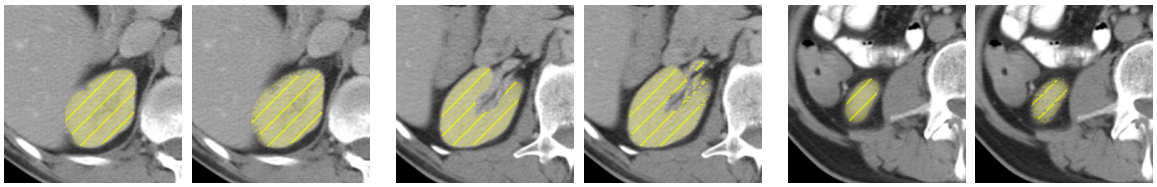
(a) **(b)**

Figure 5.7: (a) the ground truth and the automated segmentation of k10; (b) the ground truth and the automated segmentation of k11. In both cases the kidney is slightly under-segmented.

producing smaller similarity scores (and hence, higher difference scores) than most existing metrics. In this experiment its value range agrees better with DSC, JSC, and HD.

When there is great disagreement between classes of metrics or between metrics inside a class, like in k10–k11, SBD acknowledges the errors without being overly sensitive to the error. These two kidney segmentations are ranked higher than k1–k9 by Prec, HD, and ASSD. At the same time, DSC, JSC, TPVF, RVD penalise these segmentations ranking them lowest. SBD is the only metric that does not yield extreme scores for these two cases; its directional variants (DBD_G and DBD_M) do, as do the two groups of conventional metrics.

The great disagreement of the existing metrics for these two particular kidneys is explained by the variation in the number of slices in the images considered. Images k10–k11 consist of only a single CT slice. The number of CT slices for k1–k9 is at least 5, where k1 is the biggest image with 38 slices. We present the ground truth and the automated segmentation of k10 and k11 in Figure 5.7. For comparison, we report a few slices from k1 in Figure 5.8. This is a typical example where, although DSC is high (93%), at the same time the images show a poor segmentation around the boundary: 5.8(b) leaks and 5.8(c) presents salt-and-pepper errors (of the sort we simulated in the synthetic data). A further study of the effect of image size on the metric scores is presented in Section 5.3.3.



(a) **(b)** **(c)**

Figure 5.8: The ground truth and the automated segmentation of k1 captured from three different slices in (a)–(c). Both undersegmentation and oversegmentation can be observed in different slices. Random errors similar to our simulated salt-and-pepper are (just about) visible inside (c).

In order to preserve the level of detail in the graphs, Figure 5.6 does not include data points for two kidney scans with pathological cases where the ground truth disagreed strongly with the machine segmentation. For these images, the machine segmented 3D region was more than 3.5 times larger than the ground truth (because the human operator decided not to include any necrotic tissue in the ground truth). SBD scored these kidney segmentations around 0.33 and 0.38, sharing the sensitivity of other metrics: DSC of 0.28 and 0.34, and HD of 97 and 85 mm. RVD flags this stark difference and should always be applied first, as a sanity check.

5.3.3 Further tests

Ranking tests

We set up a small experiment in order to understand the correlation between segmentation rankings produced with evaluation metrics and end user expectations. For that, we asked four researchers in image segmentation to rate the synthetic fractal images (Table 5.3) against the ground truth image f6. They were offered a discrete scoring range of 0 to 10, where 0 indicates segmentation failure and 10 indicates that the segmented region matches the ground truth perfectly. There was a mutual agreement that this scoring range was adequate to rank the images (f0–f12) unambiguously.

The comparison of the averaged human ranking (mapped from 0–10 onto $[0, 1]$)

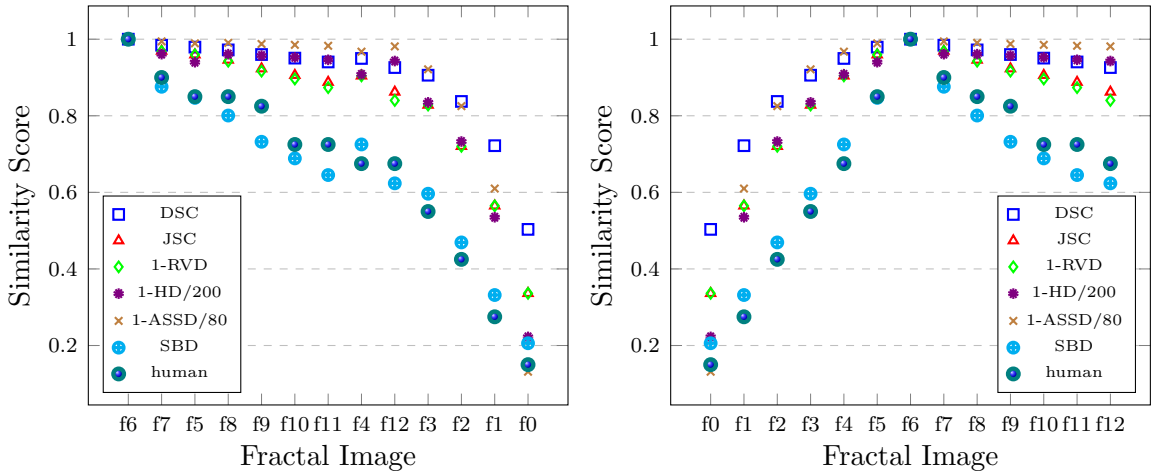


Figure 5.9: The ranking of the fractal images with different metrics with the ground truth set at f6. The images are sorted in the order of human ranking in the first graph; the second graph preserves the order of fractal recursion.

of the fractal images and the considered metrics, including SBD, is reported in Figure 5.9. We exclude those metrics that produce the best attainable scores on one or the other side of the ground truth—TPVF, TNVF, Prec (see Figure 5.5). Difference metrics have been scaled and converted into corresponding similarity metrics to fit the scoring range between 0 and 1. As the graphs show, the human ranking correlates best with SBD with a single ranking disagreement at f4. In addition, their maximum absolute difference in score per image stands at less than 9.3%. For comparison, corresponding differences between the human ranking and DSC or $1 - \text{RVD}$ stand at more than 44.6% or 29.5%. These are encouraging initial results for possible future study of ranking properties of evaluation metrics versus human perception of segmentation results.

Parameterisation tests

In addition to comparing SBD to existing and commonly used metrics, we analysed how the SBD performance changes based on its parameterisation. Consistently changing the Moore neighbourhood radius from 1 to 5 for the synthetic data, we reveal that SBD similarity scores gradually increase for larger radii (for most images).

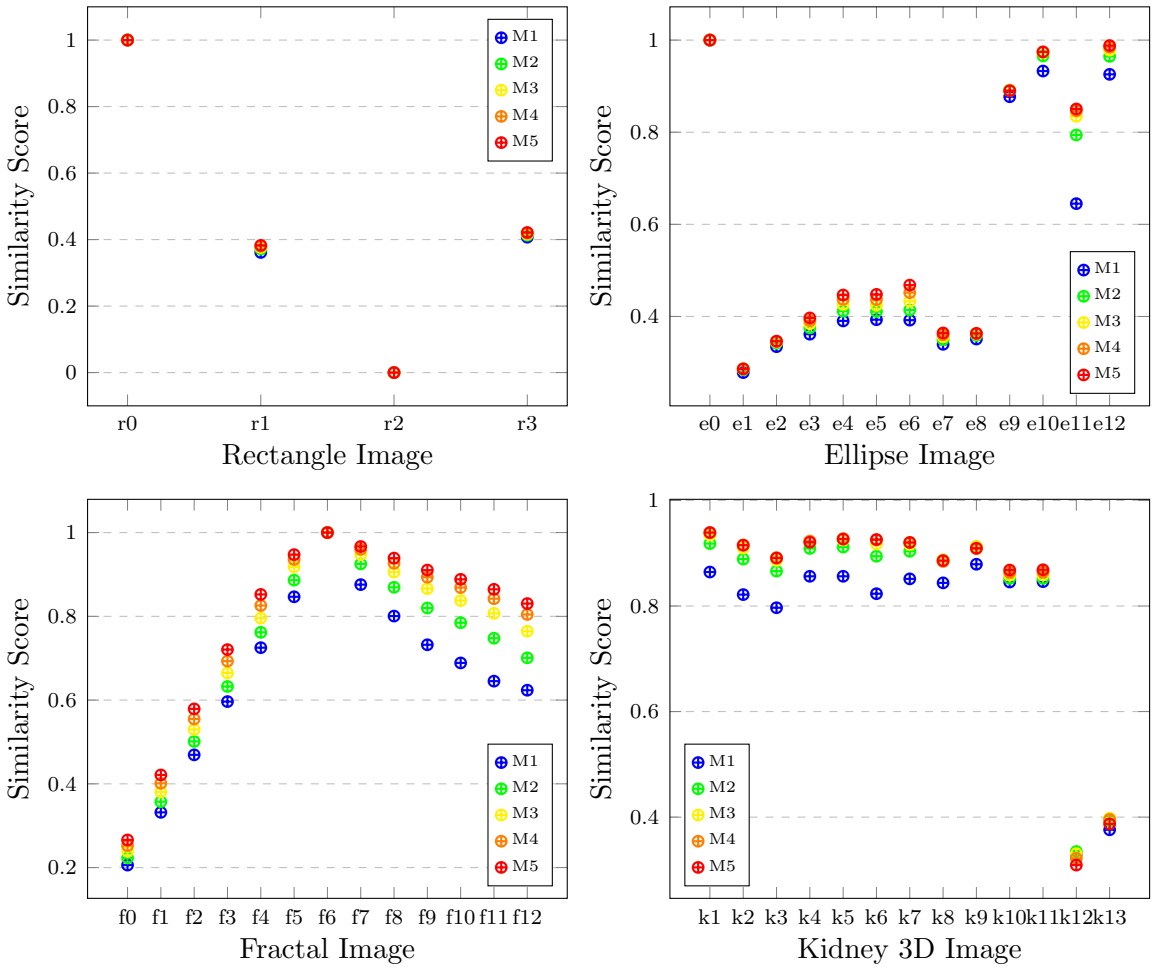


Figure 5.10: Comparison of SBD results for rectangle, ellipse, and fractal (ground truth at step 6) images and medical kidney 3D images using Moore neighbourhoods of radius 1 to 5. Please note that some SBD scores with different neighbourhood radii are equal or have very close values for a few images, and their plots are not easily traced.

Table 5.4: Value ranges and average values of SBD results over 13 star images using Moore neighbourhoods of radii 1–5

Moore Radius	1	2	3	4	5
Value Range	0.246–1	0.258–1	0.265–1	0.272–1	0.276–1
Average	0.509	0.534	0.543	0.548	0.552

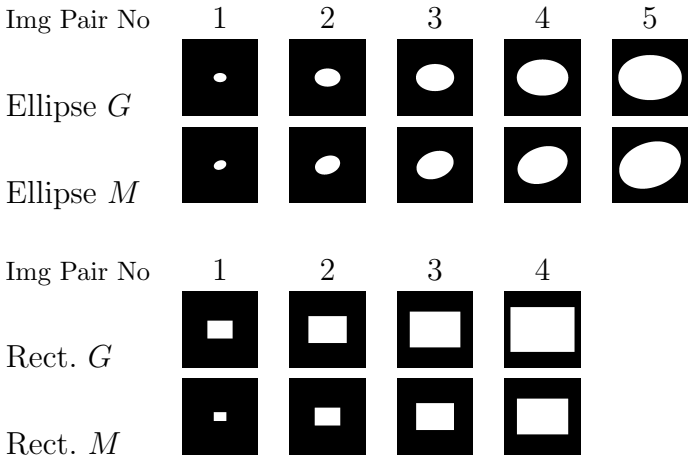
The growth is around 9–10% between radius 1 to 5 in average, although for specific segmentation results it reaches 46–47%. Our overall conclusion here is that the simple neighbourhood with radius 1 should be generally preferred (in the case of fixed resolution experiments). Increasing the size of neighbourhood has only a minor effect on similarity score values and ranges but extends execution times.

The sample graphs in Figure 5.10 depict how the SBD values vary with the change in Moore neighbourhood radius from 1 to 5. The growth of the neighbourhood size causes only a tiny rise in the similarity score for most images. For a few images this is slightly more noticeable, e.g. ellipse images e6 and e11 feature a score increase of 30–32%.

When the neighbourhood size increases, more internal points of the segmented regions end up labelled as boundary points. Those, in turn, introduce higher local similarity Dice scores; thus, for the new extended set of local scores, the average increases. Table 5.4 presents additional data for star images illustrating this trend. The lower bounds of value ranges and average scores (over 13 star images) form increasing sequences, from lower neighbourhood sizes to larger ones. It is important to note that the average score rise is only from 0.509 to 0.552. This growth in scores for larger neighbourhoods means (only slightly) decreased sensitivity to segmentation errors; therefore, we propose generally using simple neighbourhoods of radius 1.

Finally, we considered how the execution time increases for larger neighbourhood radii. We implemented a Python SBD scoring script for a sequence of synthetic images against a single ground truth for a user-set neighbourhood radius. The real execution

Table 5.5: A small collection of simulated machine segmentation (M) and ground truth (G) image pairs with monotonically increasing region size based on ellipse and rectangle shapes. The image resolution is set at 512×512.



time of the programme ranged between 10.8 seconds and 3 minutes 7.4 seconds (12 seconds and 3 minutes 22.3 seconds) for the 13 fractal images, where image f3 (f6) was taken as ground truth. The lowest and highest running times in these ranges correspond to radii 1 and 10. In both cases the execution times show monotonic growth with the increase of the radius between 1 and 10. This fact supports our preference for smaller neighbourhoods.

Size and resolution tests

A further aspect of interest is the response of the metrics to variations in the size of segmented regions and in image resolution. Table 5.5 lists the contents of a small synthetic dataset that was constructed to analyse the impact of region size and image resolution variations on metric scores. The corresponding evaluation results for region size variation are reported in Table 5.6. Most existing metrics for the five ellipse image pairs behave very differently from the same metrics for the four rectangle image pairs. An exception to this pattern is seen for TNVF and Prec. For the five ellipse image pairs, TNVF is absolutely insensitive to the errors in the image pairs with smaller regions (ellipse image pairs 1–2) and slightly more sensitive for the larger regions

Table 5.6: The impact of region size variation on metric scores using simulated segmentation images with ellipse and rectangle shapes.

EllipPair	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD
1	0.921	0.854	0.922	0.999	0.921	0.000	5.000	2.606	0.502
2	0.923	0.857	0.923	0.995	0.922	0.001	8.944	5.307	0.460
3	0.922	0.855	0.922	0.987	0.922	0.000	13.454	8.150	0.441
4	0.922	0.855	0.922	0.974	0.922	0.000	17.692	10.974	0.434
5	0.922	0.855	0.922	0.950	0.921	0.000	21.932	13.816	0.428
RectPair	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD
1	0.397	0.248	0.248	1.000	1.000	0.752	52.202	36.748	0.264
2	0.614	0.443	0.443	1.000	1.000	0.557	53.600	36.225	0.319
3	0.720	0.562	0.562	1.000	1.000	0.438	51.865	35.720	0.342
4	0.780	0.639	0.639	1.000	1.000	0.361	52.431	35.717	0.355

(ellipse image pairs 4–5). Prec is stably high for the five ellipse image pairs. The best attainable TNVF and Prec scores for the four rectangle image pairs are explained by their total insensitivity to undersegmentation.

The other overlap-based metrics (DSC, JSC, TPVF) and the size-based RVD metric maintain approximately the same value for all five ellipse image pairs. The boundary-distance-based metrics (HD, ASSD), on the other hand, gradually increase for these examples indicating higher dissimilarity for the image pairs with larger region sizes. SBD is in agreement with the boundary-distance-based metrics for these examples, considering the match of the smaller regions in ellipse image pairs 1–2 to be better than the match of the larger regions in ellipse image pairs 4–5.

DSC, JSC, TPVF, and RVD show significant variation for the four rectangle examples: image pairs 1–2 get significantly lower similarity (higher difference) scores than image pairs 3–4. On the other hand, HD and ASSD values are relatively stable for the rectangle image pairs. SBD agrees with the former metrics, considering rectangle image pair 1 to be the worst, and rectangle image pair 4 the best out of the four cases.

The slow decrease in SBD scores for ellipse image pairs is explained by the local

Table 5.7: The impact of image resolution variation on metric scores using simulated segmentation images with ellipse shape. A single pair of ellipse G and M images is considered at resolutions from 128×128 to 2048×2048.

Resolution	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD
128x128	0.923	0.857	0.924	0.987	0.921	0.003	3.606	1.898	0.535
256x256	0.923	0.857	0.923	0.987	0.923	0.000	6.708	3.932	0.476
512x512	0.922	0.855	0.922	0.987	0.922	0.000	13.454	8.150	0.441
1024x1024	0.922	0.855	0.922	0.987	0.922	0.000	26.401	16.645	0.424
2048x2048	0.922	0.855	0.922	0.987	0.922	0.000	52.802	33.475	0.419

overlaps in the neighbourhoods of boundary points shrinking as the regions get bigger and their boundaries diverge. In the case of the rectangles, the distance between the boundaries stays the same; hence, the local overlap scores do not change. It is the ratio of the boundary lengths that affects the weights of the local overlaps and leads to the small increase in SBD values for the rectangle image pairs. The monotonic change of the SBD scores in accordance with region size is inherent to its definition and reflects its hybrid nature between overlap metrics and boundary distance metrics.

In Table 5.7, we report how image resolution affects the scores of the metrics. DSC, JSC, TPVF, TNVF, Prec, and RVD maintain their values for all resolutions. In case of HD and ASSD, the scores increase by a factor of 2 in parallel with image resolution because our calculation does not take into account physical dimensions of pixels. In a clinical setting, those dimensions would decrease by a factor of 2 when the resolution increases, thus keeping the metric scores to the original values.

SBD scores go down slightly as image resolution increases, as reported in Table 5.7. Increasing image resolution is equivalent to decreasing neighbourhood size for SBD, since it makes the region boundary thinner. Selecting larger SBD neighbourhood radii for higher resolution images would keep SBD scores at the same level.

For general use, when experimental results for images of varying pixel resolution need to be compared, the neighbourhood radius for SBD can be chosen based on the physical dimensions of pixels. For instance, the neighbourhood radius can be set to 1

for 4 millimetres pixel images, set to 2 for 2 millimetres pixel images, and set to 4 for 1 millimetres pixel images. This will keep the SBD value range consistent for images of different resolutions.

5.4 Comparison of boundary overlap methods

In Section 5.2.1 we defined a number of new boundary-overlap-based metrics. So far, in our experiments for comparison against conventional metrics we used only one of them—SBD. This was done in order to prove the concept without overburdening the discussion and the performance graphs. We now turn to analyse the performance of all the boundary overlap metrics in the family.

In Figure 5.11, we report the comparison of all the new metrics on the fractal synthetic data with ground truth f6. Figure 5.12 shows the corresponding results for the real kidney data. From the results in the first set of graphs, the behaviour of all the metrics is very similar except the Boundary True Negative Fraction (both directional and symmetric). SBTN is insensitive to the undersegmented fractal steps f0–f5, but on images f7–f12 it performs relatively similar to the other metrics. The insensitivity is a consequence of the peculiar definition of this metric. Around ground truth boundary points, it gives the best attainable local neighbourhood score if the machine segmentation is absent from that neighbourhood: $TNVF(N_x) = \frac{|N_x|-|M(N_x)\cup G(N_x)|}{|N_x|-|G(N_x)|} = 1$ if $|M(N_x)| = 0$.

Once again, in Figure 5.12, the SBTN scores disagree with the rest, this time assigning much lower scores in each case. In cases where the machine segmentation boundary includes the ground truth (oversegmentation), this leads to the overall SBTN score being lower, thus penalising such segmentations. $TNVF(N_y) = \frac{|N_y|-|M(N_y)\cup G(N_y)|}{|N_y|-|G(N_y)|} = 1 - \frac{|M(N_y)|}{|N_y|}$ if $|G(N_y)| = 0$ where $y \in \partial M$. When the machine segmentation boundary is included inside the ground truth (undersegmentation), the

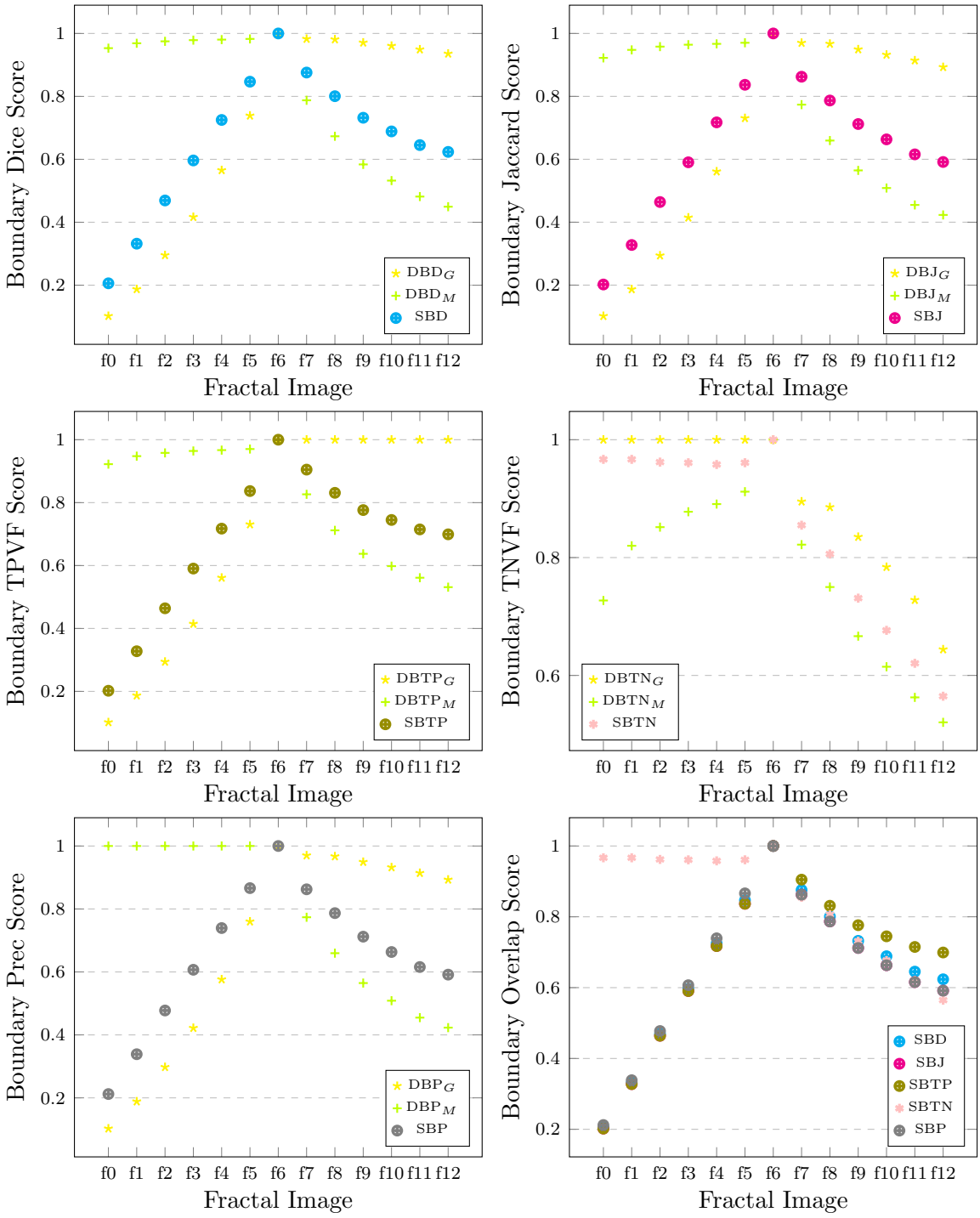


Figure 5.11: Different boundary overlap metrics (their directional and symmetric variants) compared on the synthetic fractal images with ground truth f6. The first five graphs correspond to Boundary Dice, Jaccard, TPVF, TNVF, and Prec metrics. The sixth graph combines the symmetric variants of all five metrics.

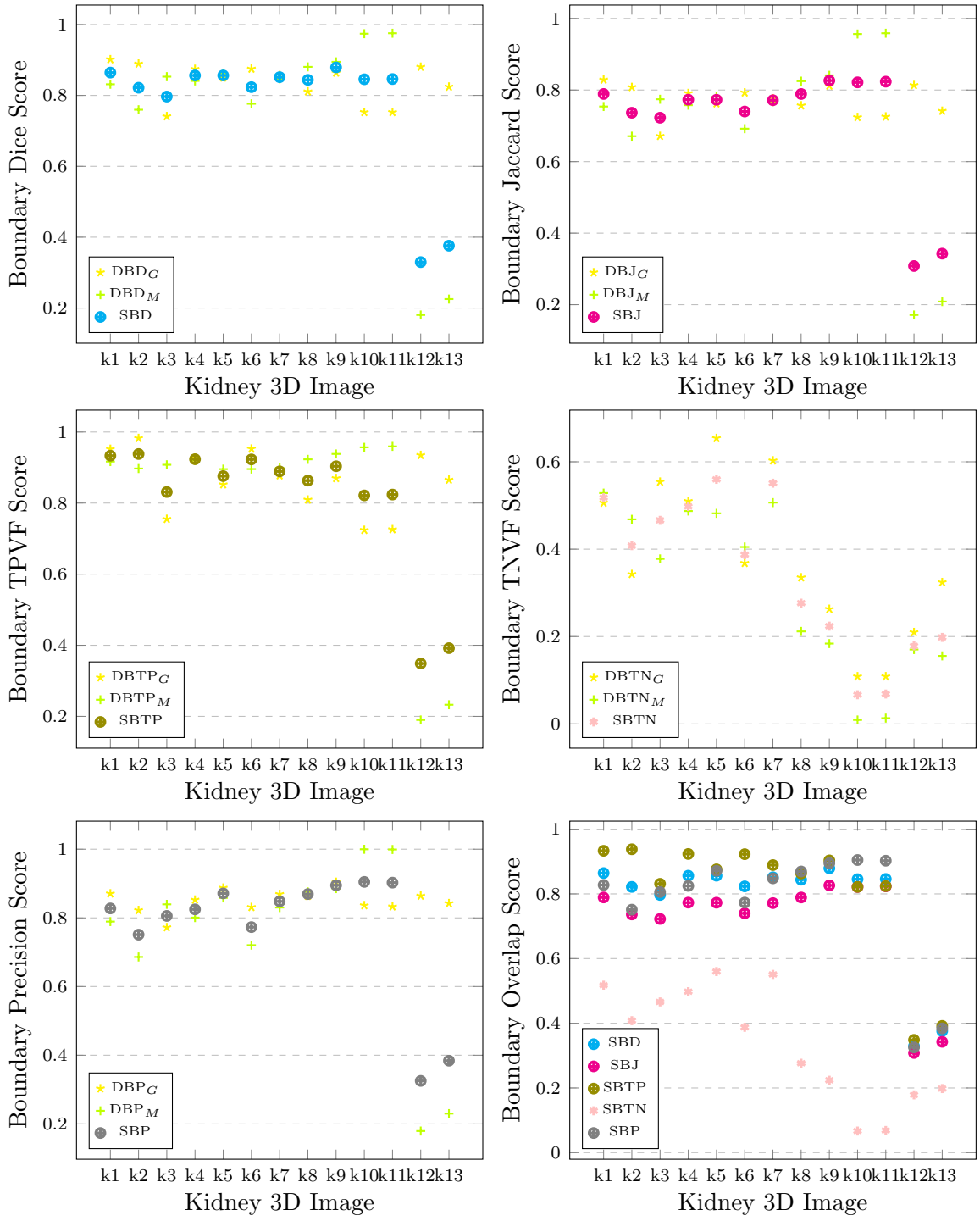


Figure 5.12: Different boundary overlap metrics (their directional and symmetric variants) compared on the real kidney 3D images. The first five graphs correspond to Boundary Dice, Jaccard, TPVF, TNVF, and Prec metrics. The sixth graph combines the symmetric variants of all five metrics.

SBTN score is higher, due to an increase in the local TNVF. However, when the undersegmentation is substantial, i.e. a considered neighbourhood is fully inside the ground truth, the local TNVF value is not defined. The latter is the case with our particular kidney segmentations; hence, the overall SBTN score is lower.

We explain the occasional disagreement of the SBD, SBJ results with SBTP and SBP (and between SBTP and SBP) by the directional complementary nature of the overlap-based TPVF and Prec metrics used in the local neighbourhoods. Hence, on some of the examples, the insensitivity to the local errors causes higher scores with these metrics. For instance, kidney images k1, k2, k4, k6 get high SBTP scores, while high SBP scores are recorded for images k5, k10, k11.

Unsurprisingly, the metrics introduced in this chapter inherit features from the corresponding overlap-based metrics. For instance, the value range of the new metrics is between 0 and 1 (similar to the overlap-based metrics), where 1 indicates a perfect match and 0—a total mismatch. Or, SBD and SBJ are linked, with SBJ values always being slightly smaller than SBD. However, they do not always produce the same ranking patterns. In the case of SBTN, when a local neighbourhood size is fixed, an important feature is that it be independent of the size of the whole image (unlike the original TNVF metric). This fact is reflected in the huge difference of the value ranges of TNVF (see Figure 5.6, ≥ 0.997) and SBTN (Figure 5.12, 0.066–0.56).

To summarise, Figure 5.13 illustrates the comparison between the boundary overlap family and the existing metrics in a final set of graphs, using the synthetic fractal images with ground truth at f6 as well as the kidney 3D scans. In order to preserve clarity, only symmetric variants of the new metrics are included in these graphs.

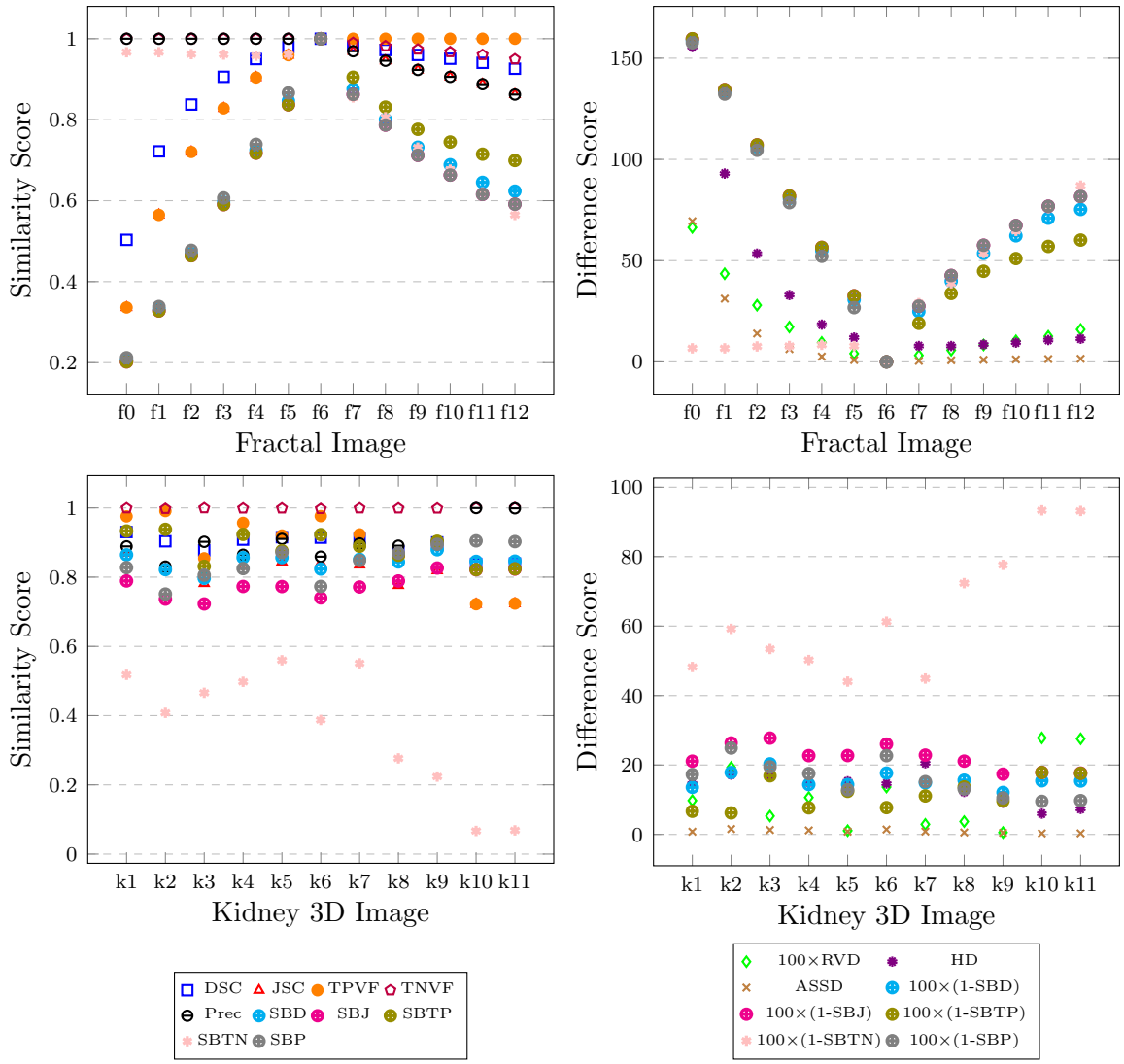


Figure 5.13: The performance of the conventional metrics against the proposed family of Symmetric Boundary Overlap metrics on the examples of the synthetic fractal images with ground truth at f6 and the kidney 3D images (excluding k12–k13).

5.5 Summary

Our results demonstrate how, in contrast to existing metrics, SBD reacts to a wider range of error types. It penalises errors more, producing a wider spread of similarity scores (lower for more errors). SBD does not incur score inflation, especially in cases where existing metrics disagree (one metric gives a high score and another gives a low one to the same segmentation result). The fractal experiments demonstrate that SBD is fit to evaluate segmentation results which are prone to error in the neighbourhood of the region boundaries. Other members of the boundary overlap family have similar properties to SBD.

In order for our family of metrics to be adopted into general use, it will be necessary to prove its effectiveness by conducting systematic studies of large publicly available medical datasets. Meanwhile, we propose SBD as the best and easiest to use out of the 15 metrics in the family because it is easy to implement, quick to run, and fulfils all the listed criteria for a good metric.

Chapter 6

Path Reducing Watershed for the GPU

6.1 Introduction

The watershed transform interprets input greyscale images as topographic terrain maps (see Section 3.2) where the greyscale value of a pixel is used as an altitude value. The image gets divided into regions around minima, where a downhill path exists between every pixel in a segmented region and its region minimum.

Modern segmentation applications usually involve large amounts of data. In fields like medical image analysis 3D images of hundreds of millions of voxels are not rare. Conventional sequential algorithms prove to be very slow for such large images. In recent years, with the advent of affordable programmable GPU devices, other parallel watershed algorithms have been designed specifically for GPU implementation.

We introduce a novel parallel watershed procedure designed for GPU implementation. This algorithm constructs paths of steepest descent for all image pixels and reduces these paths into direct pointers to catchment basin minima in logarithmic time. Our watershed algorithm simplifies label initialisation in minimal plateaux and

converts label propagation into fast path reduction. Other watershed algorithms usually expect pre-processed input in the form of a lower-complete image in order to resolve non-minimal plateaux. In contrast, our proposed algorithm deals successfully with non-minimal plateaux in the original input.

We propose three different GPU implementation variants of our parallel watershed. Our algorithm is easy both to understand and to implement, and it outperforms existing GPU watershed algorithms. We thoroughly discuss the merits of our watershed for images of different sizes and textures.

6.2 Path reducing watershed

6.2.1 Path reducing parallel watershed

Our parallel watershed procedure consists of four major steps:

S1. initialisation of pixel states and labels (into four groups)

S2. resolution of non-minimal plateaux

S3. label propagation, or path reduction

S4. merging of minimal plateau labels

Since the path reduction is central to the algorithm we propose, we hereafter refer to it as the path reducing watershed (PRW).

This procedure can be classified as a watershed transform based on topographic distance: it relies on paths of steepest descent. An alternative view on the four steps is: (S1) initialisation of paths of steepest descent; (S2) resolution of paths on non-minimal plateaux; (S3) path reduction into pointers to regional minima (local minima and minimal plateaux); (S4) addition of extra paths inside minimal plateaux



Figure 6.1: Simple 2D neighbourhoods: Moore (left) and von Neumann (right). The inclusion of the central grid point in the neighbourhood is task dependent.

to achieve single pointer target (i.e. label) per catchment basin. Algorithm 3 provides details of the four steps of the path reducing watershed.

If I is the set of all pixel positions in the image then for every $p \in I$ we can consider the local neighbourhood $N(p)$—the set of positions that are a unit distance away from p in the image. The 2D 8-pixel (3D 26-voxel) Moore neighbourhood and 2D 4-pixel (3D 6-voxel) von Neumann neighbourhood are the popular options, see Figure 6.1.

The initialisation step relies on the classification of the image pixels into four groups, or states, based on the image values in the pixel neighbourhoods. A fixed ordering of all the image positions is essential to the suggested algorithm; here we consider row-major order. Let q be the pixel position in the neighbourhood $N(p)$ with the smallest input image value. If the minimum is not uniquely identified, q is set to the largest (according to the chosen ordering) among the positions with the minimum image value in that neighbourhood:

$$q = \max\{s \in N(p) | \forall t \in N(p), image(s) \leq image(t)\},$$

where $image(t)$ is the input image value at position $t \in I$.

- If the value of the input image at q is smaller than at p, then q is the direction of steepest descent from p. We can set the initial label of p to q, so that it later borrows its final label from q. In addition, we assign pixel p into the first group of pixels with lower neighbours ($state(p) = 0$).

Algorithm 3 Path reducing parallel watershed

1: **for all** $p \in I$ in parallel **do** ▷ Step S1: initialisation of pixel labels and states
2: $q \leftarrow \max\{s \in N(p) | \forall t \in N(p), image(s) \leq image(t)\}$
3: **if** $image(q) < image(p)$ **then**
4: $label(p) \leftarrow q;\ state(p) \leftarrow 0$
5: **else if** $image(q) > image(p)$ **then**
6: $label(p) \leftarrow p;\ state(p) \leftarrow 1$
7: **else if** $q > p$ **then**
8: $label(p) \leftarrow q;\ state(p) \leftarrow 2$
9: **else if** $q < p$ **then**
10: $label(p) \leftarrow p;\ state(p) \leftarrow 3$
11: **end if**
12: **end for**

13: **repeat** ▷ Step S2: resolution of non-minimal plateaux
14: **for all** $p \in I$ in parallel **do**
15: **if** $state(p) \geq 2$ **and** $\exists q \in N(p), state(q) = 0 \wedge image(q) = image(p)$ **then**
16: $label(p) \leftarrow q;\ newstate(p) \leftarrow 0$
17: **else**
18: $newstate(p) \leftarrow state(p)$
19: **end if**
20: **end for**
21: $swap(state, newstate)$
22: **until** $\neg change$

23: **repeat** ▷ Step S3: label propagation, or path reduction
24: **for all** $p \in I$ in parallel **do**
25: **for** $i \leftarrow 1, RR$ **and** $label(p) \neq label(label(p))$ **do**
26: $label(p) \leftarrow label(label(p))$
27: **end for**
28: **end for**
29: **until** $\neg change$

30: **repeat** ▷ Step S4: merging of minimal plateau labels
31: **for all** $p \in I$ in parallel **do**
32: **if** $state(p) \geq 2$ **then**
33: **for all** $q \in N(p), state(q) \geq 2$ **do**
34: $lp \leftarrow label(p);\ lq \leftarrow label(q)$
35: **while** $label(lp) \neq label(lq)$ **do**
36: $label(lp) \leftarrow \min(label(lp), label(lq))$
37: $label(lq) \leftarrow \min(label(lp), label(lq))$
38: **end while**
39: **end for**
40: **end if**
41: **while** $label(p) \neq label(label(p))$ **do**
42: $label(p) \leftarrow label(label(p))$
43: **end while**
44: **end for**
45: **until** $\neg change$

- If the input image value at q is larger than at p, then p is a local minimum. Then $label(p) = p$ and later the whole catchment basin will borrow this value for a label. The set of local minimum pixels is the second group with state fixed at 1.

- Finally, if the image values at p and q are the same, then p is part of a plateau. We differentiate these into two groups: plateau pixels with at least one equal neighbour ordered after them (state 2) and plateau pixels with equal neighbours ordered only before them (state 3). In case the plateau is confirmed as minimal after step S2, plateau pixels at state 2 will borrow their label from plateau pixels at state 3. Thus, $label(p)$ is initialised as q if $state(p) = 2$, and $label(p) = p$ is a self-loop if $state(p) = 3$.

Figure 6.2 details the procedure of the path reducing watershed on a small example step by step. A simple 6×2 example with middle greyscale input image values is presented, detailing the procedure of path reducing watershed with von Neumann neighbourhood. Pixels with a change are highlighted in red; considered but not changed pixels are green. Numbers in the range 100–107 represent greyscale values; roman numerals represent pixel indices and labels; numbers 0–3 correspond to the four possible pixel states.

To recap, the state table corresponding to the image contains the following values:

- 0 when the pixel has a lower neighbour,

- 1 when the pixel is a local minimum,

- 2 when the pixel belongs to a plateau and has at least one equal neighbour ordered after itself,

- 3 when the pixel belongs to a plateau and equal neighbours are ordered only before itself.

100	105	105	105	104	104	I	II	III	IV	V	VI
102	104	105	106	104	107	VII	VIII	IX	X	XI	XII

(a) input 6×2 pixel image with the expected watershed line; pixel indices

1	0	2	0	2	3	I	I	IX	V	XI	VI
0	0	0	0	3	0	I	VII	VIII	XI	XI	XI

(b) step S1: initial states and labels

1	0	0	0	2	3	I	I	II	V	XI	VI
0	0	0	0	3	0	I	VII	VIII	XI	XI	XI

(c) step S2: updated states and labels in non-minimal plateaux

1	0	0	0	2	3	I	I	I	XI	XI	VI
0	0	0	0	3	0	I	I	I	XI	XI	XI

(d) step S3: labels of minima propagated uphill to all pixels

1	0	0	0	2	3	I	I	I	VI	VI	VI
0	0	0	0	3	0	I	I	I	VI	VI	VI

(e) step S4: minimal plateau labels merged

Figure 6.2: A simple 6×2 example with middle greyscale input image values detailing the procedure of path reducing watershed with von Neumann neighbourhood. Pixels with a change are highlighted in red; considered but not changed pixels are green.

The second step of the algorithm resolves non-minimal plateaux by iteratively propagating information inwards from the plateau boundaries. For every pixel p in states 2 or 3, $state(p) \geq 2$, if there is a pixel q in its neighbourhood such that $state(q) = 0$ and the input image values are the same at p and q, then the state of p is reset to 0 and its label is updated to equal q. This process is repeated iteratively, using a buffer for the states and swapping the two state arrays after each iteration, until there are no further state changes.

By the end of step S2 the states of the image pixels can be interpreted in a new way, slightly different from the initial state groups. State 0 indicates either a pixel with a lower neighbour or a non-minimal plateau pixel. A pixel in state 1 is still a local minimum. Finally, states 2 and 3 correspond to pixels only in minimal plateaux. Since labels are essentially pointers from one pixel to a neighbouring one or itself, these labels can be followed recursively to form directed paths between pixels. For all the pixels in state 0 their labels form paths leading to minima: pixels in

states 1–3. **These are the paths of steepest descent forming the different catchment basins**. Furthermore, inside minimal plateaux the labels of state 2 pixels form additional paths into the state 3 pixels. Thus, any label path starting at any pixel will eventually lead to a pixel in state 1 or 3, where the label is a self-loop.

Step S3 reduces all the label paths into direct pointers from any image pixel into the corresponding catchment basin minimum, i.e. state 1 or 3 pixel. This can be done independently for every pixel in parallel by following the label pointers until a self-loop is reached. However, such a procedure is linear in the length of the longest label path, hence linear in the image size in the worst case. Instead, we reduce the complexity to logarithmic in the image size by synchronising all labels every fixed number of updates and repeating the procedure iteratively. We call this number of updates between synchronisations the *reduction rate (RR)*. For every pixel p its label is updated to its label's label RR times, and then the new value of the label is recorded instead of the original $label(p)$ and made available to other pixels to read. This process is repeated iteratively until there are no further label changes. The overall number of operations is $O(RR \times \lceil \log_{RR+1} L \rceil) = O(\lceil \log_{RR+1} L \rceil) \leq O(\log_{RR+1} |I|)$, where RR is the reduction rate, L is the length of the longest path, $|I|$ is the image size.

Figure 6.3 illustrates the path reduction process. The reduction of a path of length 22 with a reduction rate 2 is achieved in only $\lceil \log_3 22 \rceil = 3$ iterations.

In the case of a 1D image, the label array after algorithm steps S1–S3 would be a valid watershed result. This is true because every (minimal) plateau in 1D is an interval and only the rightmost plateau pixel receives state 3. Hence the label array after algorithm step S3 will contain only one label value per catchment basin. However, this property breaks for higher dimensionality images: there may be several state 3 pixels in a minimal plateau, depending on its shape. Figure 6.4 contains a few shapes for minimal plateaux with highlighted state 3 pixels.

Step S4 solves the issue of multiple labels for minimal plateaux by merging those

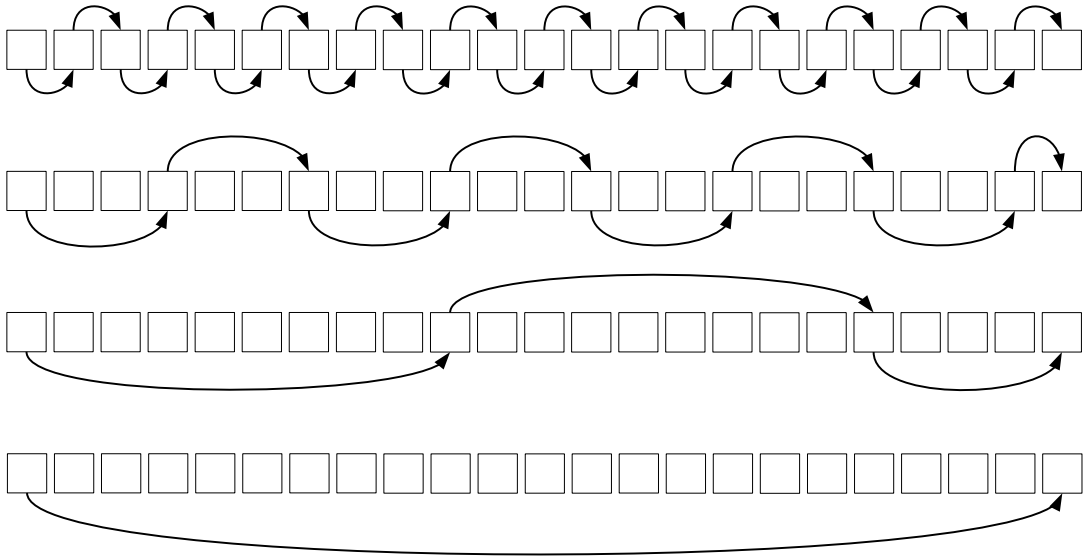


Figure 6.3: The reduction of a path of length 22 with a reduction rate 2 in only $\lceil \log_3 22 \rceil = 3$ iterations.

labels. The label array can be viewed as a disjoint set where the height of every tree is at most one after step S3. For neighbouring pixels p, q in a minimal plateau, $lp = label(p), lq = label(q)$ are the roots of the corresponding trees. If $lp \neq lq$, then the two trees are merged by setting the smaller label as the parent of the larger label. Since the suggested procedure is for parallel implementation, more than two trees may be merged at the same time. Thus the check and the following value update are repeated until equality is reached between $label(lp)$ and $label(lq)$. A **label path reduction loop** follows for all pixels to reset the tree heights to the maximum of one, i.e. converting label paths into direct pointers to self-loop labels. This process of merging and reducing is repeated iteratively until there are no further label changes. After the completion of step S4, there is only one label value per catchment basin, and for each pixel p, $label(p)$ denotes which catchment basin p belongs to.

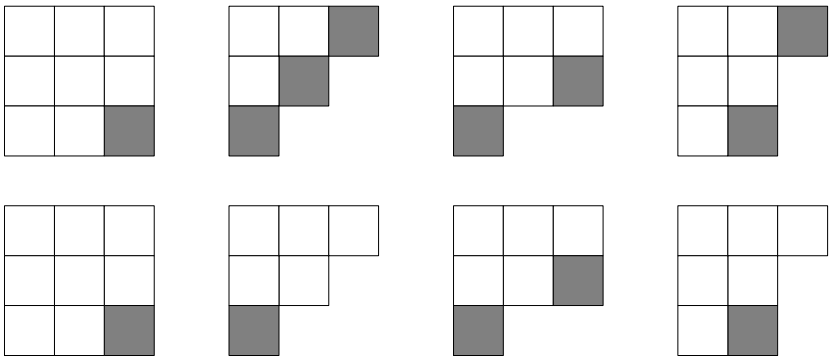


Figure 6.4: A few shapes for minimal plateaux with state 3 pixels highlighted in grey. Von Neumann neighbourhood in the first row; Moore neighbourhood in the second.

6.2.2 GPU synchronous implementation

We consider three different implementations of PRW in Algorithm 3. Our watershed implementations use CUDA [3], the parallel computing platform and programming model invented by NVIDIA. The first and the simplest implementation of PRW is described here and it is called *synchronous* path reducing watershed (PRW_{sync}). The other two are presented in Section 6.2.3. All three variants of the algorithm assign separate threads to every pixel in the image, and use von Neumann neighbourhood of size four in 2D and six in 3D.

PRW_{sync} consists of a host code function and four CUDA kernels, one for each step of the procedure. A kernel is a function which is executed by multiple threads at the same time on the GPU device. The four kernels in PRW_{sync} correspond to and implement the lines 2–11, 15–19, 25–27 and 32–43 in Algorithm 3, respectively. The host code is described in Algorithm 4 and is the same for all three variants of the watershed algorithm.

The most expensive step of PRW, and PRW_{sync} in particular, is the resolution of non-minimal plateaux. Unlike steps S1 and S3–S4, the number of iterations of the code block in lines 4–7 in Algorithm 4 (13–22 in Algorithm 3) may be linear in the

Algorithm 4 The host code of the PRW algorithm

```
 1: HostToDevice(image)                          ▷ Copy image from host to device
 2: DimGrid, DimBlock                            ▷ Set grid and block dimensions
 3: InitialisationKernel                         ▷ Step S1
 4: repeat                                       ▷ Step S2
 5:     ResolutionKernel
 6:     swap(state, newstate)
 7: until ¬change
 8: repeat                                       ▷ Step S3
 9:     ReductionKernel
10: until ¬change
11: repeat                                       ▷ Step S4
12:     MergingKernel
13: until ¬change
14: DeviceToHost(label)                          ▷ Copy labels from device to host
```

image size in the worst case. For instance, if the image is the simple $(2n+2)\times 1$ example

$$105 \quad \underbrace{109 \quad 109 \quad 109 \quad \ldots \quad 109 \quad 109}_{2n} \quad 106,$$

then it takes n iterations to complete step S2 of PRW_{sync} because of the expensive global inter-block synchronisation after each state update. The pixel states are initialised in step S1 as

$$1 \quad 0 \quad \underbrace{2 \quad 2 \quad \ldots \quad 2}_{2n-2} \quad 0 \quad 1;$$

after $n-2$ iterations of step S2 we have

$$1 \quad \underbrace{0 \quad 0 \quad \ldots \quad 0}_{n-1} \quad 2 \quad 2 \quad \underbrace{0 \quad 0 \quad \ldots \quad 0}_{n-1} \quad 1.$$

An $(n-1)^{\text{th}}$ iteration converts the remaining 2-states into 0 and a final n^{th} iteration does not make any changes and completes the resolution of non-minimal plateaux.

In addition, since the state update for the plateau pixels in lines 15–19 of Algorithm 3 takes place in parallel and the order of update completion for various pixels is non-deterministic, we need the buffer state memory *newstate* to ensure correctness, i.e. fair and symmetric propagation of labels inside plateaux. The use of the buffer also entails fair division of non-minimal plateaux among adjacent catchment

basins. For the considered example with $n = 5$, the label values after the completion of watershed step S2 are guaranteed to be

$$1 \leftarrow 0 \leftarrow 0 \leftarrow 0 \leftarrow 0 \leftarrow 0 \qquad 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 1,$$

where the numbers are the pixel states, arrows are the labels. The final labelling after watershed step S3 (step S4 is not required in 1D), using roman numerals instead of arrows, would be

$$\text{I} \quad \text{I} \quad \text{I} \quad \text{I} \quad \text{I} \quad \text{I} \quad \text{XII} \ \text{XII} \ \text{XII} \ \text{XII} \ \text{XII} \ \text{XII}.$$

Thus, the sizes of non-minimal plateaux inside an image are expected to largely affect the parallel watershed execution time. Such trends are documented in existing literature [98, 63]; our results in Section 6.3 agree with this assumption.

6.2.3 Block-asynchronous and balanced implementations

In addition to PRW_{sync}, we introduce two more implementations of PRW which attempt to speed up the watershed. This is done by using the intra-block synchronisation mechanisms of CUDA. Block-level shared memory provides faster access to data to be shared among different threads; in-block synchronisation is computationally cheaper than global synchronisation. We use these block-level instruments to reduce the number of global iterations of the non-minimal plateau resolution step. Coalesce memory access ensures more efficient operations.

The *block-asynchronous* path reducing watershed ($\text{PRW}_{\text{async}}$) is different from PRW_{sync} in the second step of the algorithm. While the host code is exactly the same, as given in Algorithm 4, the CUDA kernel is responsible for multiple state updates using synchronisation at the level of blocks. Algorithm 5 details the procedure for this kernel.

Algorithm 5 Non-minimal plateau resolution kernel of $\text{PRW}_{\text{async}}$

```
1: GlobalToLocal(label)                                   ▷ Copy label from global to local memory
2: GlobalToShared(image)                                  ▷ Copy image from global to shared memory
3: GlobalToShared(state)                                  ▷ Copy state from global to shared memory
4: repeat
5:     SyncThreads                                        ▷ Synchronise threads in block
6:     SharedToLocal(state)                               ▷ Copy state from shared to local memory
7:     if state(p) ≥ 2 and ∃q ∈ N(p), state(q) = 0 ∧ image(q) = image(p) then
8:         label(p) ← q; state(p) ← 0                     ▷ Update local label and state
9:     end if
10:    SyncThreads                                        ▷ Synchronise threads in block
11:    LocalToShared(state)                               ▷ Copy updated state from local to shared memory
12: until ¬blockchange
13: LocalToGlobal(label)                                  ▷ Copy updated label from local to global memory
14: SharedToGlobal(newstate)                              ▷ Copy updated shared state into global newstate
```

The non-minimal plateau resolution kernel of $\text{PRW}_{\text{async}}$ first loads the data into the shared and local memory. Since every pixel needs access to the input image and state values of the neighbouring pixels to update its own state, this data is stored in shared memory. Every pixel p is responsible for copying its $image(p)$ and $state(p)$ values into the shared memory. In addition, pixels at the boundary of the block, also copy the corresponding values of their neighbours outside of the block. For instance, if the block size is 16×16, then the shared memory used will be of size 18×18 and an additional band of $16 \times 4 = 64$ values will be copied by the 60 boundary threads. Figure 6.5 illustrates the structure of shared memory allocated for each thread block. In a 2D image, the four threads at the block corners copy two extra values each; a single extra value is copied by any other boundary thread.

The threads inside the block iteratively update the pixel states until convergence inside the block. Due to the thread synchronisation after each state update inside the block, there is no need for state buffers at this level and the updated state values are put into the same positions in the shared memory. After the completion of the state update iterations, the new states and labels are copied back into global memory (states are copied into the *newstate* buffer).

Each thread block has a separate shared memory space, thus during kernel execu-

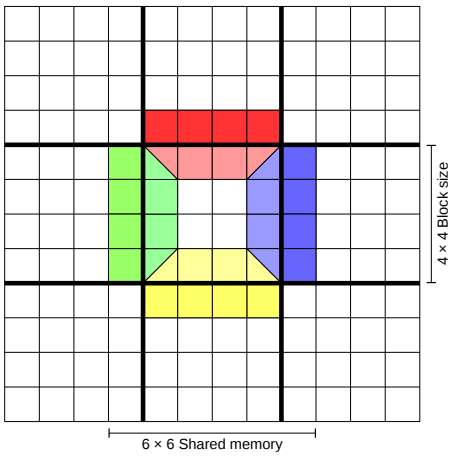


Figure 6.5: The structure of shared memory allocated for each thread block. For a block of size 4×4, memory of size 6×6 is allocated. The threads on the boundary of the block are responsible for copying the adjacent extra values into the shared memory.

tion there is no communication between different blocks. Updates from one block are not accessible to threads in other blocks until the data is copied into global memory and kernel execution is completed, i.e. global synchronisation points. This means that the values in shared memory corresponding to the additional band of pixels are never changed during kernel execution (they are changed in *their own* block) and maintain their values from the previous global synchronisation point. Information is propagated from one block to another once per global iteration.

Since information propagation speeds inside and across thread blocks are non-uniform, the division of non-minimal plateaux among adjacent catchment basins is affected by discontinuities. Labels and states are propagated more rapidly inside blocks thus resulting in watershed lines[1] near block boundaries.

Figure 6.6 demonstrates the issue on a simple example. The image of size 128×128 contains two square-shaped minimal plateaux in black and a white non-minimal plateau that covers the rest of the image. The correct watershed line, produced with PRW_{sync}, is a smooth line that fairly and symmetrically divides the non-minimal

[1]Note, that PRW does not label any pixels as belonging to watershed lines. Rather, neighbouring pixels belonging to different catchment basins are conventionally referred to and illustrated in black or white in figures as watershed lines.

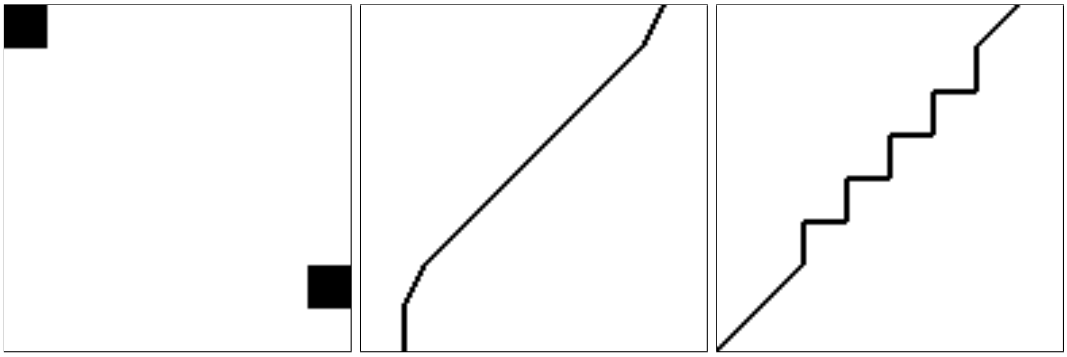


Figure 6.6: An image of size 128×128 (left); correct watershed line produced with PRW_{sync} (middle); watershed line of $\text{PRW}_{\text{async}}$ tracking the 16×16 block boundaries (right). Example inspired by a similar one in [98].

plateau between the two watershed regions. By contrast, the watershed line of $\text{PRW}_{\text{async}}$ tracks the 16×16 block boundaries. This example is inspired by a similar one in [98].

The third variant of our watershed—*balanced* block-asynchronous path reducing watershed (PRW_{bal})—makes use of the block-level synchronisation mechanisms, like $\text{PRW}_{\text{async}}$, while maintaining the fairness of watershed line positioning, similar to PRW_{sync}. We extend the 0–3 state space into negative numbers thus: when $state(p) < 0$ then value $|state(p)|$ denotes the distance of a non-minimal plateau pixel from the plateau boundaries, i.e. the number of state update operations for information to reach p. Algorithm 6 describes the non-minimal plateau resolution step (kernel) for PRW_{bal}.

After a non-minimal-plateau-pixel changes its state from 2 or 3 to a negative value, its state can be iteratively updated to larger values (still smaller than 0) during further in-block iterations or after global synchronisation points. The execution of the resolution step will complete only when no more improvements can be made to the states of the non-minimal plateau pixels. On the simple 14×1 pixel example

105 109 109 109 109 109 109 109 109 109 109 109 109 106,

Algorithm 6 Non-minimal plateau resolution kernel of $\mathrm{PRW}_{\mathrm{bal}}$

```
1: GlobalToLocal(label)                                  ▷ Copy label from global to local memory
2: GlobalToShared(image)                                 ▷ Copy image from global to shared memory
3: GlobalToShared(state)                                 ▷ Copy state from global to shared memory
4: repeat
5:     SyncThreads                                       ▷ Synchronise threads in block
6:     SharedToLocal(state)                              ▷ Copy state from shared to local memory
7:     if (state(p) ≥ 2 and ∃q ∈ N(p), state(q) ≤ 0 ∧ image(q) = image(p)) or
              (∃q ∈ N(p), state(p) + 1 < state(q) ≤ 0 ∧ image(q) = image(p)) then
8:         label(p) ← q;  state(p) ← state(q) − 1        ▷ Update local label and state
9:     end if
10:    SyncThreads                                       ▷ Synchronise threads in block
11:    LocalToShared(state)                              ▷ Copy updated state from local to shared memory
12: until ¬blockchange
13: LocalToGlobal(label)                                 ▷ Copy updated label from local to global memory
14: SharedToGlobal(newstate)                             ▷ Copy updated shared state into global newstate
```

assuming block size 3, the first global iteration of step S2 produces states

$$\underbrace{1 \quad 0 \quad -1}\quad \underbrace{2 \quad 2 \quad 2}\quad \underbrace{2 \quad 2 \quad 2}\quad \underbrace{-3 \quad -2 \quad -1}\quad \underbrace{0 \quad 1}.$$

After the second global iteration we get

$$\underbrace{1 \quad 0 \quad -1}\quad \underbrace{-2 \quad -3 \quad -4}\quad \underbrace{-6 \quad -5 \quad -4}\quad \underbrace{-3 \quad -2 \quad -1}\quad \underbrace{0 \quad 1};$$

and this is corrected after iteration 3:

$$\underbrace{1 \quad 0 \quad -1}\quad \underbrace{-2 \quad -3 \quad -4}\quad \underbrace{\mathbf{-5} \quad -5 \quad -4}\quad \underbrace{-3 \quad -2 \quad -1}\quad \underbrace{0 \quad 1}.$$

The corresponding label pointers (illustrated as arrows between greyscale values) look like

$$105 \leftarrow \underbrace{109 \leftarrow 109 \leftarrow \cdots \leftarrow 109}_{6} \qquad \underbrace{109 \rightarrow 109 \rightarrow \cdots \rightarrow 109}_{6} \rightarrow 106,$$

thus producing a fair division of the non-minimal plateau.

$\mathrm{PRW}_{\mathrm{bal}}$ is slightly slower than $\mathrm{PRW}_{\mathrm{async}}$ due to (a) extra state updates and (b) extra global iterations of the resolution step. Overall, $\mathrm{PRW}_{\mathrm{bal}}$ shows a large runtime

improvement upon PRW_{sync}, especially on images with large plateaux, as will be illustrated in Section 6.3.2.

6.2.4 Implementation details

The memory requirements for PRW_{sync} and $\text{PRW}_{\text{async}}$ are similar: per pixel one byte for input image value; one byte for pixel state and another for new state buffer; four bytes for pixel label. That sums up to seven bytes per pixel. In addition, $\text{PRW}_{\text{async}}$ also relies on block-level shared memory usage for algorithm step S2. This includes input image values, pixel states and a flag for iterations. For a block of size 16×16, the shared memory per thread block will be $(16 + 2) \times (16 + 2) \times (1 + 1) + 1 = 649$ bytes.

Since PRW_{bal} uses a larger state space, the requirements for memory are 13 bytes per pixel: one byte for input image value; four bytes for pixel states and four bytes for new state buffer; four bytes for pixel label. Similarly, the shared memory size per 16×16 block is (16+2)×(16+2)×(1+4)+1 = 1,621 bytes. For all three variants of the watershed we use only global memory to store information on the device. No texture or constant memory is used in order to keep the current implementations simple. Future experiments with other memory types intended to fine-tune implementations may slightly decrease execution times.

The main parameters of the path reducing watershed affecting execution times are the thread block size and the reduction rate in algorithm step S3. The block size should be chosen based on the image size and its dimensionality. The reduction rate can be optimised depending on the image size, the size and the number of catchment basins. Our general suggestions for the block size are 16×16 for 2D images and 8×8×8 for 3D images. An RR value in the range 5–10 can be set as a default but should be changed to optimise performance in case of specific characteristics of considered images.

The path reducing watershed algorithm is dimensionality independent; the procedure can be applied to 1D, 2D, or 3D images changing only the structure of the pixel neighbourhood $N(p)$. A 3D implementation of PRW_{sync} works also on 1D or 2D images. However, this approach is not well-suited for implementations of $\text{PRW}_{\text{async}}$ and PRW_{bal}. The extra band of values triples the size of shared memory if a 3D implementation is used for a 2D image instead of a 2D implementation. Thus, our implementations of $\text{PRW}_{\text{async}}$ and PRW_{bal} are separate for 2D and 3D images.

Execution times of watershed transform in general, and of path reducing watershed in particular, are highly dependent on the input image. The time required to process images with a small number of large catchment basins can be a few times larger than the execution of the algorithm on images with a high quantity of smaller catchment basins. We try to reflect this variability in our experimental results by considering a substantial number of images of different size and quality.

6.3 Results

Although our main motivation for designing these algorithms is related to medical images, it is hard to find suitable medical benchmarking data in the public domain. Instead, in order to evaluate the performance of the suggested watersheds on a consistent set of images, we have used the Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500) [15, 2], an extension of the original BSDS300 [85]. We converted the images to greyscale without any further processing before running the watersheds. We chose this dataset so that future work in this direction can be easily compared against ours. Also, testing on 500 different images allows for a varied number of watershed catchment basins in a fixed image size (481×321 or 321×481) to be considered.

For a given image, the equivalence of the different watershed procedures is established by checking that they consistently produce the same number of regions. For the

BSDS500 set the number of watershed catchment basins is in the range 3,372–22,094. For clarity of presentation, the images are sorted in increasing order of the PRW_{sync} execution times.

6.3.1 Qualitative results

Our main interest in watershed is its application on 2D or 3D medical scan images to construct hierarchical segmentations [48, 47]. When applied to large data, the efficiency of watershed is crucial. The pipeline accepted in the medical imaging community is: smoothing the input image, constructing the gradient magnitude image and only then applying the watershed transform for watershed lines to correspond to object boundaries. Figure 6.7 illustrates this process and the watershed results on three sample images.

The first image in Figure 6.7 is the classic Lena test image [14] in greyscale, the second is a slice from an abdominal CT scan, and the third—a slice from a knee MRI scan. The input greyscale (0–255) images are presented in the first row. The gradient magnitude images (Section 2.1.2) constructed after a single application of 3×3 Gaussian blur are in the second row. Finally, the third row depicts the results of the watershed applied on the gradient magnitude images (see Section 3.2). The results are illustrated in the form of the original images on which the watershed lines are highlighted in white. These images are constructed by taking the greyscale values from the original input image for pixels inside watershed catchment basins and by setting the maximum greyscale value for pixels on boundaries of catchment basins.

Our quantitative experiments presented below skip the smoothing and the gradient magnitude constructing steps, directly applying watershed on input greyscale images.

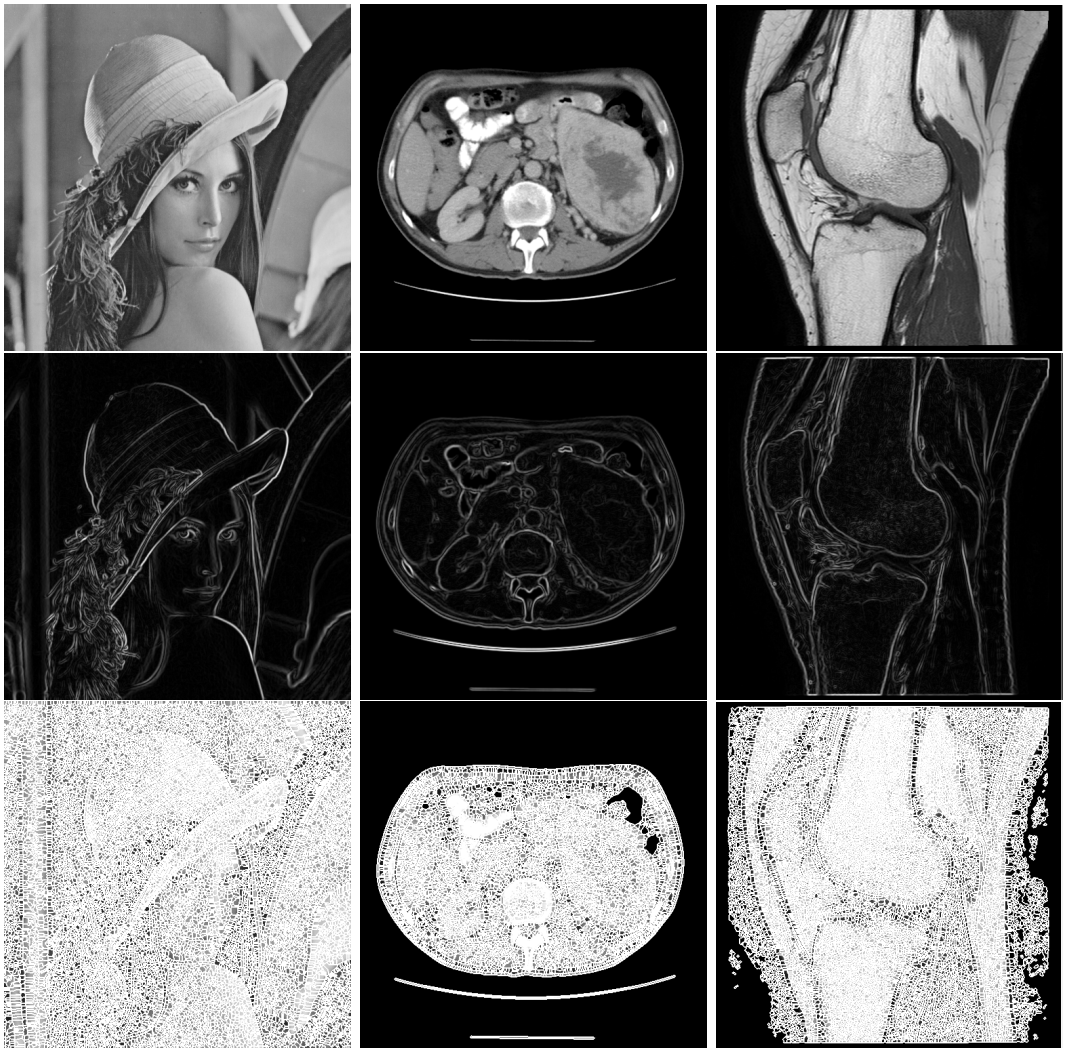


Figure 6.7: $\mathrm{PRW}_{\mathrm{bal}}$ results on three sample images: Lena, an abdominal CT slice, and a knee MRI slice. The original greyscale images are in the first row; gradient magnitude images after minimal Gaussian blurring in the second row; the original images overlaid with watershed lines in white in the third row.

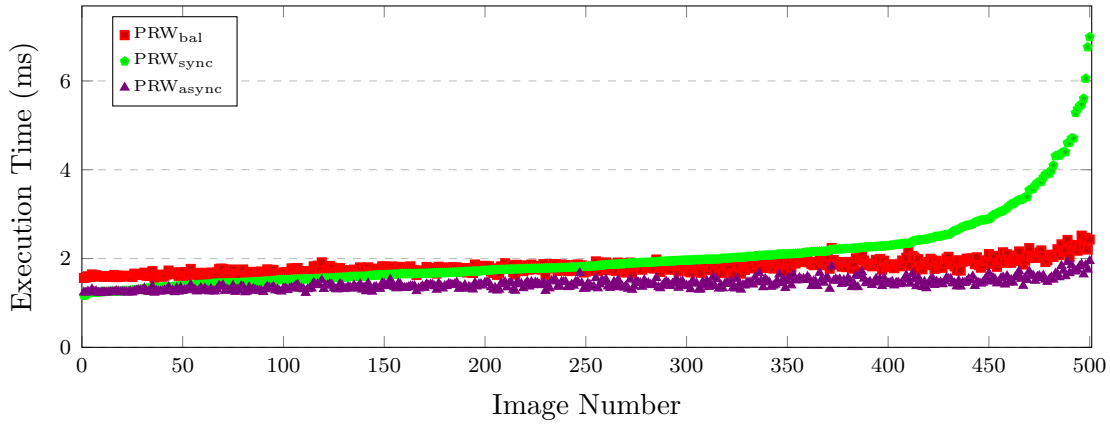


Figure 6.8: The execution times (average of ten runs per image) of the three variants of our watershed compared on the BSDS500 images. For clarity of presentation the images are sorted in increasing order of the PRW$_{sync}$ execution times.

6.3.2 Synchronous vs. block-asynchronous

We compare the performance of the three variants of our path reducing watershed algorithm: synchronous, block-asynchronous and balanced block-asynchronous. Their execution times on the BSDS500 images are reported in Figure 6.8 sorted in increasing order of the PRW$_{sync}$ runtimes. The block size is fixed at 16×16 for this experiment and RR is 5 (an extended discussion of the implementation parameters follows below). The PRW$_{sync}$ implementation is the slowest with the sum of the execution times over the 500 images at 1,023.4002 milliseconds (ms). PRW$_{async}$ with 709.5406 ms is faster than PRW$_{bal}$ (906.2504 ms) on every image in the dataset.

Initially, PRW$_{sync}$ is slightly faster than PRW$_{async}$, and hence PRW$_{bal}$, for 12 images. But as non-minimal plateaux become larger the performance of PRW$_{sync}$ deteriorates and it becomes the slowest on 265 images. This is the effect of the global synchronisation after each iteration of the non-minimal plateau resolution (algorithm step S2). The number of these iterations is as low as 3 for images 1–2 and is as high as 101, 129 and 97 for images 498–500, respectively. The average PRW$_{sync}$ execution time over BSDS500 stands at 2.0468 ms, starting at 1.1648 ms on image number 1 and reaching 6.9894 ms on image 500.

On 488 images out of 500 $\text{PRW}_{\text{async}}$ has the best performance, successfully diminishing the time consumption of the resolution of non-minimal plateaux. The in-block synchronisation of the non-minimal plateau paths decreases the number of the global synchronisations to 8–9 for images 498–500. On average, $\text{PRW}_{\text{async}}$ runs for 1.4191 ms; the lowest execution time of 1.2265 ms is on image 111 (for images 1–110 it is in the range 1.23–1.4194 ms, where 1.23 ms is seen on image 9) and the largest of 1.9449 ms on image 500.

PRW_{bal} is on average 1.278 times slower than $\text{PRW}_{\text{async}}$. On 235 images it is also slower than PRW_{sync}. On those 235 images PRW_{bal} is slower by 12.42% on average. On the other 265 images it is 18.89% faster, reaching 67.43% in the extreme case. The average time improvement of PRW_{bal} over PRW_{sync} across all 500 images stands at around 4.18%. The number of global synchronisations reaches 8–9 for images 498–500. By comparison with $\text{PRW}_{\text{async}}$, there is an expected extra iteration or two for part of the images: this is introduced by the balancing of the non-minimal plateau paths among thread blocks. The average execution time of PRW_{bal} stands at 1.8125 ms; the lowest is 1.5628 ms on image 1 and the highest is 2.5157 ms on image 496.

Figure 6.9 shows the execution times and the number of global iterations of the non-minimal plateau resolution step of the three variants of our watershed. Plateau resolution, being the main difference among the three implementations and the most time-consuming step, is behind the variability in the execution times. There is an obvious pattern between the results in Figures 6.8 and 6.9. The execution times of the other steps of the watershed procedures are stable across the 500 images. The watershed runtimes excluding the plateau resolution step (but including steps S1, S3, S4 and memory copies to/from device) are only in the ranges 1.0087–1.2773, 1.0023–1.1926, and 1.3393–1.6188 ms for PRW_{sync}, $\text{PRW}_{\text{async}}$ and PRW_{bal}, respectively. The higher values for PRW_{bal} are explained by the four times larger memory used for storing pixel states (`int` instead of `char`) to accommodate the larger state space.

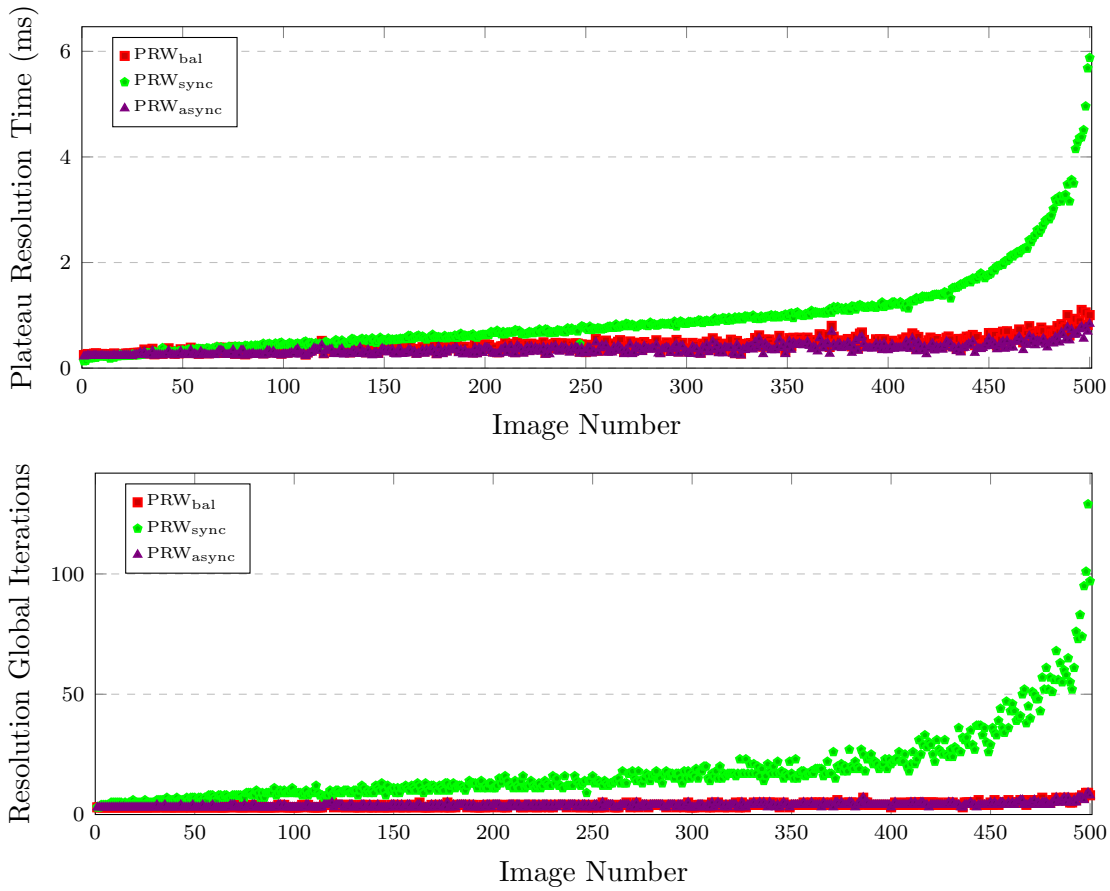


Figure 6.9: The execution times and the number of global iterations of the non-minimal plateau resolution (algorithm step S2) of the three variants of our watershed compared on the BSDS500 images.

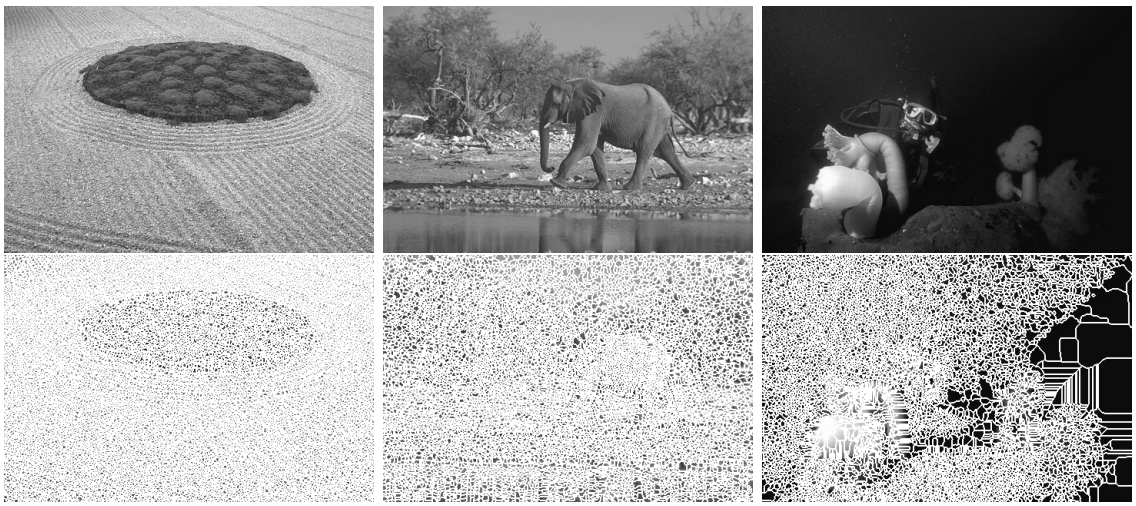


Figure 6.10: Images 1, 250, and 500, and the corresponding $\mathrm{PRW}_{\mathrm{bal}}$ results. Converted-to-greyscale images in the first row; overlays with the watershed lines in white in the second row. (Gradient magnitude was not constructed before running the watershed.)

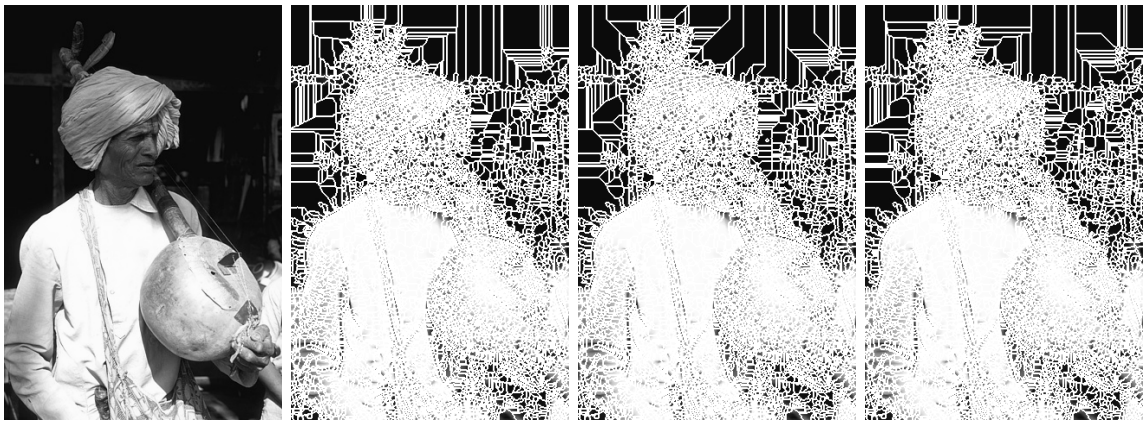


Figure 6.11: Image 498 and the corresponding PRW_{sync}, $\text{PRW}_{\text{async}}$ and PRW_{bal} results, depicted as overlays with the watershed lines in white. Apparent differences in the location of the watershed lines can be observed on the left and at the top of the image.

Images 1, 250, 500 and their overlays with the corresponding watershed lines produced with PRW_{bal} are illustrated in Figure 6.10. The overlay images are produced by overlaying the input greyscale images with the white lines that denote the boundaries of all the watershed regions. Image 1 has 22,094 watershed regions, image 250 has 13,434, and there are 9,307 watershed regions in image 500. Images with fewer and, hence, larger watershed regions usually contain more non-minimal plateaux. Thus the three variants of the algorithm disagree on such images, both in execution time and in watershed line location. Figure 6.11 demonstrates the differences in watershed line locations on non-minimal plateaux on the example of image 498. Apparent differences in how the non-minimal plateaux are divided among catchment basins can be observed on the left and at the top of the image. This is the result of the different patterns of label propagation enforced by the three implementations of PRW.

6.3.3 Comparison against the state-of-the-art

We compare our PRW_{bal} transform against other watershed algorithms, suggested earlier for GPU implementation. Direct comparison of execution times reported in different papers is not immediately possible. Firstly, the GPU devices used belong to

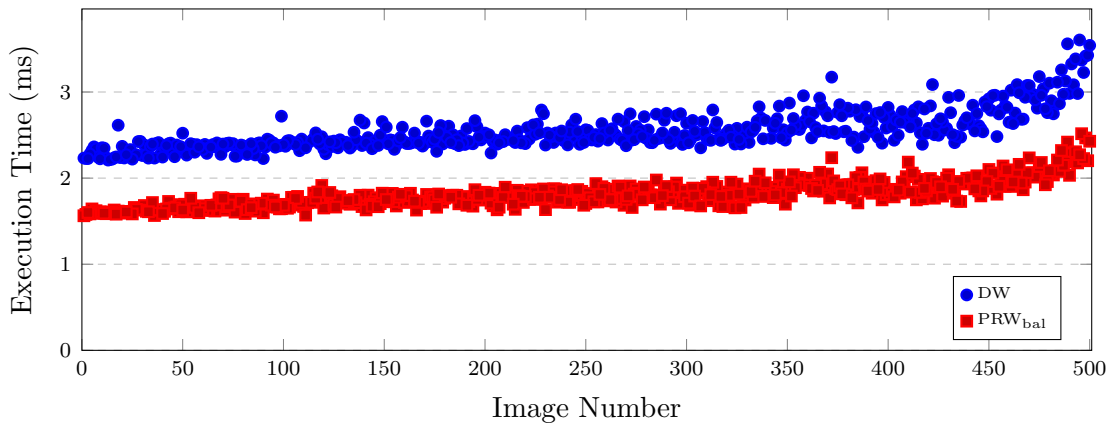


Figure 6.12: The comparison of the execution times (average of ten runs per image) of the DW watershed and our PRW$_{bal}$ implementation on the BSDS500 images, excluding six images on which the DW implementation deadlocks. The sums of the average execution times over the 494 images are 1,257.8308 ms and 895.298 ms respectively. Thus our algorithm shows an execution time improvement of more than 28.8%.

different generations, contain different numbers of streaming machines, CUDA cores, different memory sizes, etc. Secondly, even small differences in input images affect number of watershed catchment basins and execution time. For instance, for the classic Lena image in resolution 512×512, [98] reports 24,958 regions; [63] reports 24,848. If we get Lena from [8] and convert it to greyscale, PRW produces 27,301 regions. However, if we get the greyscale in PGM image from [14], the number of regions by PRW is 26,470. Thus, the only genuine comparison is the execution of the different implementations on the same images on the same device.

Körbes et al. showed in [63] that the CUDA implementation of their DW algorithm (original implementation available from [4]) was superior to CA—an algorithm suggested by Kauffmann and Piché[57, 59]. A more detailed description of DW, including procedure pseudocode, can be found in [124, 125].

A comparison of the DW runtimes against our PRW$_{bal}$ is presented in Figure 6.12. This excludes six images (out of 500) on which the DW implementation [4] deadlocks. We suspect that it is a warp divergence issue but we have not attempted to debug their code to confirm or reject that. In addition, the numbers of watershed catchment basins

produced with DW appear slightly different from the expected numbers on around 26 images out of 500. In order to confirm that, we implemented the DW procedure sequentially in Python following the pseudocode in [124, 125][2]. The numbers of labels produced with this sequential implementation always match the ones produced with PRW and differ by 1–3 from the ones that DW produces on 26 images. In addition, two consecutive runs of DW sometimes, although rarely, produce numbers of labels, differing by 1–2, for the same image.

The sum of the DW average execution times over the 494 images is 1,257.8308 ms; for PRW_{bal} it is 895.298 ms. Thus our algorithm shows an execution time improvement of more than 28.8%. The average DW runtime per image is approximately 2.5462 ms; the lowest is 2.2096 ms on image 13 and the highest is 3.6042 ms on image 495. DW is on average 1.4058 times slower than PRW_{bal}.

Quesada-Barriuso et al. [97, 98] introduced another parallel watershed for GPU based on cellular automata, of which we were unable to find any implementations. However, the authors report 'similar speedup results' over CPU watershed for their 'block-asynchronous proposal of the CA-watershed described' and DW from [63]. From this we conclude that our PRW would easily outperform the CA-based watershed from [98] if executed on the same device.

We do not compare directly against [94, 59, 127] because they have been superseded.

6.3.4 Parameters

We analyse how changing the implementation parameters affects the execution of the watershed transform. Having RR fixed at 5, we consistently change the CUDA block size for PRW_{bal} and measure the overall execution time for the BSDS500 images (the sum of averages over ten runs per image). The results are reported in Figure 6.13.

[2]There is a typo in the DW pseudocode in [124, 125]: in line 39 of Algorithm 1 '<' should be used instead of '=' for an inequality check.

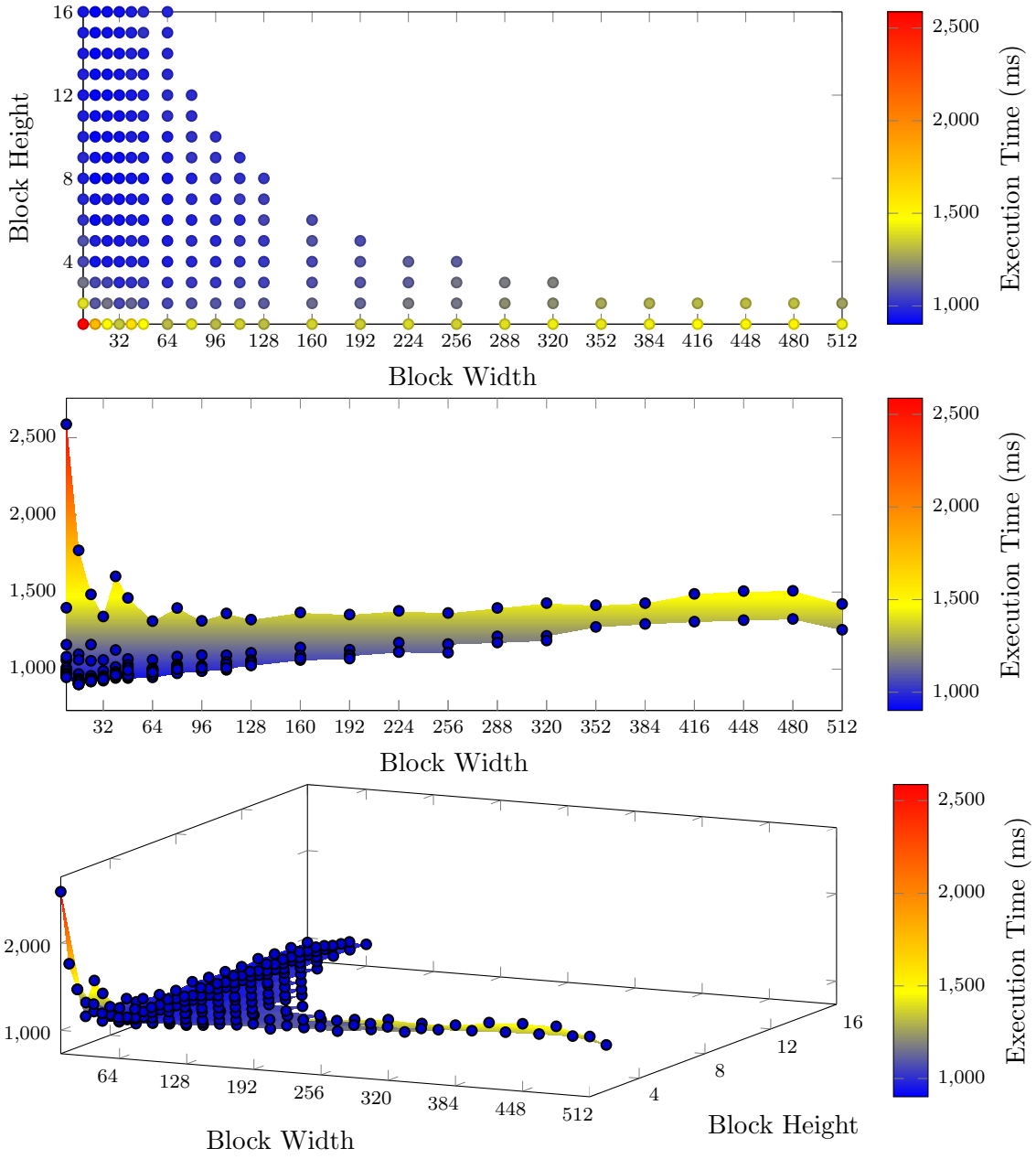


Figure 6.13: The comparison of the execution times of our PRW_{bal} with different 2D block sizes on the BSDS500 images, two alternative visualisations. The first version encodes the runtime for each block width–height pair within a colourbar scale. The second version interpolates these runtimes into a surface (shown from two different viewpoints). The best performance of < 910 ms is achieved with $16{\times}8, 16{\times}10, 16{\times}12, 16{\times}14, 16{\times}16$ block sizes.

We consider block widths between 8 and 512 and block heights between 1 and 16. Block sizes vary from $8 \times 1 = 8$ to $512 \times 2 = 1{,}024$ threads—the maximum block size possible. Hence, the number of blocks required to run the watershed on a 321×481 image can be, for instance, $41 \times 481 = 19{,}721$ or $1 \times 241 = 241$, depending on the block size.

The results in Figure 6.13 suggest that, for a fixed block width, larger block heights are generally preferred. For every block width considered the highest execution time is observed with block height 1, with the overall highest of 2,587.1578 ms seen at block size 8×1. Out of the 188 block sizes considered, an execution time of less than 940 ms is seen at only 30 with block widths between 16 and 32 and block heights between 6 and 16. Setting the threshold at 920 ms, the number of valid block sizes decreases to 7: block width is 16, block height is 6, 8, 10, 12, 14, 15, or 16. The best three configurations are 16×8, 16×14 and 16×16 with the runtimes between 901 ms and 904 ms. An additional observation is that the number of the landscape images, 481×321, is more than twice the number of portrait images, 321×481, in the dataset (348 vs. 152). This causes a small bias towards the smaller 16×8 and 16×14 block sizes for the specific dataset. However, the 16×16 block size still enables the best performance[3] in this experiment and should be preferred for the general case.

In addition to the block size tests, we also consider various values for RR when the block size is fixed at 16×16. The sums of the average path reduction step runtimes over the BSDS500 image set based on reduction rate values 1–60 are reported in Figure 6.14. The average number of global iterations, i.e. synchronisation points, for algorithm step S3 across the 500 images is also provided.

As can be seen from Figure 6.14 larger values of RR are preferable for this specific image set: the worst overall step S3 runtime of 132.5016 ms is observed when $RR = 1$

[3]The overall execution time of PRW_{bal} with block size 16×16 and reduction rate 5 in this experiment is 901.4569 ms. In the earlier experiments we reported 906.2504 ms for PRW_{bal} with the same parameter settings. This is normal variation between executions and it is below ±1% if individual execution times reported are averages of ten.

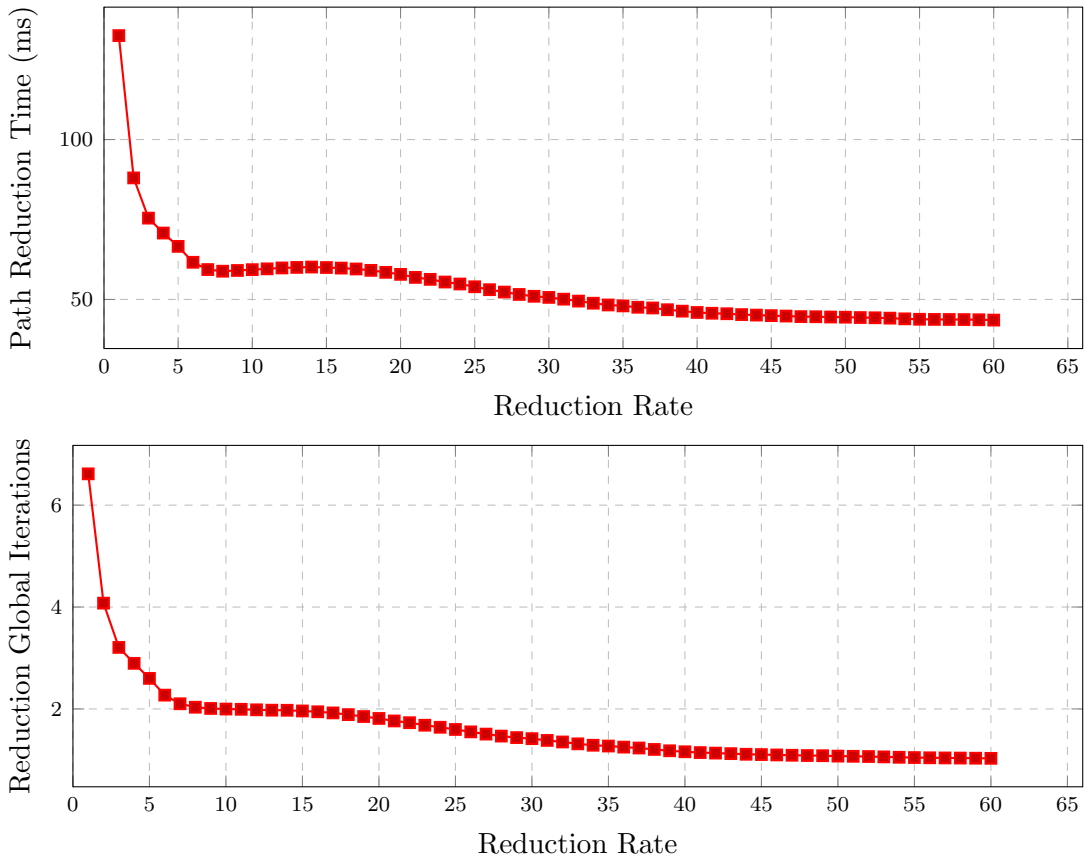


Figure 6.14: The comparison of the overall execution times (sum of average step S3 runtimes) and the number of global iterations of the path reduction (algorithm step S3) of PRW_{bal} for different reduction rate values on the BSDS500 images.

Table 6.1: The comparison of the execution times (in ms) of our PRW_{bal} with different 3D width×height×depth block sizes on the three 3D test images. The best performance for each image is highlighted.

width	height	depth	4,000×4,000×8	500×500×512	320×320×1,250
8	8	8	1,424.52	**1,597.34**	**2,138.56**
8	8	16	1,981.13	1,798.06	2,196.38
8	16	8	1,586.81	1,846.73	2,184.11
16	8	8	**1,360.30**	1,652.04	2,165.99
16	16	1	1,795.43	3,376.66	5,424.09
16	16	2	1,470.40	2,325.75	3,317.66
16	16	3	1,564.14	2,150.56	2,935.89
16	16	4	1,435.23	2,004.19	2,484.55

and the 43.5466 ms at $RR = 60$ is the best among the considered values. The runtimes correlate well with the number of global iterations. These results are explained by the catchment basins in the images in BSDS500 being generally very small with short label paths from catchment basin boundary pixels to catchment basin minima. For these images the label updating process is a lot cheaper than the global synchronisation, hence fewer global iterations means smaller step S3 runtime. Furthermore, for BSDS500 images the setting $RR = \infty$ works better with overall step S3 execution time at 42.9942 ms. This can be further decreased to 32.9034 ms if we replace the iterative global synchronisation with a single kernel call of linear path reduction. It is important to remember that this setting is optimal only for small images with little catchment basins and short label paths. If the label paths in an image are generally shorter than 50–60, then linear path reduction may be preferable. Otherwise, if the label paths are longer than around 60, logarithmic path reduction with an $RR \leq 10$ value should achieve a better runtime.

6.3.5 Large 3D images

In order to analyse the performance of PRW on typical large 3D medical images we consider a micro-CT image volume of total size 4,000×4,000×1,250 and construct three test subimages of 128 megavoxels each. A 4,000×4,000×8 image is made by taking the middle eight slices of the original image (slices 622–629); the middle 512 slices (370–881) at a reduced resolution of 500×500 form a second test image of size 500×500×512; reducing the intra-slice resolution even further to 320×320 for all slices of the original image results in the third test image of size 320×320×1,250. The overall number of voxels is exactly the same in all three test images—128,000,000.

The image slices in the three 3D test images are represented as 8-bit greyscale (0–255) images. This corresponds to the internal representation of this specific dataset in DICOM format. In case of other DICOM internal representations, pixel values can be linearly mapped onto the 8-bit greyscale range, with or without an intermediate windowing step. Alternatively, the input image value type of PRW can be changed to accommodate a wider range of distinct values, if the GPU memory limits allow that and the precision of the watershed result with reference to the value range is important.

First, we look at the PRW implementation parameters. For the fixed value of the reduction rate at 5, different configurations for the block size are considered. If the block width and the block height are fixed at 16 as per the best setting from our 2D experiments, the block depth will have to be at most 4 because of the block size limitation at 1,024 threads. Thus we also consider block sizes with two or three dimensions fixed at 8. PRW_{bal} results on the three test images for all the tested block sizes are reported in Table 6.1. The cube-shaped block size 8×8×8 holds the lowest runtimes for two out of the three test images. For the test image of size 4,000×4,000×8 a better runtime is seen with 16×8×8 blocks, which is explained by the shape of the test image with only few slices.

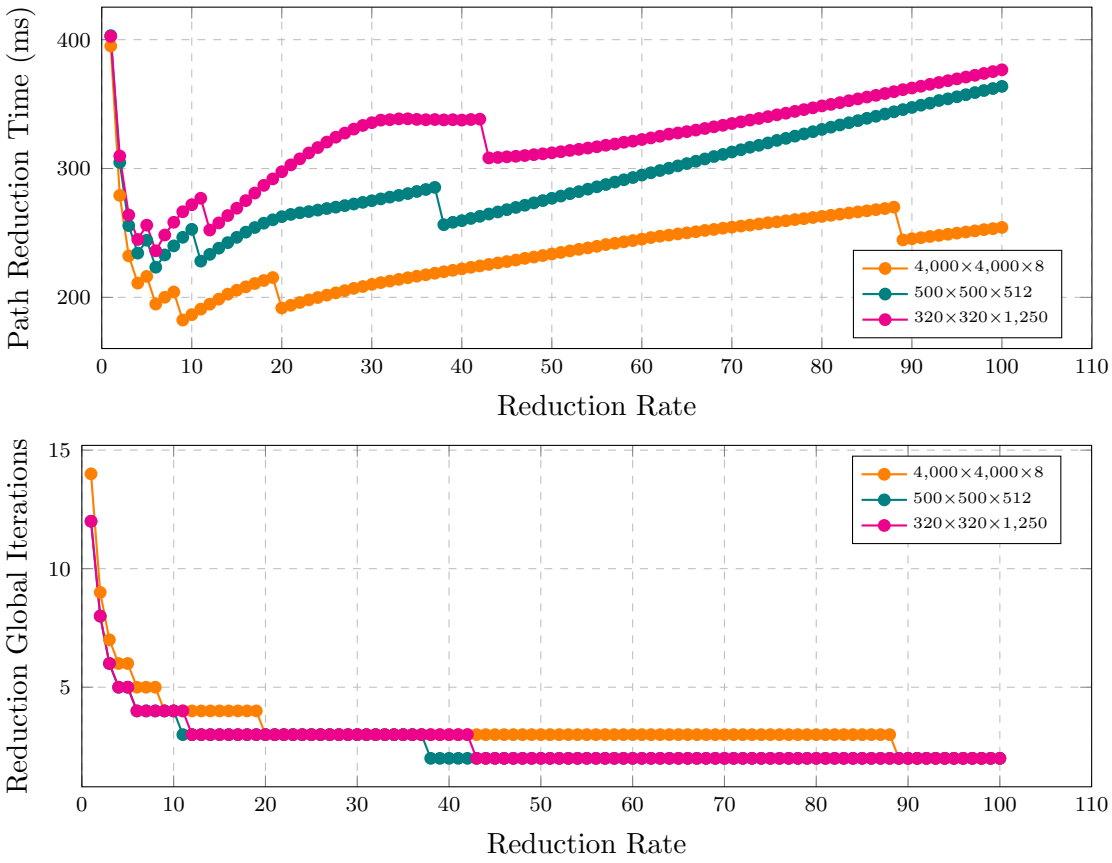


Figure 6.15: The comparison of the execution times (average step S3 runtimes) and the number of global iterations of the path reduction (algorithm step S3) of $\mathrm{PRW}_{\mathrm{bal}}$ for different reduction rate values on the 3D test images.

Table 6.2: The execution times (in ms, average of ten runs per image) of the three variants of our watershed compared on the 3D test images. The speedups of the block-asynchronous implementations over the synchronous procedure are reported in the last two rows.

	4,000×4,000×8	500×500×512	320×320×1,250
PRW_{sync}	2,306.06	3,054.32	4,827.83
$\text{PRW}_{\text{async}}$	1,226.01	1,463.46	1,912.05
PRW_{bal}	1,389.85	1,595.47	2,120.08
sync vs. async	1.881×	2.087×	2.525×
sync vs. bal	1.659×	1.914×	2.277×

In Figure 6.15 we examine the effect of various reduction rate values on PRW_{bal} step S3 runtimes and number of global iterations for the three test images with thread block size set at 8×8×8. Note, that the downshifts in all three plots of the step S3 runtimes occur at the exact same values of the reduction rate where the numbers of global iterations go down. Furthermore, when the number of global iterations is stable across an interval of RR values, the PRW_{bal} step S3 runtimes slowly increase with the RR value.

The best performance for the 500×500×512 and 320×320×1,250 test images is seen at $RR = 6$, and for the 4,000×4,000×8 image $RR = 9$ achieves the lowest runtime. We conclude that for large 3D medical images with potentially large watershed catchment basins a reduction rate in the range 5–10 is optimal. The choice of the best RR value is obviously data-specific. However, an RR value in the range 5–10 is a good guess: a difference of 20–40 ms in an execution of 1,500–2,000 ms is negligible.

Finally, the average execution times of all three variants of the path reducing watershed on the three 3D test images are reported in Table 6.2 with block size 8×8×8 and $RR = 6$. All nine runtimes are below 5 seconds (sec), and all six block-asynchronous runtimes (less than 2.2 sec) are faster than the three PRW_{sync} runtimes. $\text{PRW}_{\text{async}}$ is on average 2.164 times faster than PRW_{sync}. For PRW_{bal} the average speedup over PRW_{sync} stands at a factor of 1.95.

Figure 6.16 illustrates the composition of the PRW_{bal} execution times—data copy

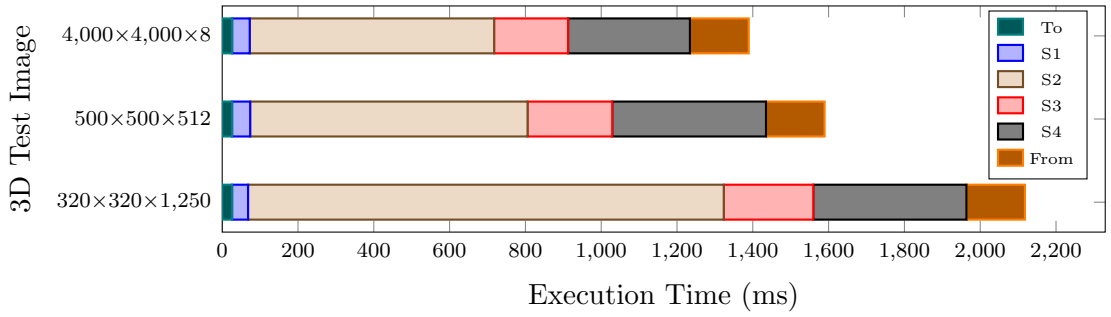


Figure 6.16: The composition of the PRW_{bal} execution times—data copy into the device, steps S1–S4, data copy from the device—on the 3D test images.

into the device, steps S1–S4, data copy from the device—on the 3D test images. The iterative steps with global synchronisation points, i.e. S2–S4, take the longest runtimes. Furthermore, the major variation in execution times across the three images is caused by the non-minimal plateau resolution step S2, reaching 1,255.0615 ms for the image 320×320×1,250. The numbers of global iterations are 7, 7, 13 for step S2; for step S3 they are 5, 4, 4; and 4, 4, 4—for step S4.

In Tables 6.1 and 6.2, and in Figure 6.15, there is a correlation between the number of slices in the test images and the PRW execution times: the larger the number of slices and the smaller the intra-slice resolution, the longer the PRW execution times. The main reason behind this trend is the weaker memory locality for the images with more slices. While adjacent voxels in the x-direction occupy adjacent memory positions and may be loaded from the global memory at the same time, the data for two adjacent voxels in the z-direction needs to be loaded in at least two copies.

6.4 Hierarchical segmentation with PRW

While the watershed transform is very popular in many applications, its main drawback of over-segmentation is inherent to this procedure. The examples in Section 6.3 clearly illustrate this trend. Hierarchical segmentation relies on post-processing of the watershed transform to solve the over-segmentation issue by merging non-significant

catchment basins. The waterfall transform [19, 84, 48] constructs the neighbourhood graph of the watershed catchment basins and applies the watershed transform on the minimal spanning tree of this graph to achieve a new coarser segmentation of the input image. Recursive application of the waterfall transform results in a hierarchical segmentation of the image with coarser layers of fewer segmented regions produced at each iteration.

6.4.1 Parallel waterfall transform with PRW

We suggest a novel parallel waterfall approach reminiscent of the original one [19] based on image reconstruction. After the primary watershed transform is applied on the input image, the image is modified by raising each of the catchment basins to a plateau the height of which equals the lowest pass point on the watershed line surrounding that catchment basin. In other words, the image modification comprises the two steps (kernels):

S5. identification of new minima

S6. update of image with new minima

Algorithm 7 provides the details of steps S5–S6 of the image reconstruction for the waterfall transform.

If I is still the set of pixel positions in the input image and $N(p)$ is the neighbourhood of the position $p \in I$, then for p and $q \in N(p)$ on the watershed line, i.e. $label(p) \neq label(q)$, we can define the watershed height as

$$height(p, q) = \max(image(p), image(q)).$$

Since our watershed does not explicitly assign any pixel as belonging to watershed lines and all pixels are put into catchment basins, we can speculate that the watershed

lines pass between pixels at the level of the higher pixels. When the pair p and q belong to different catchment basins, then $height(p, q)$ is the local height of the watershed line separating the catchment basins. For a given watershed catchment basin, its lowest pass point is the part of the surrounding watershed line with smallest height. This means that the height of the lowest pass point of a catchment basin equals the minimum among $height(p, q)$, where p belongs to this catchment basin and q does not. In the terrain map abstraction, if we flooded this catchment basin with water, then it would leak into a neighbouring catchment basin only after reaching the level, or the height, of the lowest pass point. Step S5 identifies this minimal level for each of the catchment basins by considering the whole watershed line. Before step S5 the array with the new minimum values ($newmin$) is initialised to a large constant M such that M is at least the upper bound of the greyscale range of the input image. For a 256-value greyscale range 0–255, we can set $M = 255$ or at any larger value.

Algorithm 7 Image reconstruction for parallel waterfall

```
 1: memset(newmin, M, |I|)                          ▷ Initialise new minima to a large constant

 2: for all p ∈ I in parallel do                    ▷ Step S5: identification of new minima
 3:     for all q ∈ N(p), label(q) ≠ label(p) do
 4:         height(p, q) ← max(image(p), image(q))
 5:         newmin(label(p)) ← min(newmin(label(p)), height(p, q))
 6:     end for
 7: end for

 8: for all p ∈ I in parallel do                    ▷ Step S6: update of image with new minima
 9:     if image(p) < newmin(label(p)) then
10:         image(p) ← newmin(label(p))
11:     end if
12: end for
```

Step S6 updates the input image based on the new minimum values. Every pixel below the new minimum for its catchment basin is raised to the level of that new minimum, i.e. $image(p) = newmin(label(p))$ if $image(p)$ was smaller than $newmin(label(p))$ before step S6.

Figure 6.17 illustrates the image reconstruction process for the waterfall transform. It depicts the watershed transform of an input 1D image followed by three recursive

applications of waterfall—the image reconstruction and the watershed transform. The image boundaries and watershed lines are red. The horizontal lines correspond to the new minimum values for each of the old watershed regions. Their colours indicate the new larger watershed regions that they get grouped into in the waterfall applications.

If the watershed transform is applied on the reconstructed image, the result will be a set of new catchment basins, where each corresponds to several catchment basins in the primary watershed transform. The image reconstruction and the following watershed transform comprise the waterfall. If applied recursively, this transform gradually reduces the number of catchment basins, eventually grouping the whole image into a single region, see Figure 6.17.

Our implementation of the waterfall relies on PRW_{bal} for the watershed steps S1–S4. The image reconstruction steps S5–S6 are implemented with a single kernel each. Each pixel position in the image is still assigned to a separate execution thread; the parameters are set as per the PRW_{bal} implementation. The host code of the procedure that applies watershed followed by a fixed number of waterfall applications is presented in Algorithm 8.

After the image is copied into the device and the thread blocks are set up, the hierarchical segmentation is constructed layer by layer. The layer generation starts with a watershed run (lines 4–14 in Algorithm 8). For all layers except the last one—if $1 \leq n < NL$—steps S5–S6 follow (lines 15–19 in Algorithm 8) to prepare for the next watershed application by reconstructing the image with new minima levels. Finally, the new layer of the segmentation is copied from the device to the host memory (line 20).

The memory requirements of the hierarchical segmentation procedure are the same as for PRW_{bal}: new variables or arrays are *not* used. This is achieved by reusing the available memory: the same $image, label, state, newstate$ arrays are used at every application of the watershed. This is allowed because: (a) $label, state, newstate$ are

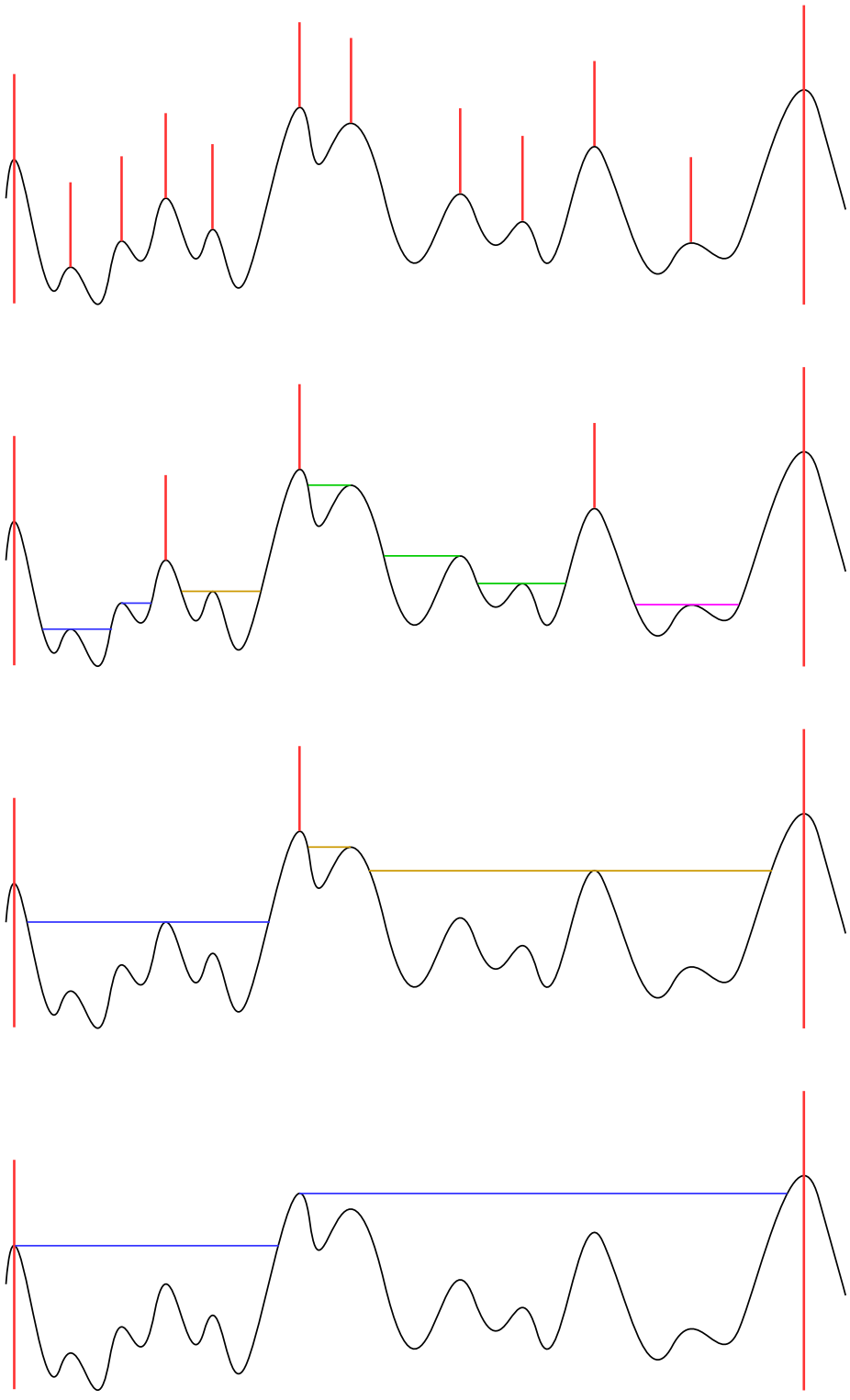


Figure 6.17: The watershed transform of an input 1D image followed by three recursive applications of waterfall—the image reconstruction and the watershed transform. The image boundaries and watershed lines are red.

Algorithm 8 The host code of the hierarchical segmentation

```
1: HostToDevice(image)                          ▷ Copy image from host to device
2: DimGrid, DimBlock                            ▷ Set grid and block dimensions
3: for n ← 1, NL do
4:     InitialisationKernel                     ▷ Step S1
5:     repeat                                   ▷ Step S2
6:         ResolutionKernel
7:         swap(state, newstate)
8:     until ¬change
9:     repeat                                   ▷ Step S3
10:        ReductionKernel
11:    until ¬change
12:    repeat                                   ▷ Step S4
13:        MergingKernel
14:    until ¬change
15:    if n < NL then
16:        memset(newmin, M, |I|)
17:        IdentificationKernel                 ▷ Step S5
18:        UpdateKernel                         ▷ Step S6
19:    end if
20:    DeviceToHost(label)                      ▷ Copy labels from device to host
21: end for
```

initialised from scratch during steps S1–S2 and (b) *label* values are copied back into a new address in the host memory at each iteration. Furthermore, the *newstate* array is reused under the name *newmin* in the image reconstruction steps S5–S6 thus eliminating the need for new memory.

The only new parameter that the hierarchical segmentation adds is the number of the waterfall applications. We denote the number of watershed applications (primary or as part of waterfall) NL, indicating the number of segmentation layers generated in the hierarchical segmentation. But this parameter does not affect the execution of the individual waterfall applications and just defines their number. For a typical input image, a hierarchy leading to a single region corresponding to the whole image is usually achieved in under ten waterfall applications. A setting of $NL = 5$ to $NL = 7$ is usually enough to get a semantically meaningful hierarchy of segmentations. This estimate is based empirically on multiple previous experiments in our research group on the application of sequential watershed/waterfall to various data; our results in the next subsection agree with this.

Table 6.3: The numbers of catchment basins in different layers of the hierarchical segmentation for a few 2D test images. Layer 7 is only required for one image out of six.

layer	Lena	CT	MRI	Image 1	Image 250	Image 500
1	21088	8665	19540	21671	15094	10258
2	3945	1872	3536	4704	3078	1176
3	649	401	653	1062	621	166
4	110	82	122	236	113	28
5	21	20	22	55	20	4
6	6	7	5	13	5	1
7	1	1	1	**2**	1	1

In terms of dimensionality, this hierarchical segmentation mimics the watershed transform used—$\mathrm{PRW}_{\mathrm{bal}}$, in this case. While a 3D implementation would work for 1D or 2D images, the memory consumption pattern of $\mathrm{PRW}_{\mathrm{bal}}$ is optimised in case of separate 2D and 3D implementations.

6.4.2 Parallel waterfall results

Figure 6.18 illustrates how our hierarchical segmentation procedure works on three sample 2D images for $NL = 6$: Lena, the abdominal CT slice and the knee MRI slice. Again, the watershed and the following waterfall transforms are applied on the gradient magnitude image of the minimally smoothed original input image. Six layers of the hierarchical segmentation are presented—the watershed and five subsequent waterfalls. The waterfall layers (2–6) clearly demonstrate how smaller regions get gradually grouped together in higher layers of the segmentation. In addition, remaining watershed lines always trace high-contrast lines in the image, thus following actual boundaries of objects. For instance, tibia and femur are nicely segmented into single regions in layers 5 and 6, respectively, of the MRI slice segmentation.

In case of all three images considered, layer 7 (not shown) corresponds to the whole image: the segmentation contains only a single region. Further details about the numbers of regions in different layers of the hierarchical segmentation for the

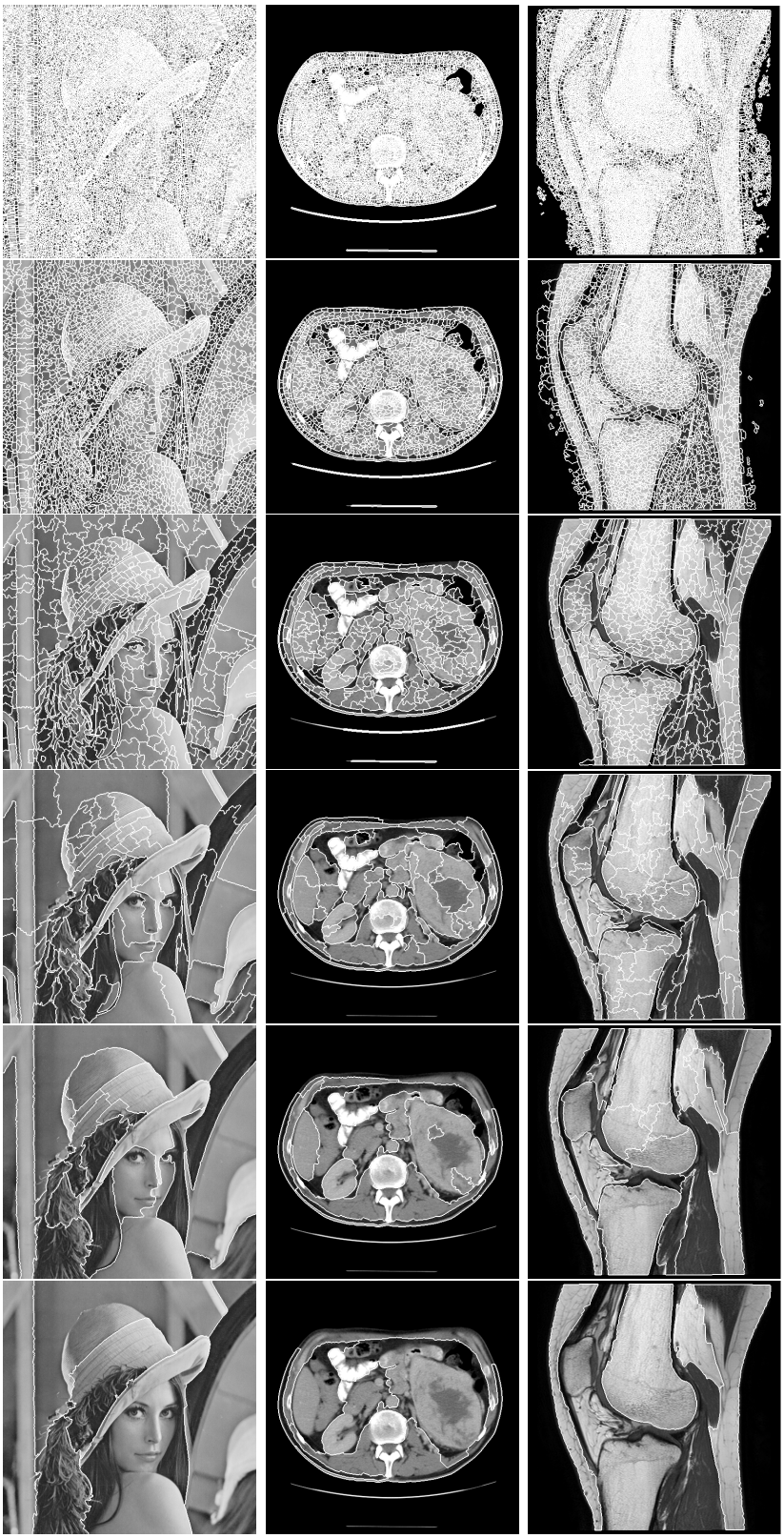


Figure 6.18: Hierarchical segmentation of (gradient magnitude of minimally smoothed) sample images Lena, abdominal CT slice, and knee MRI slice for $NL = 6$.

considered 2D images are presented in Table 6.3. For five images out of six, the whole image is merged into a single region in at most seven segmentation layers. The seventh layer is only useful for one—image 1 from BSDS500—for which the final merge takes place at layer 8.

Further qualitative results are presented for images 1, 250, and 500 from the BSDS500 set in Figure 6.19. Note, that this time we run the watershed/waterfall transforms on the gradient magnitude image for these examples unlike the results in Section 6.3. The segmentation gives poorer results for two of these images (namely, image 250 and 500) since: (a) the images are quite low-contrast with blending object boundaries and (b) even the minimal smoothing applied on such low resolution images has a negative impact. Skipping the smoothing pre-processing step or replacing the Gaussian blur with a more elaborate technique, like anisotropic diffusion filtering [95, 48], may be options to improve performance.

One important drawback of the suggested algorithm is the dislocation of watershed lines in plateaux between layers. Sequential waterfall transforms based on neighbourhood graphs [84, 48] function on graphs where watershed regions are represented as nodes which get merged together. When the waterfall result is projected back onto the image, some of the watershed lines are dismissed while other stay intact. Thus the waterfall transform just erases some line segments.

In our case, the transform is applied directly to the image. Non-minimal plateaux may change shape and new plateaux may appear after image reconstruction steps S5–S6. Our waterfall relies on an independent balanced watershed application with fair division of the newly formed plateaux. As a result, the outcome of our waterfall is the erasure of some watershed lines with possible dislocation of others. For example, if the result of the primary watershed is

$$104 \leftarrow 107 \qquad 105 \leftarrow 109 \qquad 106 \rightarrow 106 \qquad 109 \rightarrow 105 \qquad 107 \rightarrow 104,$$

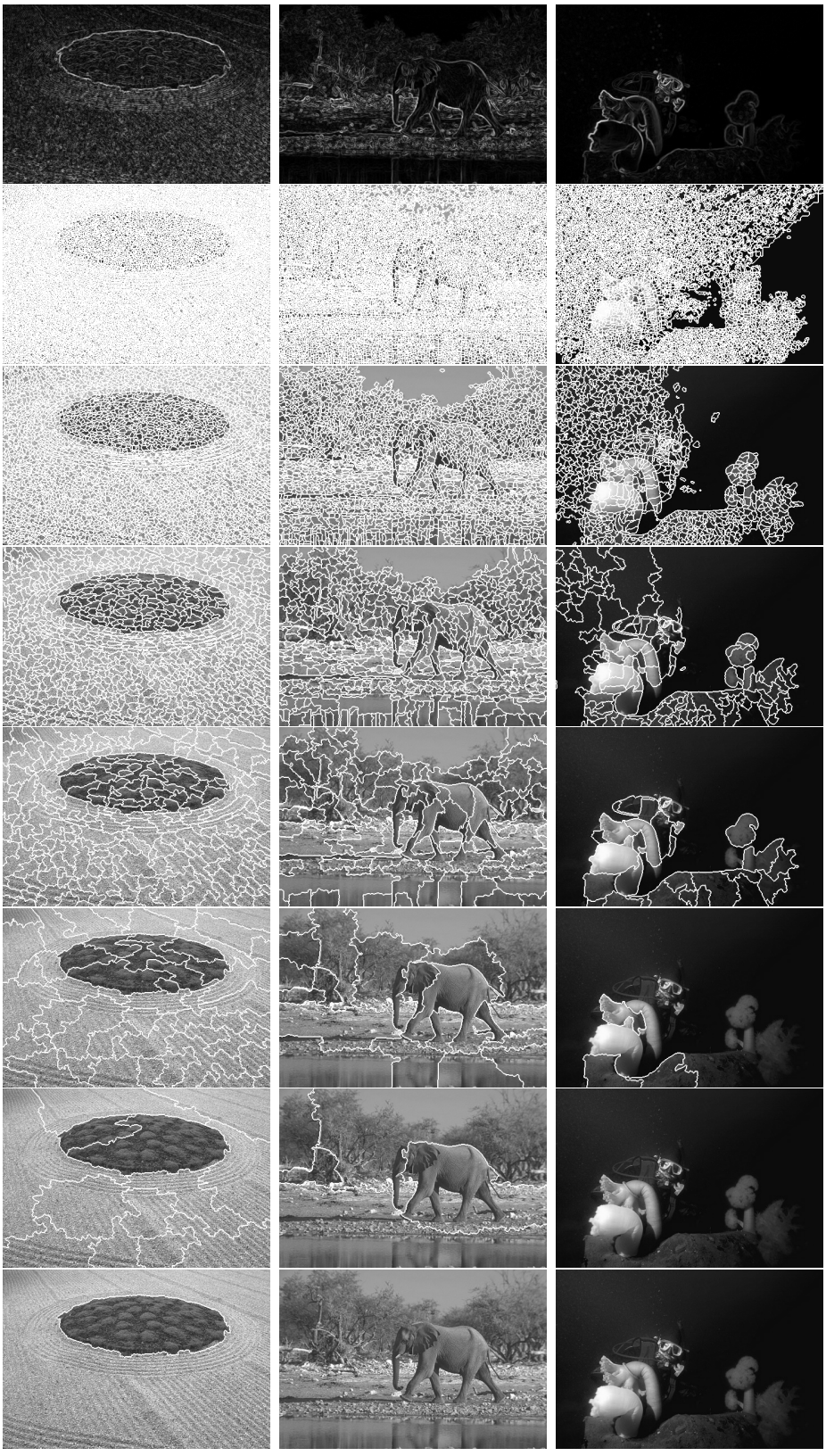


Figure 6.19: The gradient magnitude of minimally smoothed images 1, 250 and 500 from BSDS500 followed by hierarchical segmentation with $NL = 7$.

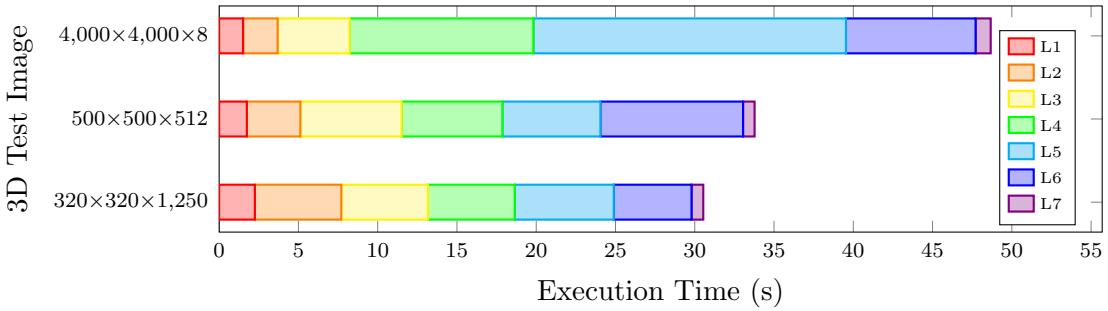


Figure 6.20: The layer generation execution times (lines 4–20 in Algorithm 8) for $NL = 7$ iterations of the hierarchical segmentation on the 3D test images.

where numbers are the pixel greyscale values and arrows are the labels, then the reconstructed image and the following watershed will look like

$$107 \leftarrow 107 \leftarrow 107 \leftarrow 109 \leftarrow 109 \qquad 109 \rightarrow 109 \rightarrow 107 \rightarrow 107 \rightarrow 107.$$

The sequential graph based methods would make two regions, grouping the original catchment basins in a 3+2 or 2+3 manner:

$$107 \leftarrow 107 \leftarrow 107 \leftarrow 109 \leftarrow 109 \leftarrow 109 \qquad 109 \rightarrow 107 \rightarrow 107 \rightarrow 107;$$

$$107 \leftarrow 107 \leftarrow 107 \leftarrow 109 \qquad 109 \rightarrow 109 \rightarrow 109 \rightarrow 107 \rightarrow 107 \rightarrow 107.$$

This limits the usability of our waterfall transform for the construction of hierarchical segmentation data structures, like IPFs [43], without additional processing. With the dislocated watershed lines, it is not always possible to establish a parent–child relationship based on inclusion between regions in subsequent layers. However, our approach successfully solves the over-segmentation issue of watershed and is readily usable for fast object segmentation in large images.

Finally, we look at the execution times of our waterfall and hierarchical segmentation using the three 3D micro-CT test images from Section 6.3. Figure 6.20 depicts the comparison of the overall layer generation runtimes across the three images with

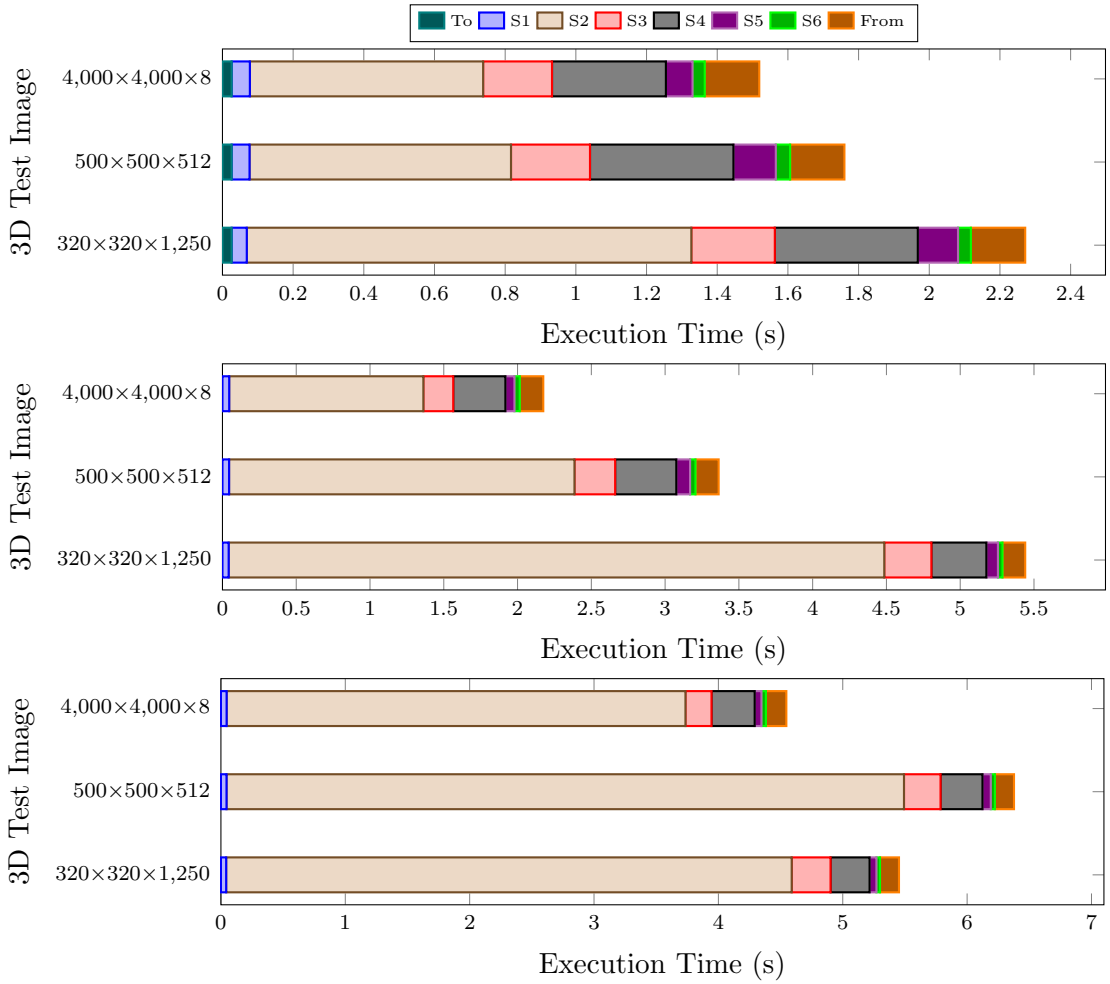


Figure 6.21: The composition of layer generation execution times (lines 4–20 in Algorithm 8) for segmentation layers 1–3 of the hierarchical segmentation with $NL = 7$ on the 3D test images. Note, scale of values is different in each plot.

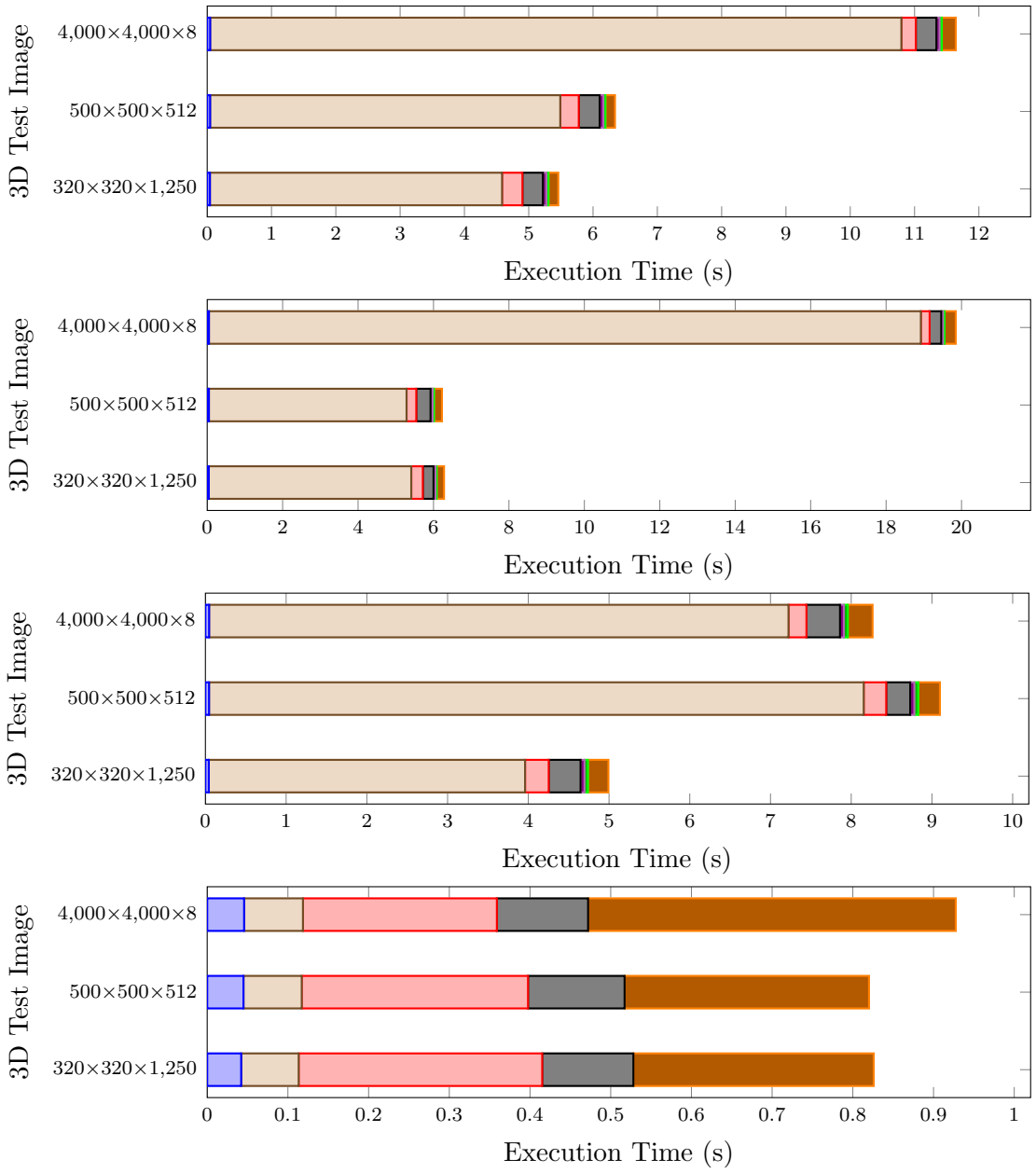


Figure 6.22: The composition of layer generation execution times (lines 4–20 in Algorithm 8) for segmentation layers 4–7 of the hierarchical segmentation with $NL = 7$ on the 3D test images. Note, scale of values is different in each plot.

$NL = 7$. Each measured layer generation time includes watershed with steps S1–S4, image reconstruction with steps S5–S6 (except layer 7), and transfer of watershed labels from device to host. A further detailed breakdown of runtimes for individual steps at each layer is presented in Figures 6.21 and 6.22.

The layer generation times in Figure 6.20 suggest that the runtimes tend to increase with layer number and drop at the end. This trend is especially noticeable for the 4,000×4,000×8 image and can be explained by large intra-slice non-minimal plateaux appearing after the image reconstruction steps. Since this image has the largest dimension length difference among the three images, the phenomenon is observed for that specific volume. Further consideration of the plots in Figures 6.21 and 6.22 confirms that for a fixed layer, the main difference in the runtime among the three images occurs in the length of step S2. For every other step (S1, S3–S6, or device-to-host) the difference between the largest value and the smallest value across all three images and all seven layers (21 values for watershed steps and label copy; 18 values for image reconstruction) is below 308 ms. For step S2 the runtime ranges from around 71 ms to almost 18.9 sec.

For all three test images, the whole image is merged into a single region at layer 6. This explains the quick drop in execution time at layer 7: the image reconstruction steps S5–S6 of the sixth layer generation convert the image into a uniform set of maximal greyscale values. Thus, for the given images the last two layers are unnecessary. If we remove those, we get overall execution times of less than 40 sec for 4,000×4,000×8 and less than 25 sec for 500×500×512 and 320×320×1,250.

6.5 Summary

We introduced and discussed three variants (synchronous, block-asynchronous, and balanced block-asynchronous) of a new parallel watershed transform for execution on

the GPU. Our approach relies on the construction of paths of steepest descent and their logarithmic reduction into pointers to catchment basin minima. The procedure also successfully incorporates the identification and resolution of plateaux, thus no preprocessing is required for that.

We considered various parameter settings to reveal execution time patterns; compared the three variants of the path reducing watershed on a large number of 2D and 3D images to find runtime ranges for fixed size images of different construction. Tests with CUDA implementations proved our PRW algorithm superior to existing parallel watershed algorithms for the GPU, showing a runtime improvement over the state-of-the-art of around 30%. For 3D images of 128 megavoxels we achieved execution times of approximately 1.5–2 sec per image volume, depending on the image dimensions.

An important limit to the performance of our watershed and similar ones [63, 98] is the size of the memory available on our rather modest hardware specifications: the whole image needs to be stored in the GPU memory. For 3D images significantly larger than the ones tested, an alternative approach of streamed watershed [54] may be more appropriate on consumer level GPUs. More powerful GPU devices targeted for high-performance computing, like the NVIDIA Tesla range, can be used for images of thousands of megavoxels.

We also presented a parallel waterfall transform consisting of an image reconstruction step followed by the watershed. Qualitative experimental results proved the method applicable for solving the over-segmentation issue of watershed. Results on large 3D images showed that a full hierarchical segmentation can be performed on 128 megavoxel images in 25–40 sec. The main disadvantage of displaced watershed lines of the suggested parallel waterfall limits its immediate applicability on IPF construction. The search for a lightweight fix is an interesting potential topic for further research.

Chapter 7

Fast Dashing Method for the GPU

7.1 Introduction

Given all the knowledge from Chapters 3, 4, and 6, we can now set the aim to construct a novel parallel procedure that incorporates information about the image partitioning into the front propagation. As was shown in the previous chapter, partitioning an image into multiple small homogeneous regions can be made into a fairly inexpensive task.

This chapter deals with partitioning-based front propagation whose main objective is the increased local visibility. While using the simplest neighbourhood stencils, such as the 6-voxel von Neumann neighbourhood in 3D, the propagation rules can use the partitioning labels to conduct faster and easier front advancement inside partitioning regions. At the same time, propagation across region boundaries can be slowed down. This way, the global front will essentially propagate region by region, rather than point by point.

We believe that a partitioning-based front propagation algorithm will improve upon the performance of traditional eikonal solvers for the task of medical image segmentation. The novel approach will reduce the oversensitivity of the eikonal solvers

to small image heterogeneities. It will also result in a better location of object boundaries, thus improving the segmentation quality.

7.2 Fast dashing method

7.2.1 Fast dashing parallel method

Our algorithm is called fast dashing method (FDM); it works in parallel and consists of two main steps:

1. initialisation of arrival times and states

2. propagation of arrival times using active state pixel positions

This is a hybrid algorithm combining classic solution of eikonal equation for front propagation with watershed (or any other) grouping of pixel positions. The algorithm expects a non-empty set of initial pixel positions—seeds—from which the front development starts. The actual propagation of the front follows; it is modelled with iteratively improved approximations of arrival times for the whole image. In contrast with traditional front propagation, where the front moves with a locally determined speed at each point, our front 'dashes' through watershed regions with infinite speed. Thus the arrival time is uniform inside watershed regions and increases when the front crosses the watershed lines. Algorithm 9 gives a description of the FDM.

The first step initialises arrival times at all pixel positions in the image based on the set of seed positions. Afterwards, the second step propagates true arrival times by iteratively improving arrival times across the image. Instead of re-calculating arrival times at each position at each iteration, the algorithm relies on a state array which differentiates between active state positions to be updated and passive state positions which skip the iteration.

Algorithm 9 Fast dashing parallel method

1: **for all** $p \in I$ in parallel **do** ▷ Initialisation of arrival times and states
2: **if** $seed_{global}(p)$ **then** ▷ implementation re-uses $state_{global}$ for $seed$
3: **for all** $q \in N(p)$ **do**
4: $state_{global}(q) \leftarrow true$
5: **end for**
6: $time_{global}(p) \leftarrow 0$
7: **else**
8: $time_{global}(p) \leftarrow \infty$
9: **end if**
10: **end for**

11: **repeat** ▷ Propagation of arrival times
12: **for all** $p \in I$ in parallel **do**
13: **if** $state_{global}(p)$ **then**
14: $state_{global}(p) \leftarrow false$
15: $time_{local}(p) \leftarrow time_{global}(p)$
16: **for all** $q \in N(p)$ **do**
17: **if** $time_{global}(q) < time_{local}(p)$ **and** $label_{global}(q) = label_{global}(p)$ **then**
18: $time_{local}(p) \leftarrow time_{global}(q)$ ▷ 'dash' inside watershed region
19: **end if**
20: **end for**
21: $time_{local}(p) \leftarrow \min(time_{local}(p), SolveEikonal(p))$ ▷ solve eikonal equation
22: **if** $time_{local}(p) < time_{global}(p)$ **then**
23: $time_{global}(p) \leftarrow time_{local}(p)$
24: $ThreadFence$ ▷ ensure write order
25: **for all** $q \in N(p), time_{global}(q) > time_{local}(p)$ **do**
26: $state_{global}(q) \leftarrow true$
27: **end for**
28: **end if**
29: **end if**
30: **end for**
31: **until** $\neg change$ ▷ $change$ is $true$ while there are active states

The initialisation step (lines 1–10 in Algorithm 9) considers all the pixel positions $p \in I$ in the image in parallel and initialises arrival time and state arrays. If a point p belongs to the set of seeds, then *time* is set to 0; otherwise, $time = \infty$. After initialisation, the arrival times can be updated only at positions neighbouring with seeds. Thus, during the initialisation step, for every seed point the state of all their neighbours is set to active (line 4 in Algorithm 9). Otherwise, the state stays passive for other points until the front comes near them.

During the propagation step, at each iteration only active state points are processed. First, the state is reset to passive, then the arrival time is updated using arrival times at neighbouring points as follows. If there exists a neighbour with a smaller arrival time such that the two positions are in the same watershed (or, generally, any other partitioning) region, i.e. $\exists q \in N(p), time(q) < time(p) \wedge label(q) = label(p)$, then the current arrival time at p is replaced by that neighbour's arrival time. In addition, a solution of the eikonal equation is attempted and, if it is better, then the arrival time is replaced by that solution (line 21 in Algorithm 9). If the result after the considered updates at position p improves the initial time, then a further iteration may improve the arrival time values for points in the neighbourhood $N(p)$. Hence, for every $q \in N(p)$, its state is set as active if $time(q) > time(p)$.

Algorithm 10 *SolveEikonal*—solve eikonal equation at point p in 3D

1: $a \leftarrow \min\{time(p \pm 1)\}$ ▷ minimum of neighbours in 1st dimension
2: $b \leftarrow \min\{time(p \pm widthI)\}$ ▷ minimum of neighbours in 2nd dimension
3: $c \leftarrow \min\{time(p \pm widthI \times heightI)\}$ ▷ minimum of neighbours in 3rd dimension
4: $Sort\{a, b, c\}$ ▷ $c \leq b \leq a$
5: $u \leftarrow c + \frac{1}{f(p)}$
6: **if** $u \leq b$ **then**
7: **return** u ▷ update based on one neighbour
8: **end if**
9: $u \leftarrow \frac{b+c+\sqrt{\frac{2}{f(p)^2}-(b-c)^2}}{2}$
10: **if** $u \leq a$ **then**
11: **return** u ▷ update based on two neighbours
12: **end if**
13: $u \leftarrow \frac{a+b+c+\sqrt{\frac{3}{f(p)^2}-(a-b)^2-(a-c)^2-(b-c)^2}}{3}$
14: **return** u ▷ update based on all three neighbours

Algorithm 10 provides the details of the solution of the eikonal equation, see also [56, 16, 108]. This solution is based on the discretisation of the eikonal equation with first-order upwind finite difference approximation scheme, where the distance between voxels in each dimension is taken to be unit ($\Delta x = \Delta y = \Delta z = 1$). The discretisation arrives at a quadratic equation where the unknown is the arrival time value at the currently considered point p. In each dimension, the smaller of the two arrival time values of neighbours is stored in a, b, or c, respectively. Afterwards, these values are sorted in non-increasing order, i.e. $a \geq b \geq c$, and the solution of the quadratic equation is constructed.

The update iteration (lines 12–30 in Algorithm 9) of the propagation step is repeatedly applied until no further changes are made. The flag *change* is maintained throughout the process for that check. It is reset to *false* before each iteration and updated to *true* during the iteration as soon as the state of any point becomes active (line 26). Once the flag is *false* after an iteration, then no further arrival time improvements can be made in the image, and the execution is terminated.

The number of the parallel update iterations is limited by the number of positions in the grid since at least a single position will get its final arrival time approximation at each iteration. This means that the overall parallel complexity of the algorithm is $O(N)$ in case of a massively parallel implementation with $O(N)$ execution threads, where N is the number of positions in the discrete grid.

Further implementation details of the fast dashing procedure described in Algorithm 9 are presented below.

7.2.2 GPU synchronous implementation

The fast dashing method can be implemented for GPU execution in two different ways: dimensionality independent synchronous implementation and block-asynchronous implementation optimised for specific image dimensionality. The details of the syn-

chronous implementation that can be executed for both 2D and 3D images are presented below.

The synchronous implementation, FDM_{sync}, is fully based on the procedure presented in Algorithm 9; the host code of the procedure is described in Algorithm 11. The device code consists of 2 kernels—one for each of the initialisation (lines 2–9 in Algorithm 9) and propagation (lines 13–29) steps—and a device function, as per Algorithm 10, for the solution of the eikonal equation. The implementation relies on the von Neumann neighbourhood of 6 points in 3D.

Algorithm 11 The host code of the FDM algorithm

```
1: HostToDevice(image)                    ▷ Copy image from host to device
2: HostToDevice(label)                    ▷ Copy labels from host to device
3: HostToDevice(seed)                     ▷ Copy seeds from host to device
4: DimGrid, DimBlock                      ▷ Set grid and block dimensions
5: InitialisationKernel                   ▷ Initialisation
6: repeat                                 ▷ Propagation
7:     PropagationKernel
8: until ¬change
9: DeviceToHost(time)                     ▷ Copy times from device to host
```

The host code first copies the input image, the watershed partitioning labels and the set of seeds into the global device memory. Next, the size and the number of blocks are set up, the latter based on the image size. The initialisation kernel is executed once, followed by iterative application of the propagation kernel. And, finally, the results are copied into the host memory in the form of arrival time values. Both kernels are executed with a separate thread used for each pixel position. Thus the thread corresponding to point p may update $time_{global}(p)$ and $state_{global}(q)$ in case of both kernels, where $q \in N(p)$.

In order to decrease the memory consumption of the algorithm, our implementation re-uses the $state_{global}$ (of type `char`) array for $seed_{global}$. On line 3 of Algorithm 11 the *seed* array from the host memory is, in fact, copied into the $state_{global}$ array in the device memory. Value $state_{global} = 1$ denotes a seed; $state_{global} = 0$ denotes a non-seed. In addition, lines 2–5 of Algorithm 9 in the implementation are replaced

with

```
...
if odd(state_global(p)) then                    ▷ implementation re-uses state_global for seed
    state_global(p) ← false
    for all q ∈ N(p) do
        state_global(q) ← state_global(q) + 2
    end for
    ...
end if
...
```

The CUDA execution model does not assume fully parallel execution of threads: the execution of one thread can start after the completion of another thread. Hence, there is a need to differentiate the points with $state_{global}(p) = true$ into two groups: the seeds with odd $state_{global}(p)$, and neighbours of seeds with even $state_{global}(p)$. If p is a seed, then at the beginning of its initialisation thread execution $state_{global}(p) = 1 + 2k$, where $k \geq 0$ seed neighbours have already updated the state of p. If, on the contrary, p is *not* a seed, then $state_{global}(p) = 2k$. Thus, only seeds will pass the state parity check and their states will be set to passive for the following propagation. The neighbours of seeds will fail the state parity check and their states will remain non-zero, hence active, for the following propagation. Once the initialisation step is complete, all the state values can be interpreted as boolean *true–false* values denoting active–passive states. Note, that a seed can still be in active state after initialisation if a seed neighbour of it is initialised later than it.

The CUDA memory model is weakly-ordered: writes into memory by a thread are not necessarily observed in the same order by other threads. When (the thread of) a point p activates the state of its neighbour q (line 26 in Algorithm 9), p's arrival time is expected to have just been updated, so that the neighbour can update its own arrival time based on it. Otherwise, if $state_{global}(q) = true$ is set by p but the updated $time_{global}(p)$ is not available to q to read, then the propagation iteration at q may make a vain attempt at updating $time_{global}(q)$ and change q to passive state. The $ThreadFence$ on line 24 of Algorithm 9 ensures that the updated $time_{global}(p)$

is visible to q before $state_{global}(q)$ becomes active. Both state and time global array variables are declared with the `volatile` keyword to disable access optimisation for these variables and enforce actual reads and writes to this memory whenever state and time global variables are referenced.

In Algorithm 9 and in a few following procedure descriptions the scope of variables is highlighted with a subscript. The synchronous implementation of the fast dashing method always relies on the global arrays for state, label, and image (used to calculate propagation speed) variables. In the case of the arrival time variables, the propagation kernel makes a local copy of the time for active state points and updates it using the global time variables of the neighbours. If, after the attempted updates, the local arrival time is better than the corresponding global variable, then the new value is copied into the global memory. This manipulation with a local arrival time copy enables the check for arrival time improvement and reduces the number of expensive global memory accesses.

7.2.3 Block-asynchronous implementation

The GPU block-asynchronous implementation of the fast dashing method, $\text{FDM}_{\text{async}}$, is aimed at improving the execution time of FDM by reducing the number of global synchronisation points in the propagation step. This is achieved by exploiting the block-level synchronisation mechanisms of CUDA and using block-level shared memory. Algorithm 12 presents the propagation kernel of the $\text{FDM}_{\text{async}}$ implementation. The host code, the initialisation kernel, and the eikonal solver are unchanged from FDM_{sync}.

The propagation kernel of $\text{FDM}_{\text{async}}$ first copies the data from the global memory into the shared memory. Then follow arrival time update iterations for active state pixel positions, until convergence inside the block. Finally, the data is copied from the shared memory into the global memory.

Algorithm 12 Arrival time propagation kernel of $\text{FDM}_{\text{async}}$

1: $GlobalToShared(image)$ ▷ Copy image from global to shared memory
2: $GlobalToShared(label)$ ▷ Copy label from global to shared memory
3: $GlobalToShared(state)$ ▷ Copy state from global to shared memory
4: **if** $state_{shared}(p)$ **then**
5: $state_{global}(p) \leftarrow false$
6: **end if**
7: $GlobalToShared(time)$ ▷ Copy time from global to shared memory
8: $time_{local}(p) \leftarrow time_{shared}(p)$ ▷ Copy time from shared to local memory
9: **repeat**
10: $SyncThreads$ ▷ Synchronise threads in block
11: **if** $state_{shared}(p)$ **then**
12: $state_{shared}(p) \leftarrow false$
13: **for all** $q \in N(p)$ **do**
14: **if** $time_{shared}(q) < time_{local}(p)$ **and** $label_{shared}(q) = label_{shared}(p)$ **then**
15: $time_{local}(p) \leftarrow time_{shared}(q)$ ▷ 'dash' inside watershed region
16: **end if**
17: **end for**
18: $time_{local}(p) \leftarrow \min(time_{local}(p), SolveEikonal(p))$ ▷ solve eikonal equation
19: **if** $time_{local}(p) < time_{shared}(p)$ **then**
20: $time_{shared}(p) \leftarrow time_{local}(p)$
21: $ThreadFenceBlock$ ▷ Ensure write order
22: **for all** $q \in N(p), time_{shared}(q) > time_{local}(p)$ **do**
23: $state_{shared}(q) \leftarrow true$
24: **end for**
25: **end if**
26: **end if**
27: $SyncThreads$ ▷ Synchronise threads in block
28: **until** $\neg blockchange$
29: $SharedToGlobal(time)$ ▷ Copy updated time from shared to global memory
30: $ThreadFence$ ▷ Ensure write order
31: **for all** $q \in N(p), \neg InBlock(q) \wedge state_{shared}(q)$ **do**
32: $state_{global}(q) \leftarrow true$
33: **end for**

Lines 1–7 of Algorithm 12 briefly describe what data is copied into the shared memory by the propagation kernel of $\text{FDM}_{\text{async}}$. Each thread is responsible for copying the global image, label, state, and time values for the single corresponding pixel position p into the shared memory. If the state is active for that point, then its arrival time will be updated later by this thread, so the global state variable can be reset to passive. In addition, the threads on the boundary of the block are supposed to copy global image, label, and time values into the shared memory for their neighbours outside of the block. The state values of the out-of-the-block neighbours are not copied from the global memory; they are just initialised as passive by the corresponding boundary threads. The arrival time values for any active state out-of-the-block neighbours will be updated in their own blocks, thus information about their initial state is redundant in the currently considered block. At the same time, if the execution of the current block updates any arrival times on the boundary of the block, the states of their out-of-the-block neighbours should be changed to active. Thus, the extra band of state values is still required in the shared memory, and their initialisation to passive allows to recognise any later state change by the current block.

Following the initialisation of the shared memory, a local time variable gets its value from the corresponding shared variable (line 8 of Algorithm 12). Actual time update iterations, presented in lines 11–26 of Algorithm 12, are repeated until no active state points remain within the block. The *blockchange* variable is set to *false* before each iteration and becomes *true* when any state is activated during the iteration. The iteration process mimics that of the FDM_{sync} procedure at the level of shared variables inside a block.

When the time update iterations are complete, all $state_{shared}(p)$ values inside the block are *false*; only states on the extra band can be active. Hence, state values inside the block do not need to be copied into the global memory; only arrival time values are copied from the shared memory to the global memory. Once

time copying is done, if the current thread has out-of-the-block neighbours q in active state ($state_{shared}(q) = true$), then it sets the global state of q to *true*. Both $ThreadFenceBlock$ and $ThreadFence$ in Algorithm 12 ensure that the time update is visible to neighbouring points (and, generally, other threads) before the neighbour's state activations: the first—for neighbours in the same block, the second—for all neighbours.

7.2.4 Implementation details

The global memory (no texture or constant memory used in order to keep current implementations simple) requirements for both FDM_{sync} and $\text{FDM}_{\text{async}}$ implementations are 14 bytes per pixel position: one byte for input image, one byte for watershed label, one byte for state, and eight—for double precision arrival time. $\text{FDM}_{\text{async}}$, in addition, consumes block-level shared memory for propagation step, including arrays for input image, label, state, and time (all with extra bands of values), plus the flag *blockchange* for iterations. For instance, a 16×16 block requires $(16 + 2) \times (16 + 2) \times (1 + 4 + 1 + 8) + 1 = 4537$ bytes.

The algorithmic parameters of FDM include the set of seed points, the watershed or other partitioning labels, and the propagation speed, which are all task dependent. The only FDM implementation parameter is the block size, the generally recommended value of which is 16×16 for 2D and 8×8×8 for 3D images (the corresponding discussion can be found in Section 7.4.2).

As already mentioned, our FDM_{sync} implementation is designed for 3D images and can also be applied on lower-dimensionality images. We propose separate 2D and 3D implementations of $\text{FDM}_{\text{async}}$ to eliminate redundant use of shared memory in the propagation step, as in the case of asynchronous path reducing watersheds.

7.3 Simple dashing method

A simpler and faster to execute counterpart to the fast dashing method is called the *simple dashing method (SDM)*. This is an alternative hybrid approach combining 'dashing' inside watershed (or other partitioning) regions with more relaxed propagation rules across region boundaries. Instead of modelling front development with solutions to the eikonal equation, SDM relies on arrival time promotion from a single neighbour. The simple dashing method is presented in Algorithm 13.

Algorithm 13 Simple dashing parallel method

1: **for all** $p \in I$ in parallel **do** ▷ Initialisation of arrival times and states
2: **if** $seed_{global}(p)$ **then** ▷ implementation re-uses $state_{global}$ for $seed$
3: **for all** $q \in N(p)$ **do**
4: $state_{global}(q) \leftarrow true$
5: **end for**
6: $time_{global}(p) \leftarrow 0$
7: **else**
8: $time_{global}(p) \leftarrow \infty$
9: **end if**
10: **end for**

11: **repeat** ▷ Propagation of arrival times
12: **for all** $p \in I$ in parallel **do**
13: **if** $state_{global}(p)$ **then**
14: $state_{global}(p) \leftarrow false$
15: $time_{local}(p) \leftarrow time_{global}(p)$
16: **for all** $q \in N(p), time_{global}(q) < time_{local}(p)$ **do**
17: **if** $label_{global}(q) = label_{global}(p)$ **then**
18: $time_{local}(p) \leftarrow time_{global}(q)$ ▷ 'dash' inside watershed region
19: **else**
20: $time_{local}(p) \leftarrow \min\left(time_{local}(p), time_{global}(q) + \frac{1}{f(p)}\right)$
21: **end if**
22: **end for**
23: **if** $time_{local}(p) < time_{global}(p)$ **then**
24: $time_{global}(p) \leftarrow time_{local}(p)$
25: $ThreadFence$ ▷ ensure write order
26: **for all** $q \in N(p), time_{global}(q) > time_{local}(p)$ **do**
27: $state_{global}(q) \leftarrow true$
28: **end for**
29: **end if**
30: **end if**
31: **end for**
32: **until** $\neg change$ ▷ $change$ is $true$ while there are active states

A comparative inspection of Algorithm 13 against Algorithm 9 reveals differences

only in lines 16–22(21). FDM, for arrival time improvement, attempts solution of eikonal equation (line 21 of Algorithm 9) as an alternative to inside-region 'dashing'. SDM, on the other hand, updates the local arrival time from the arrival time of one neighbour, either 'dashing' inside regions (line 18 of Algorithm 13) or using local propagation speed (line 20 of Algorithm 13) across region boundaries. The latter case is reminiscent of the simplest solution of the eikonal equation (line 5 of Algorithm 10). Everything else is identical between FDM and SDM.

Similar to FDM, we provide two implementations of SDM: synchronous SDM_{sync} applicable to both 2D and 3D images, and block-asynchronous $\text{SDM}_{\text{async}}$ implemented in separate variants for 2D and 3D images. Each main step of the algorithm—initialisation and propagation—is implemented with a kernel in a thread-per-pixel manner, with iterative application of the propagation kernel until convergence. Memory requirements, algorithmic and implementation parameters are similar to FDM.

The SDM procedure can also be interpreted as a shortest path implementation. The image can be represented as a graph, where each pixel position is a node and there is a weighted edge between every pair of neighbouring pixels. We can define the weight of every directed edge as 0 if the two nodes belong to the same region, or $\frac{1}{f(p)}$ if they do not:

$$W(p,q) = \begin{cases} 0 & \text{if } label(p) = label(q) \\ \frac{1}{f(p)} & \text{if } label(p) \neq label(q). \end{cases}$$

Then the arrival time value of each pixel in the SDM output corresponds to the shortest distance from that pixel position to the nearest seed point. SDM, in principle, illustrates that watershed-region-incorporating variants of established algorithms other than eikonal solvers can also be easily implemented in a similar way for GPU parallel execution.

7.4 Fast dashing method results

We use the same BSDS500 image dataset for a number of FDM benchmarking experiments to reveal execution time patterns for different parameter values and implementation choices. The algorithm performance is further analysed for the three 3D test images from the previous chapter. The default implementation settings[1] throughout the experiments are

- a single seed position at the top left corner of (the first slice of) the image, $p = 0$
- inverse linear propagation speed $f_{lin}(p) = \frac{1}{image(p)+1}$
- watershed partitioning labels of a PRW_{bal} application on the input image (without any preprocessing)
- 16×16 2D block size; 8×8×8 3D block size

We consistently change the implementation settings one at a time, and report how the FDM execution is affected by the change. Again, each execution time reported is an average of ten runs. Finally, some qualitative results are presented to illustrate how FDM works.

7.4.1 Synchronous vs. block-asynchronous

The difference between the synchronous and block-asynchronous implementations of the fast dashing method is considered first, with all the implementation settings fixed at the defaults. Figure 7.1 presents the comparison of FDM_{sync} and $\text{FDM}_{\text{async}}$ on the BSDS500 images. The execution times are illustrated for two orderings of the 500 images: increasing order of PRW_{sync} runtimes, consistent with the results in the previous chapter; increasing order of $\text{FDM}_{\text{async}}$, easier to interpret and preferred in this chapter for various FDM result plots.

[1]These implementation settings are *not* recommended best performance values; they are just simplest options.

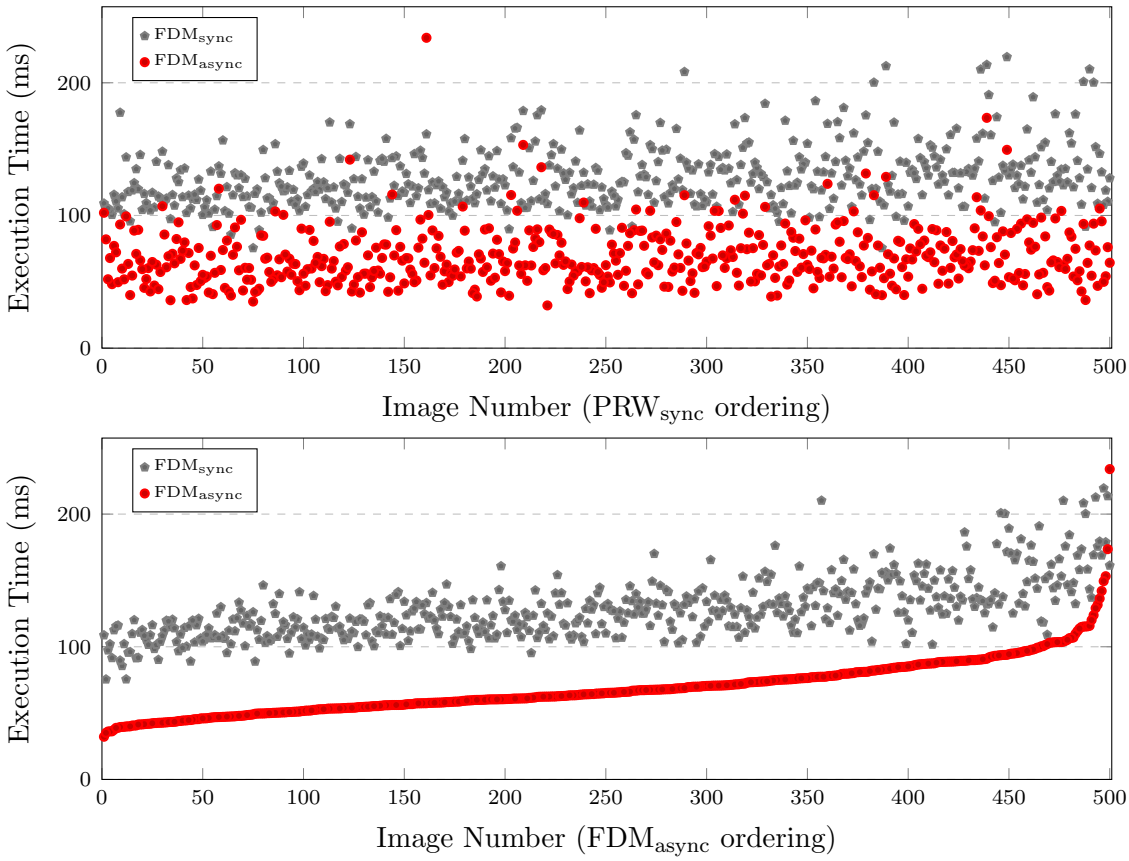


Figure 7.1: The execution times (in ms) of the synchronous and block-asynchronous fast dashing method implementations on BSDS500. Two orderings of the 500 images illustrated: increasing order of PRW_{sync}, consistent with the results in the previous chapter; increasing order of $\text{FDM}_{\text{async}}$, preferred in this chapter for various FDM results.

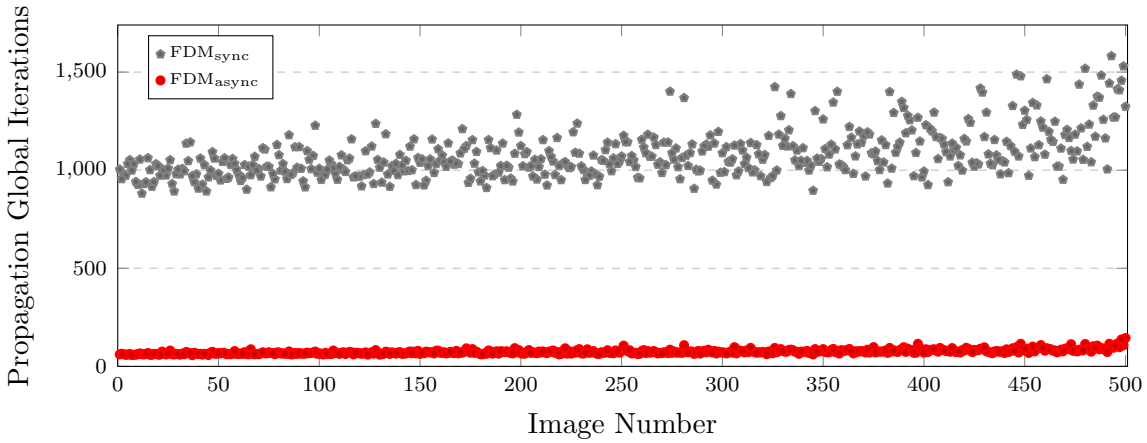


Figure 7.2: The number of global iterations during the propagation step of FDM_{sync} and $\text{FDM}_{\text{async}}$ on BSDS500.

On 499 out of the 500 images, $\text{FDM}_{\text{async}}$ is faster than FDM_{sync}. This consistent trend excludes only a single image—the 500^{th} image in the second, $\text{FDM}_{\text{async}}$ performance based, ordering. The sum of the execution times over the 500 images is around 64.149 sec for FDM_{sync} and 34.523 sec for $\text{FDM}_{\text{async}}$. The execution times of the former on individual images are in the range 75.4739–219.4439 ms, with the average at 128.2987 ms. For $\text{FDM}_{\text{async}}$ the range of runtimes is 32.1981–233.9138 ms (173.5847 ms, excluding image 500), with the average at 69.0455 ms. The average of the per-image speedups of $\text{FDM}_{\text{async}}$ over FDM_{sync} is 1.948 times.

The equivalence of the results of the two implementations is confirmed by the absolute match between the arrival time arrays produced by both implementations across all 500 images. $\text{FDM}_{\text{async}}$ achieves the execution time improvement over FDM_{sync} by replacing the global synchronisation after each propagation update with a reduced number of global synchronisation points and intermediate block-level synchronised iterations. The reduction in the number of global iterations of $\text{FDM}_{\text{async}}$ as opposed to FDM_{sync} is illustrated in Figure 7.2.

The number of global iterations ranges from 881 to 1,581.9 for FDM_{sync}; for $\text{FDM}_{\text{async}}$ it is 57–144.4[2]. The per image reduction of the number of global itera-

[2]The fractional numbers are the result of averaging across ten executions.

tions is 14.559 times on average, starting at 9.169 times and reaching 17.819 times. A global iteration of $\text{FDM}_{\text{async}}$ is generally a few times slower than a global iteration of FDM_{sync}, since the first includes several block-level update iterations. Hence, for an average improvement of almost fifteen times of the number of global iterations of the propagation step, $\text{FDM}_{\text{async}}$ is on average only around twice faster than FDM_{sync} on BSDS500.

7.4.2 Parameters

The effect of algorithmic and implementation parameters on FDM is considered by altering the experimental defaults for $\text{FDM}_{\text{async}}$ and measuring execution times. The one parameter that is not analysed in this way is the set of seed points. The reason for this is that the number and the position of the seed points do not directly affect arrival time calculation duration and propagation patterns. They do affect the number of time update iterations and, hence, number of global iterations, just like image size or image greyscale values do. The set of seeds is a task dependent parameter and does not require general execution time analysis experiments. The default value of a single seed point $p = 0$ is an adequate choice for examining the effect of the other parameters on the $\text{FDM}_{\text{async}}$ execution time.

Figure 7.3 illustrates the effect of different propagation speed terms on the $\text{FDM}_{\text{async}}$ execution times. We consider four speed options popular in segmentation applications of eikonal solvers (see Section 3.4). These are

- $f_{lin}(p) = \frac{1}{image(p)+1}$
- $f_{quad}(p) = \frac{1}{image(p)^2+1}$
- $f_{cub}(p) = \frac{1}{image(p)^3+1}$
- $f_{exp}(p) = 2^{-image(p)}$

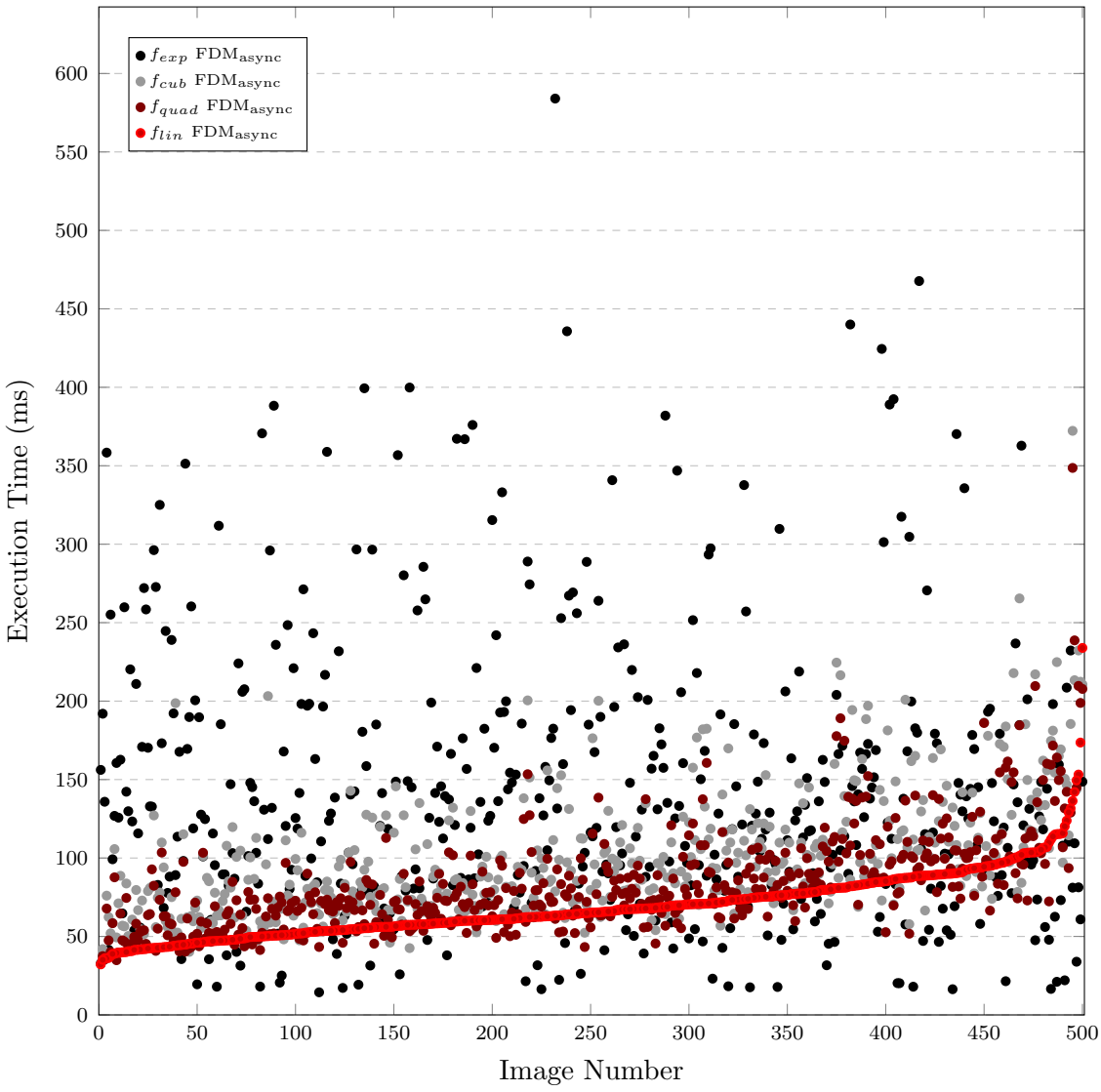


Figure 7.3: The execution times of FDM$_{\text{async}}$ with four speed options.

As can be seen from the plot in Figure 7.3, the runtimes generally increase with the complexity of the speed terms[3]. This is mainly explained by the extra computational burden of speed value calculations. The sums of $\text{FDM}_{\text{async}}$ execution times over BSDS500 are 34.523 sec, 42.017 sec, 49.627 sec, and 70.605 sec for f_{lin}, f_{quad}, f_{cub}, and f_{exp} speeds, respectively. The calculation of f_{exp} is significantly more expensive than f_{lin}, f_{quad}, and f_{cub}; the implementation relies on a separate CUDA function call for f_{exp}, unlike the other three speeds. This explains the largely scattered runtimes with f_{exp} and the big step in the overall execution times. The main conclusion from the results in Figure 7.3 is that simpler propagation speed terms are preferable for efficient calculation of FDM results.

The effect of substituting watershed labels with higher layer labels of hierarchical segmentation is shown in Figure 7.4. Seven hierarchical segmentation layers are considered: watershed layer L1[4] and waterfall layers L2–L7 of decreasing granularity. 'Dashing' propagation inside partitioned regions is computationally much cheaper than solving the eikonal equation. Figure 7.4 clearly illustrates that $\text{FDM}_{\text{async}}$ is faster for partitionings with fewer larger regions. The overall execution times for L1 through L7 are 34.571 sec, 23.907 sec, 18.770 sec, 14.815 sec, 11.541 sec, 8.586 sec, and 6.848 sec. For layer 7, where the segmentation contains only a single region for most images in BSDS500, the variation in execution times is minimal with 'dashing' propagation taking place throughout the images. The speed term and, hence, image greyscale values do not affect the runtimes for such cases. At the same time, L7 $\text{FDM}_{\text{async}}$ results are useless since 'dashing' throughout the image means zero arrival times everywhere. The choice of the best segmentation layer is a tradeoff between runtimes and level of detail in the FDM results.

[3]Note that f_{exp} is calculated using the `pow(double,double)` function from the CUDA Math API rather than `exp2(double)` to reflect the complexity of the general case with varied exponent bases.

[4]Note that there is an occasional small difference between the $\text{FDM}_{\text{async}}$ execution times reported in Figure 7.1 and in Figure 7.4 (legend entry 'L1 $\text{FDM}_{\text{async}}$'). This is natural variation of execution times that is still not completely nullified between ten run averages.

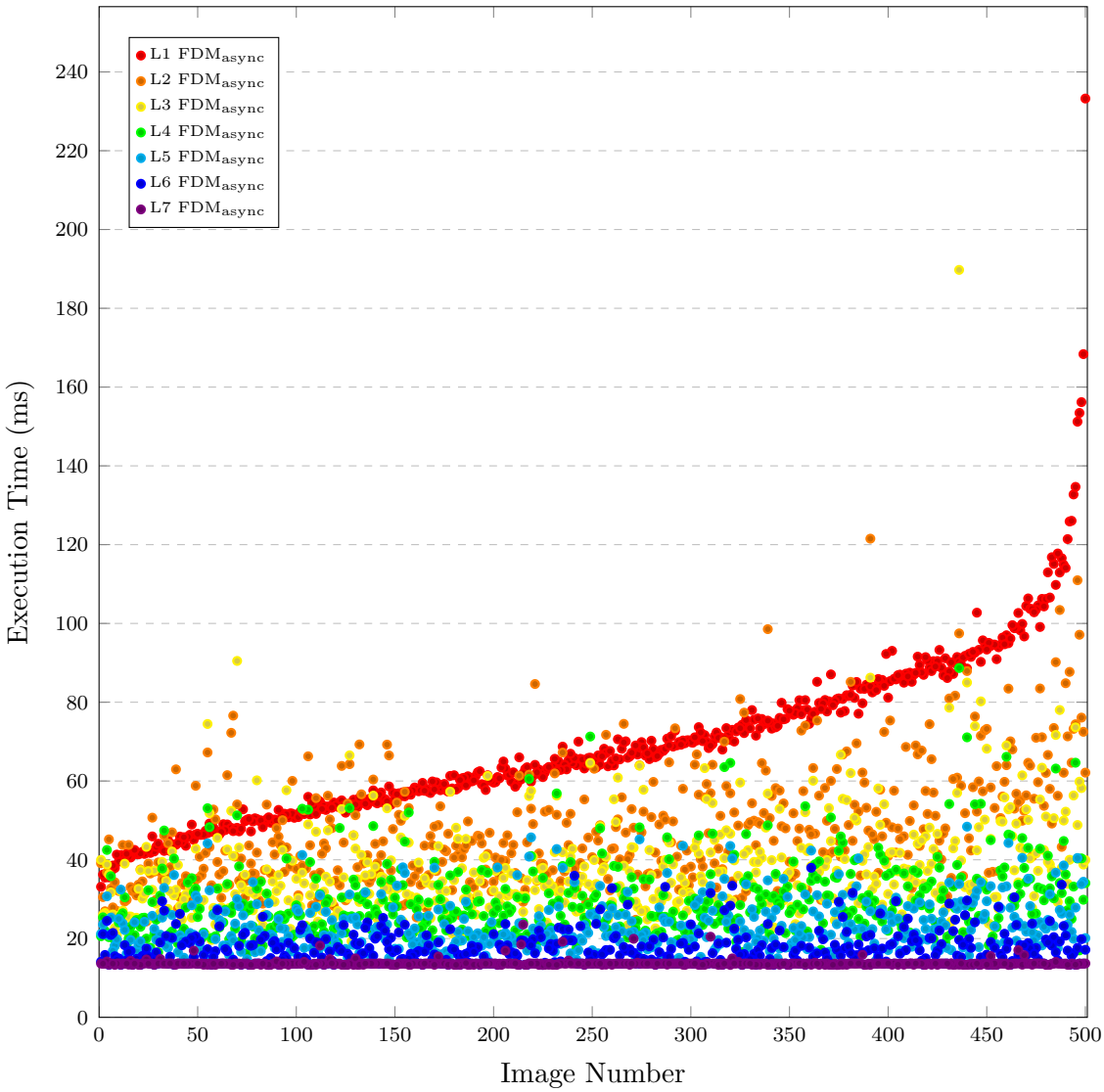


Figure 7.4: The execution times (in ms) of FDM$_{async}$ on BSDS500 with the partitioning label image varying among first seven layers of hierarchical segmentation with PRW$_{bal}$. Since the first layer is the watershed result, the first plot reported fully relies on the default experimental settings of FDM. The next six plots correspond to six waterfall layers of increasing region sizes and, as a result, faster FDM$_{async}$ execution times.

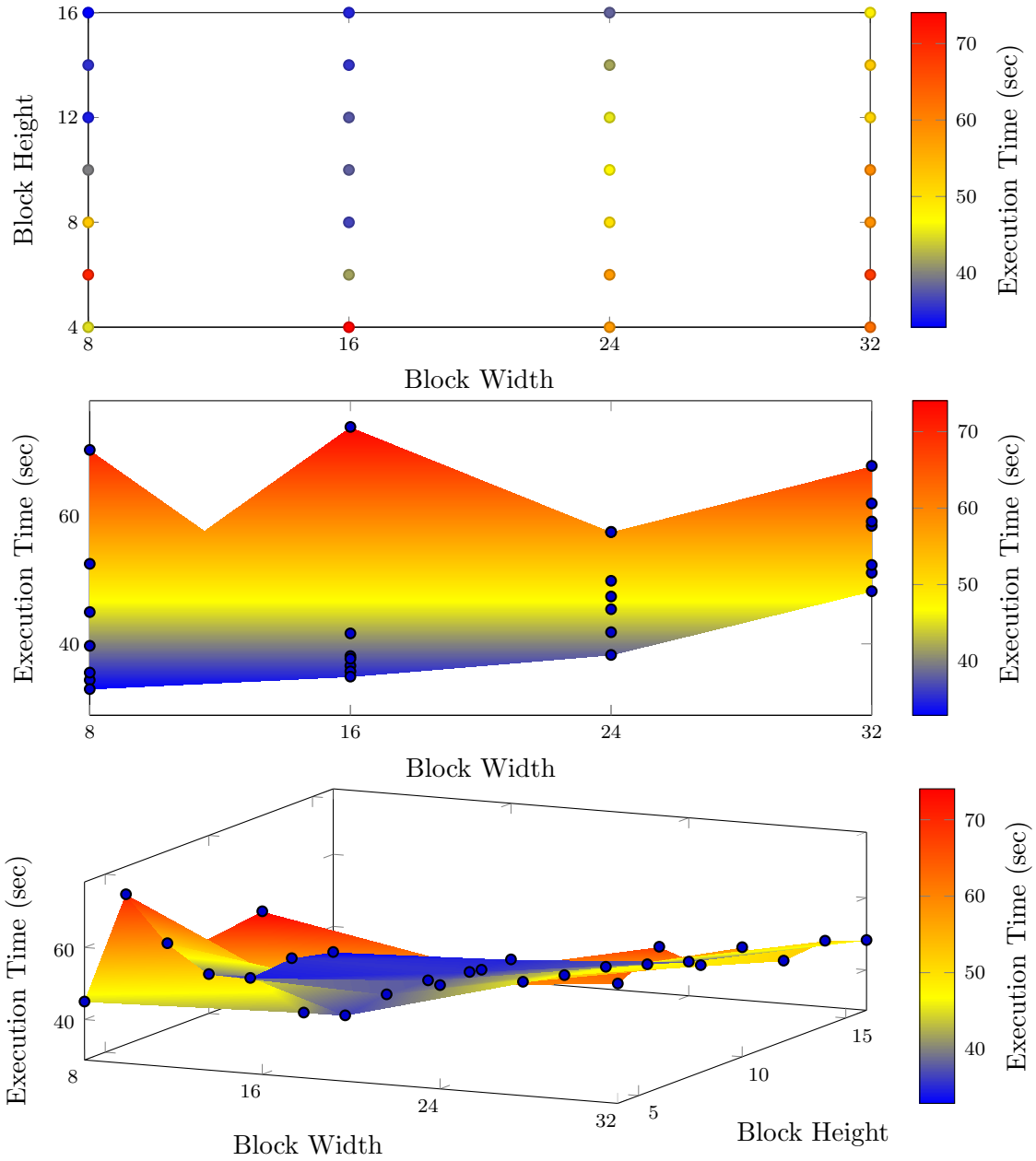


Figure 7.5: The overall execution times (sums of the 500 per image averages) of $\text{FDM}_{\text{async}}$ for different block sizes, two visualisations of the results. The first version encodes the runtime for each block width–height pair within a colourbar scale. The second version interpolates these runtimes into a surface (shown from two different viewpoints).

Table 7.1: The execution times (in sec, average of ten runs per image) of the two variants of fast dashing compared on the 3D test images. The speedups of the block-asynchronous implementation over the synchronous procedure are reported in the last row.

	4,000×4,000×8	500×500×512	320×320×1,250
FDM_{sync}	591.4240	157.0007	125.7998
FDM_{async}	428.9104	160.9361	112.3529
sync vs. async	1.379×	**0.976**×	1.120×

The last one of the default settings to be considered, and the only implementation parameter of FDM, is the CUDA block size. Taking into account the results of the runtimes of PRW_{bal} with various block sizes from the previous chapter (Section 6.3.4), we limit the current experiments to fewer block sizes to be tested. The block width is varied among values 8, 16, 24, and 32; the block height is considered at all the even values in the range 4–16. FDM_{async} performance on BSDS500 with different block sizes is presented in Figure 7.5.

A threshold of less than 38 sec is met by block sizes 8×12, 8×14, 8×16, 16×8, 16×12, 16×14, and 16×16. Reducing the threshold to 35 sec, we get 8×12, 8×16, and 16×16. The best performance of around 32.9 sec is achieved with block size 8×16. Taking into account the fact that the BSDS500 images are quite small (481×321 or 321×481) and the minor difference in the overall performance for block sizes 8×16 and 16×16, we propose that size 16×16 be generally preferred for 2D images. The choice of size 16×16 also means implementation consistency between PRW and FDM.

7.4.3 Large 3D images

The performance of FDM on large 3D images is analysed using the default settings on the example of the same three 3D test images from the previous chapter. The comparison of FDM_{sync} and FDM_{async} on these images is given in Table 7.1.

There is a significant variation in runtimes for the different images. This is explained by the contents of the images: there is a lot of in-slice greyscale vari-

Table 7.2: The comparison of the execution times (in sec) of our FDM_{async} with different 3D width×height×depth block sizes on the three 3D test images. The best performance for each image is highlighted.

width	height	depth	4,000×4,000×8	500×500×512	320×320×1,250
8	8	8	428.910	160.936	112.353
8	8	12	435.681	136.957	100.031
8	12	8	333.923	142.790	97.418
12	8	8	**324.103**	**136.157**	**91.313**
16	16	1	351.559	253.872	157.741
16	16	2	412.772	232.333	145.379
16	16	3	356.222	169.158	106.444

ation in contrast with consistent values along the z dimension. Hence watershed regions stretch across multiple slices, and watershed calculation is the slowest for the 320×320×1,250 test image, as shown in the previous chapter. FDM propagation, on the other hand, is faster inside watershed regions and, hence, along the z dimension of these images.

FDM_{sync} shows its best execution time of around 2.1 minutes on the 320×320×1,250 image; the 4,000×4,000×8 gets the worst performance of almost 10 minutes. FDM_{async} improves this runtime to around 7.1 minutes; its performance on the 320×320×1,250 image is below 1.9 minutes. So, speedups of 1.379 and 1.12 times are seen on these two images.

The case of the 500×500×512 image is more interesting: the 2.617 minutes execution of FDM_{sync} is actually slightly worsened to 2.682 minutes by FDM_{async}. Such cases are obviously rare and the general preference should be for the block-asynchronous implementation of FDM.

In addition, the implementation parameter—block size—in 3D is also reviewed for these test images. The algorithmic parameters (seeds, speed, segmentation regions) do not invite separate examination for the specific 3D images. The trends drawn from Figures 7.3 and 7.4 are generic and apply to any data.

FDM_{async} execution times on the test images with a few 3D block sizes are presented in Table 7.2. Due to the high memory requirements of the FDM implementation, and, in particular, double-precision floating point arithmetic, an FDM_{async} block can consist of up to 768 threads on the GPU device used (see Section 2.4 for specifications). Therefore, only block sizes complying to this requirement are tested; a brief technical explanation follows.

GPU devices with compute capability 5.0 (the family that our GeForce GTX 960M belongs to) contain 65,536 32-bit registers per streaming multiprocessor (SM). The propagation kernel of our FDM_{async} implementation compiles to require 78 registers per thread. This sums to 2,496 registers per warp of 32 threads. However, registers are allocated at units of 256 at the level of warps. Thus, a warp of FDM_{async} threads will actually reserve 2,560 registers. Warp allocation granularity for these devices stands at 4, and the limit of maximal number of warps per SM will be a multiple of 4. So, the actual limit for the FDM_{async} implementation is $65{,}536 \div 2{,}560 = 25.6$ rounded down to 24 warps per SM and, hence, per block. The limit of threads per block then becomes $24 \times 32 = 768$.

Table 7.2 shows that there is no general agreement among the tested block sizes for the three images, although the best performance for all three images is achieved with the same block size. We can compare, for instance, the three runtimes for 8×8×12 and 16×16×1 blocks. FDM_{async} performs fast on the 500×500×512 and 320×320×1,250 images with block size 8×8×12. For block size 16×16×1, on the other hand, FDM_{async} is slower on those two test images and much faster on the 4,000×4,000×8 test image. The main reason for such runtimes variations is the size variation among the images.

The other important factor is runtime optimisation for the specific FDM implementation and the specific GPU hardware. With a limit of 24 warps per SM, blocks of 16 warps (8×8×8, 16×16×2) reserve redundant computational power using 16/24

warps per SM. The rest of the considered block sizes utilise 24 warps per SM: three blocks with size 16×16×1 of 8 warps are allocated per SM; one block of 24 warps is allocated per SM, in case of the other four block sizes of 768 threads (8×8×12, 8×12×8, 12×8×8, 16×16×3).

Taking into account the possible variation in image sizes, the different specifications of the hardware, and possible implementation differences, our general recommendation is for the 8×8×8 block size. This can be then altered when the procedure is fine-tunes for specific images and/or hardware.

7.5 Simple dashing method results

An examination of the simple dashing method runtimes and the effects of its parameters follows, using the same experimental image data. The default parameter settings—a single seed at the image top left corner, inverse linear speed, watershed labels, 16×16 and 8×8×8 block sizes—mimic exactly those of FDM. The main interest of this study is to check how much computationally cheaper SDM is in comparison to FDM.

7.5.1 Synchronous vs. block-asynchronous

We start with a comparison of SDM_{sync} and $\text{SDM}_{\text{async}}$ execution times on the BSDS500 dataset and our 3D test images. Figure 7.6 contains details of the synchronous vs. block-asynchronous SDM execution for the BSDS500 images. Two different orderings of the 500 images are presented. First, images are sorted in the increasing order of the $\text{FDM}_{\text{async}}$ runtimes in the top plot, which matches the image order in the bottom plot of Figure 7.1. An analogy between the FDM and SDM implementations can be drawn with these two plots: the 'clouds' of the runtimes slowly move up towards the right end of the plots for all four implementations—FDM_{sync}, $\text{FDM}_{\text{async}}$, SDM_{sync},

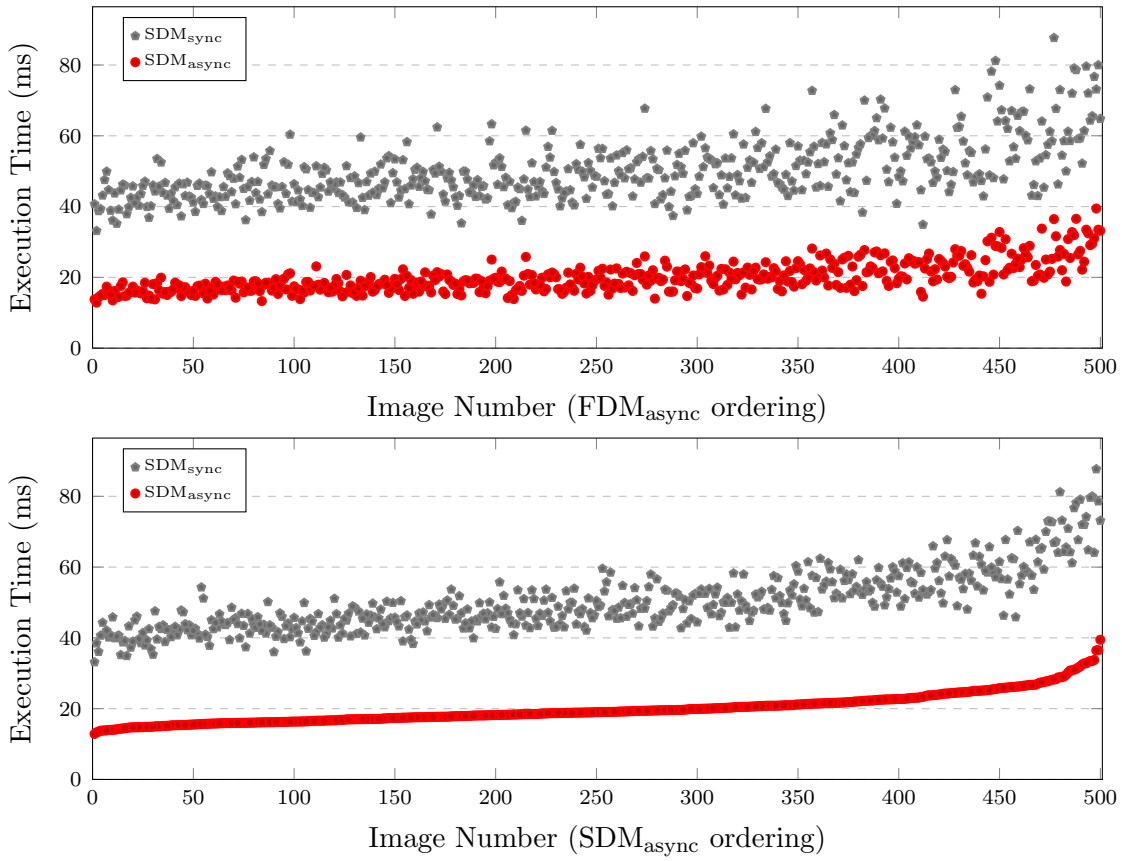


Figure 7.6: The execution times (in ms) of the synchronous and block-asynchronous simple dashing method implementations on BSDS500. Two orderings of the 500 images illustrated: increasing order of $\text{FDM}_{\text{async}}$, consistent with the results in the previous section; increasing order of $\text{SDM}_{\text{async}}$, preferred in this section for various SDM results.

Table 7.3: The execution times (in sec, average of ten runs per image) of the two variants of simple dashing compared on the 3D test images. The speedups of the block-asynchronous implementation over the synchronous procedure are reported in the middle row. The last two rows include a comparison of $\text{FDM}_{\text{async}}$ and $\text{SDM}_{\text{async}}$.

	4,000×4,000×8	500×500×512	320×320×1,250
SDM_{sync}	179.1734	43.7897	33.9170
$\text{SDM}_{\text{async}}$	139.5882	37.2264	24.1529
sync vs. async	1.284×	1.176×	1.404×
$\text{FDM}_{\text{async}}$	428.9104	160.9361	112.3529
$\text{FDM}_{\text{async}}$ vs. $\text{SDM}_{\text{async}}$	3.073×	4.323×	4.652×

and $\text{SDM}_{\text{async}}$. However, the value range for SDM implementations is much lower. Second, the increasing order of the $\text{SDM}_{\text{async}}$ runtimes is used for sorting images in the bottom plot of Figure 7.6, which better visualises the link between SDM_{sync} and $\text{SDM}_{\text{async}}$ performance.

The sum of the SDM_{sync} runtimes across the 500 images is approximately 25.061 sec, which is already below the 34.523 sec of $\text{FDM}_{\text{async}}$. $\text{SDM}_{\text{async}}$ achieves an even better overall performance of 9.964 sec, and it is faster than SDM_{sync} on every image in BSDS500. So, the simplified algorithm runs more than three times faster than the original algorithm, comparing the overall runtimes of the asynchronous implementations. The runtimes for individual images range between 33.1541 ms and 87.6854 ms for SDM_{sync}, and between 12.8695 ms and 39.4554 ms for $\text{SDM}_{\text{async}}$. The latter is on average 2.5446 times faster than the former; the extremities of the time improvement factor are at 1.7517 and 3.4725.

The average runtimes for SDM_{sync} and $\text{SDM}_{\text{async}}$ on the 3D test images are provided in Table 7.3. For the first test image the runtime is reduced from almost 3 minutes to 2.3 minutes by the block-asynchronous implementation. The speedup is smaller (1.176 times) for the second image—43.7897 sec and 37.2264 sec, respectively. Finally, a good improvement of 1.404 times from 33.917 sec down to 24.1529 sec is achieved for the 320×320×1,250 image. The results of the synchronous and block-

asynchronous SDM implementations are confirmed as identical when produced arrival times across an image are compared for all experimental images.

The comparison of $\text{FDM}_{\text{async}}$ and $\text{SDM}_{\text{async}}$ execution times on the three 3D test images (last row of Table 7.3) reveals that $\text{SDM}_{\text{async}}$ is on average around four times faster to calculate. A study of the qualitative differences between the two algorithms is given in Section 7.6.

7.5.2 Parameters

We look at the algorithmic parameters of SDM and how they affect the execution times of the algorithm. Figures 7.7 and 7.8 illustrate the $\text{SDM}_{\text{async}}$ runtimes with different speed terms and with different segmentation partitionings on BSDS500. Similar to FDM, the set of seeds does not impact SDM arrival time calculations; it only defines the order of positions considered for time updates. No experimental comparison of different sets of seeds for SDM calculation is provided. The only implementation parameter—block size—is chosen based on the similar experiments for FDM: for 2D images 16×16 blocks and for 3D images 8×8×8 blocks are suggested.

As in Figure 7.3, four different speed terms are compared in Figure 7.7: f_{lin}, f_{quad}, f_{cub}, and f_{exp}. The sums of the average runtimes over the 500 images are 9.964 sec, 12.374 sec, 14.655 sec, and 38.895 sec, respectively. All four are significantly faster than their $\text{FDM}_{\text{async}}$ counterparts (34.523 sec, 42.017 sec, 49.627 sec, 70.605 sec). However, the improvement of the $\text{SDM}_{\text{async}}$ execution with f_{exp} speed over $\text{FDM}_{\text{async}}$ with the same speed term is smaller. While the FDM implementation calculates the propagation speed only once per pixel update (line 21 of Algorithm 9), the SDM implementation may repeat the speed value calculation (line 20 of Algorithm 13) if a few neighbours have a better arrival time than the considered position.

A possible local solution to the problem for SDM may be the calculation of the speed value once at the beginning of the CUDA kernel execution, its storage in local

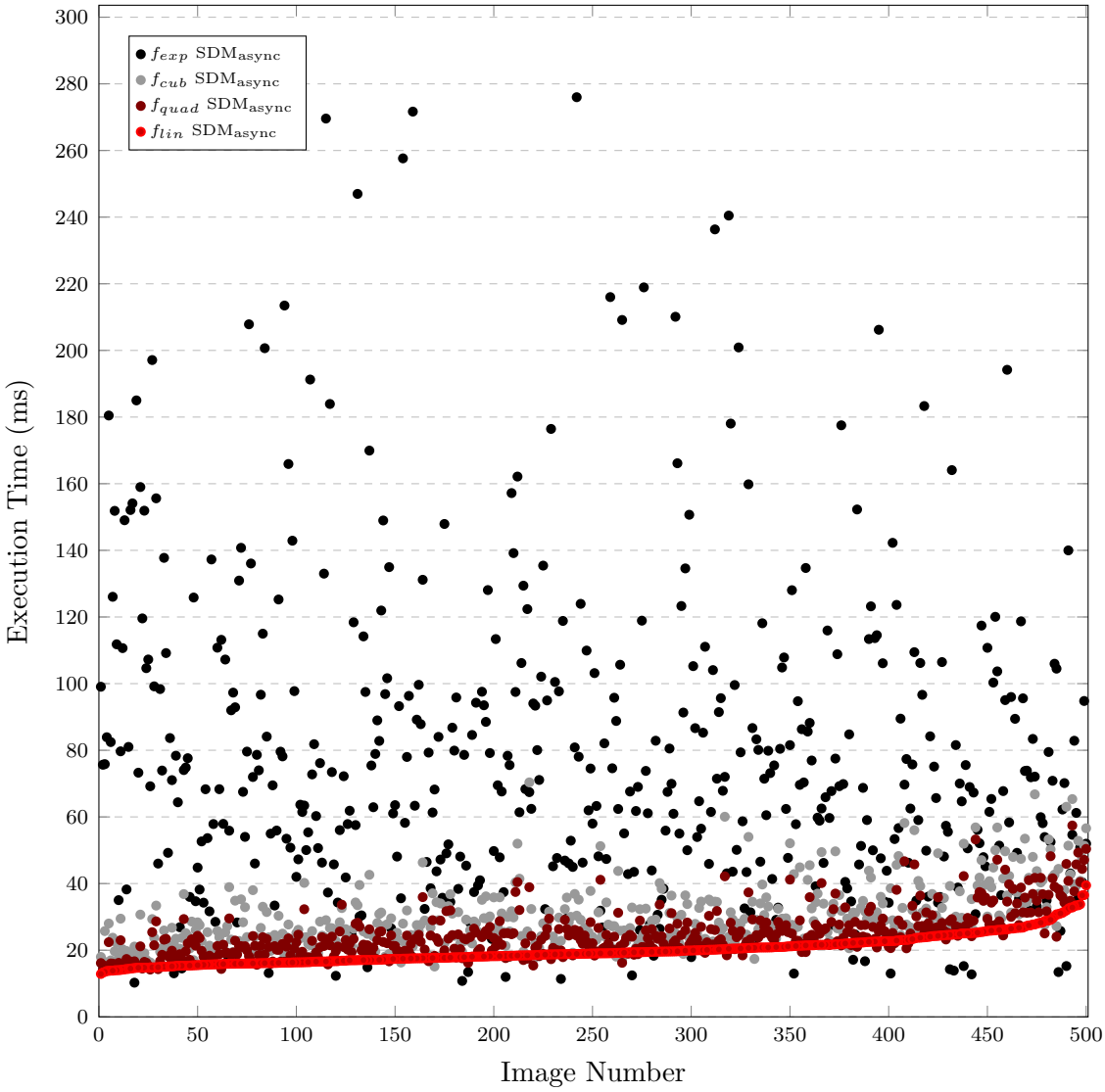


Figure 7.7: The execution times of SDM$_{\text{async}}$ with four speed options.

memory, and reuse. An alternative global solution is to create a speed array with all pixel positions in the global memory once during initialisation step and just read speed values from there every time an arrival time across region boundaries is calculated during propagation step. The second approach would also largely improve FDM runtimes with expensive speed terms like f_{exp}. We have tried to make our current experimental implementations of FDM and SDM memory-conscious and applicable on larger data given the limited GPU memory sizes.

Layers 1–7 of hierarchical segmentation with PRW_{bal} are considered as segmentation partitionings for $\text{SDM}_{\text{async}}$ calculation in Figure 7.8. As expected, based on the $\text{FDM}_{\text{async}}$ results in Figure 7.4, $\text{SDM}_{\text{async}}$ runtimes tend to go down for segmentations with larger regions. The overall BSDS500 runtimes for layers 1 through 7 are 9.965 sec, 9.214 sec, 8.004 sec, 6.88 sec, 5.94 sec, 5.033 sec, and 4.427 sec. Since the vast majority of the layer 7 segmentations on BSDS500 consist of a single region, the L7 $\text{SDM}_{\text{async}}$ runtimes are consistently small, around 8.68–12.34 ms with the average at 8.855 ms. While calculated very efficiently, L7 $\text{SDM}_{\text{async}}$ outputs are virtually useless. The tradeoff between runtime efficiency and details in output, suggested earlier for $\text{FDM}_{\text{async}}$ calculation, can also be applied to $\text{SDM}_{\text{async}}$.

7.6 Qualitative results

The outputs of both FDM and SDM algorithms have the form of floating point arrival times for all pixel positions in the image. These arrival times can be used to assess how well a position relates to the set of seeds: the lower the time value, the stronger the relationship. One approach to the visualisation of these results is linear mapping of the arrival time range into greyscale range for digital images:

$$[0, \max\{time(p) | \forall p \in I\}] \quad \mapsto \quad \{0, 1, \ldots, 255\}.$$

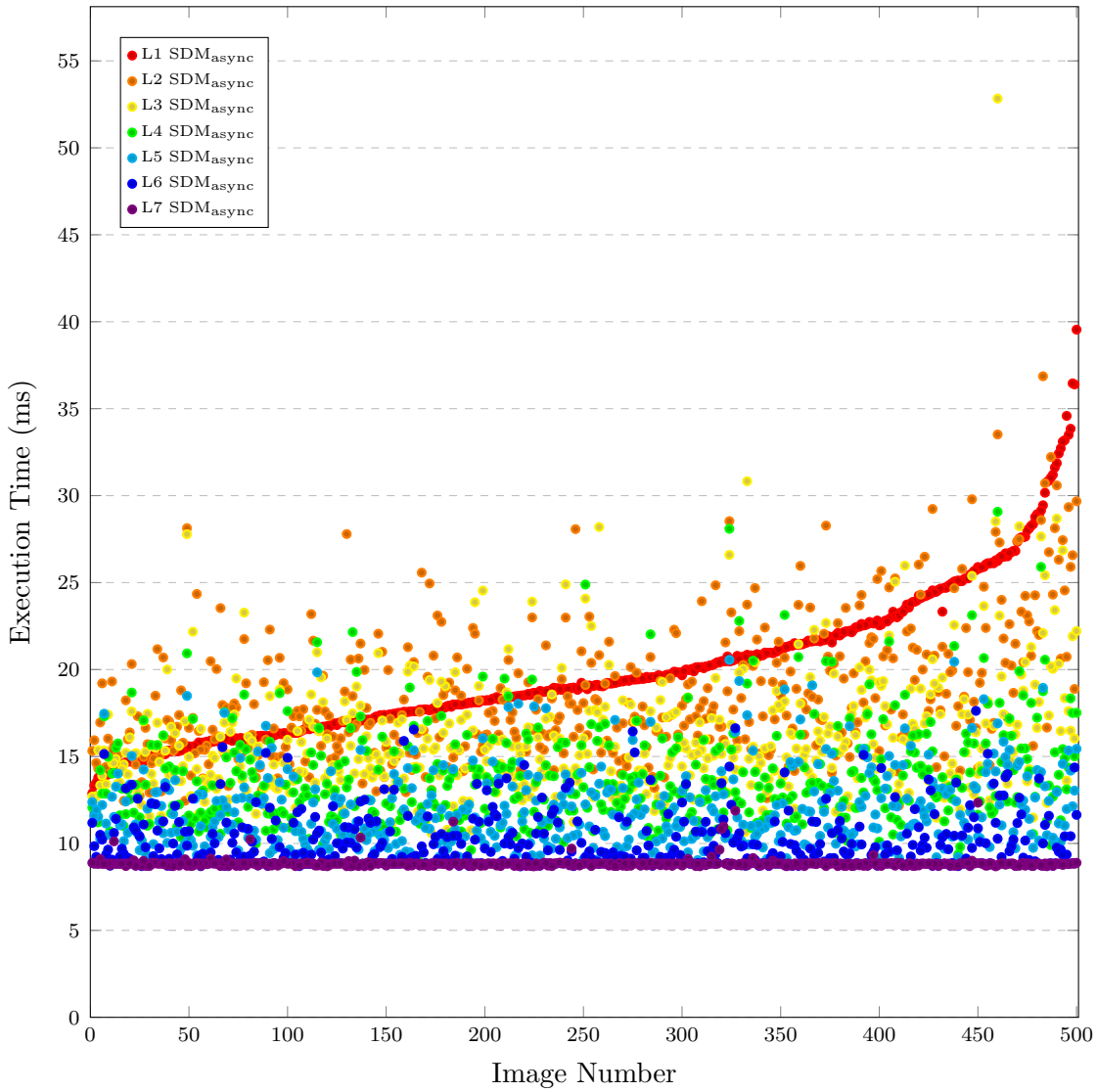


Figure 7.8: The execution times (in ms) of SDM_{async} on BSDS500 with the partitioning label image varying among first seven layers of hierarchical segmentation with PRW_{bal}. Since the first layer is the watershed result, the first plot reported fully relies on the default experimental settings of SDM. The next six plots correspond to six waterfall layers of increasing region sizes and, as a result, faster SDM_{async} execution times.

7.6.1 Visual comparison

Figure 7.9 illustrates the FDM and SDM results with their default experimental settings applied on the gradient magnitude images, produced from the minimally smoothed sample Lena, abdominal CT slice, and knee MRI slice images. The FDM results in the fourth row and the SDM results in the fifth row are visualised using the above-mentioned linear mapping. A darker pixel denotes a lower arrival time; a lighter pixel—a higher arrival time. For each image, FDM and SDM results are visually similar. This means that FDM can potentially be traded for SDM for the execution time gain without much loss of detail.

The effects of the FDM parameters on qualitative results are presented in Figures 7.10 (speed) and 7.12 (partitioning layer in hierarchical segmentation). Corresponding SDM parameter outcomes are in Figures 7.11 and 7.13. The implementation type (synchronous vs. block-asynchronous) and the block size do not have any impact on the arrival time values resulting from FDM or SDM execution.

In Figures 7.10 and 7.11 the four speed terms $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} (see Section 7.4.2) are tested on the gradient magnitude images, constructed from the minimally smoothed greyscale images 1, 250, and 500 (PRW_{sync} ordering) from BSDS500. All the other parameters are set to the experimental defaults—$p = 0$ as seed and PRW_{bal} watershed of the gradient magnitude image as partitioning. When gradually moving from f_{lin} to f_{exp}, propagation becomes slower and arrival time range increases. In addition, for f_{exp}, the difference in arrival times of neighbouring pixels becomes exponentially proportional to the gradient magnitude image value. Hence, in case of linear mapping to the greyscale range, most arrival times are mapped into black or very dark shades, and only few arrival times appear white in the mapped image.

Figures 7.10 and 7.11 also show how object boundaries (high gradient magnitude areas) become sharper for the slower propagation speeds. For instance, FDM results for image 1 with speeds f_{lin}, f_{quad}, and f_{cub} can be compared (rows 4–6 in Figure 7.10).

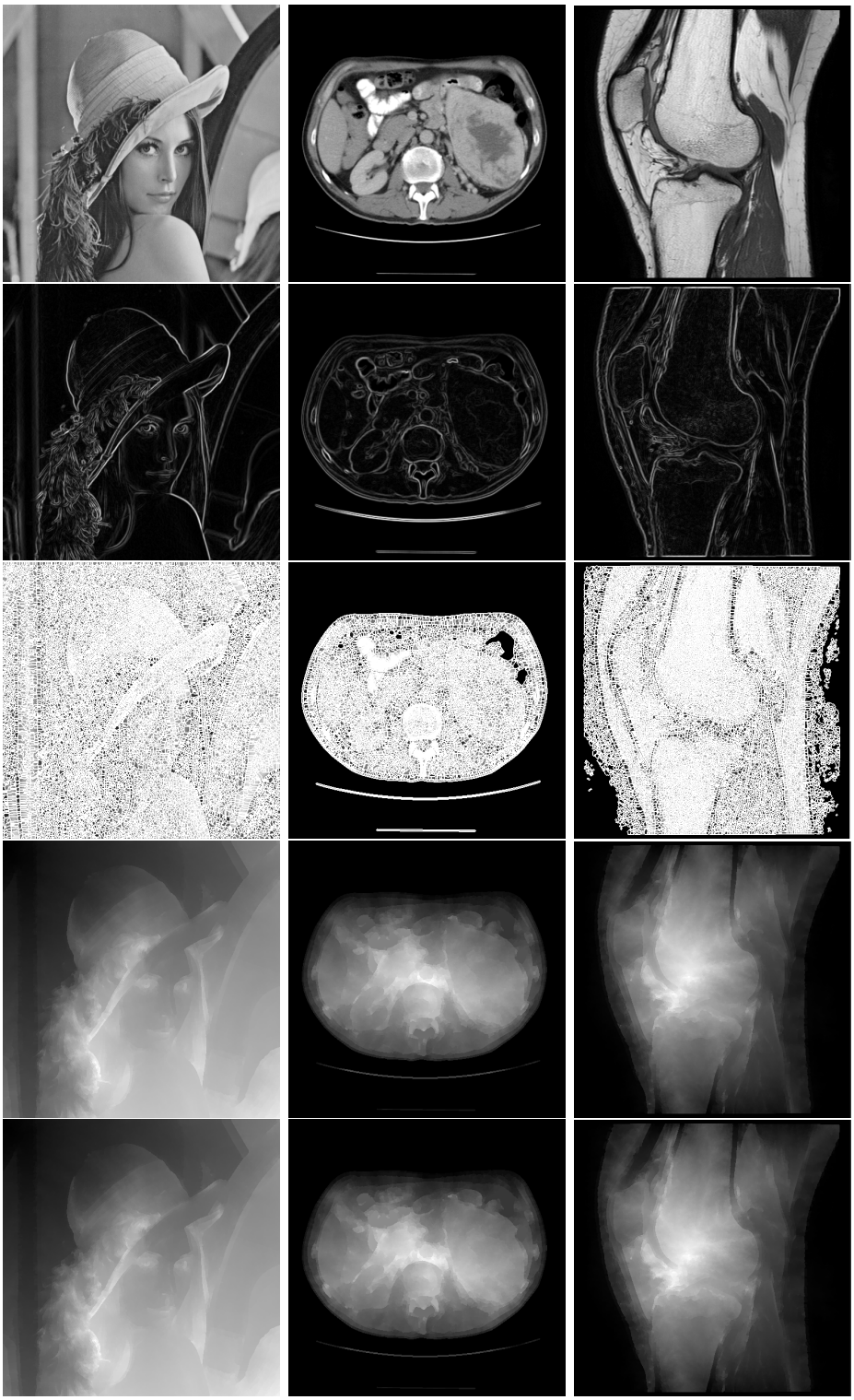


Figure 7.9: FDM and SDM results on three sample images: Lena, abdominal CT slice, and knee MRI slice. The original greyscale images are in the first row; gradient magnitude images constructed after minimal Gaussian blurring in the second row; the original images overlaid with watershed lines in the third row. FDM and SDM default settings results on gradient magnitude images in the fourth and fifth rows, respectively.

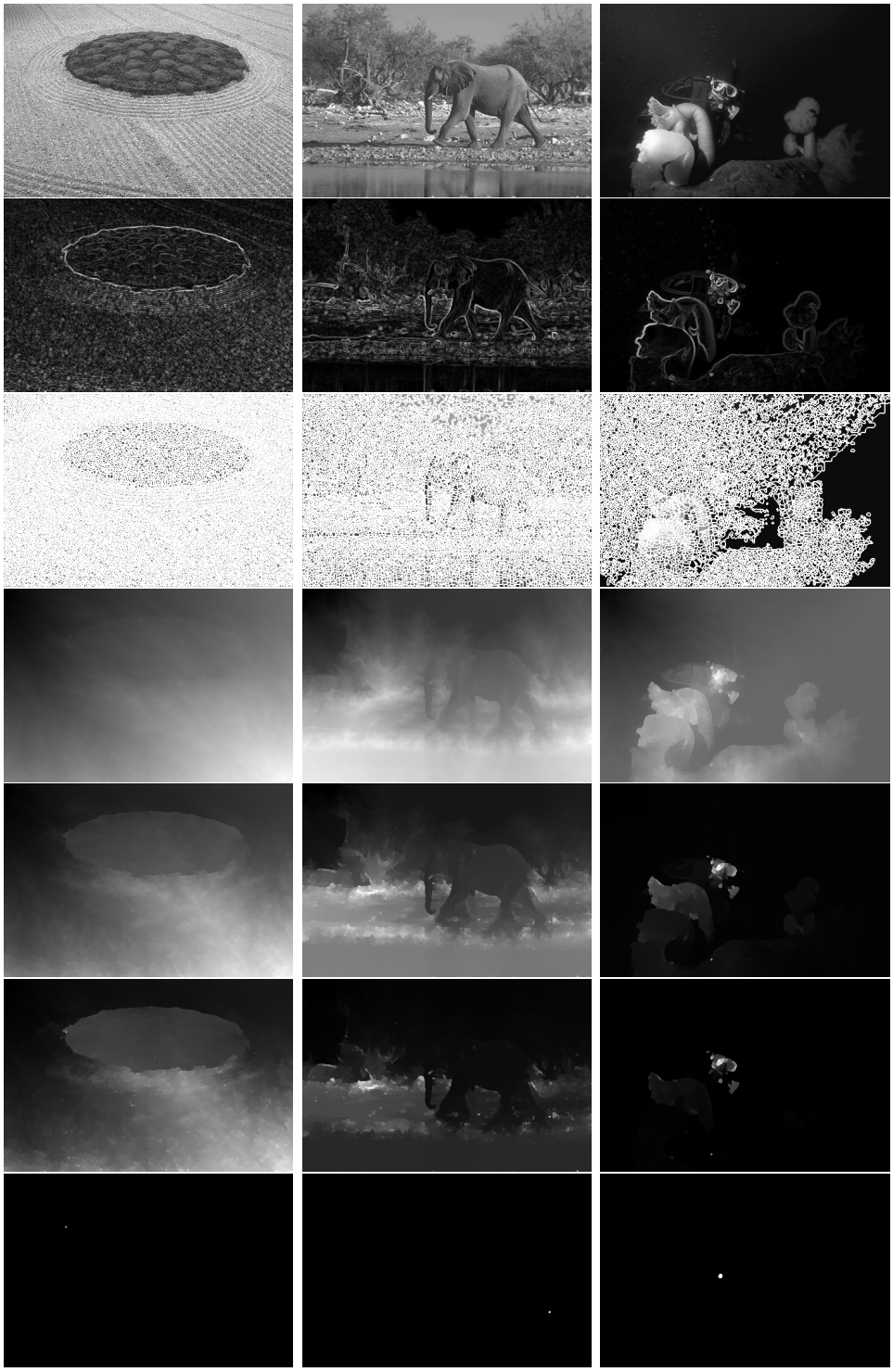


Figure 7.10: The effect of different FDM speeds on the arrival time images. Images 1, 250, and 500 (PRW_{sync} ordering) from BSDS500; corresponding gradient magnitude images constructed after minimal smoothing; original images overlaid with watershed lines. FDM arrival time images with speeds $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} in rows 4–7, respectively.

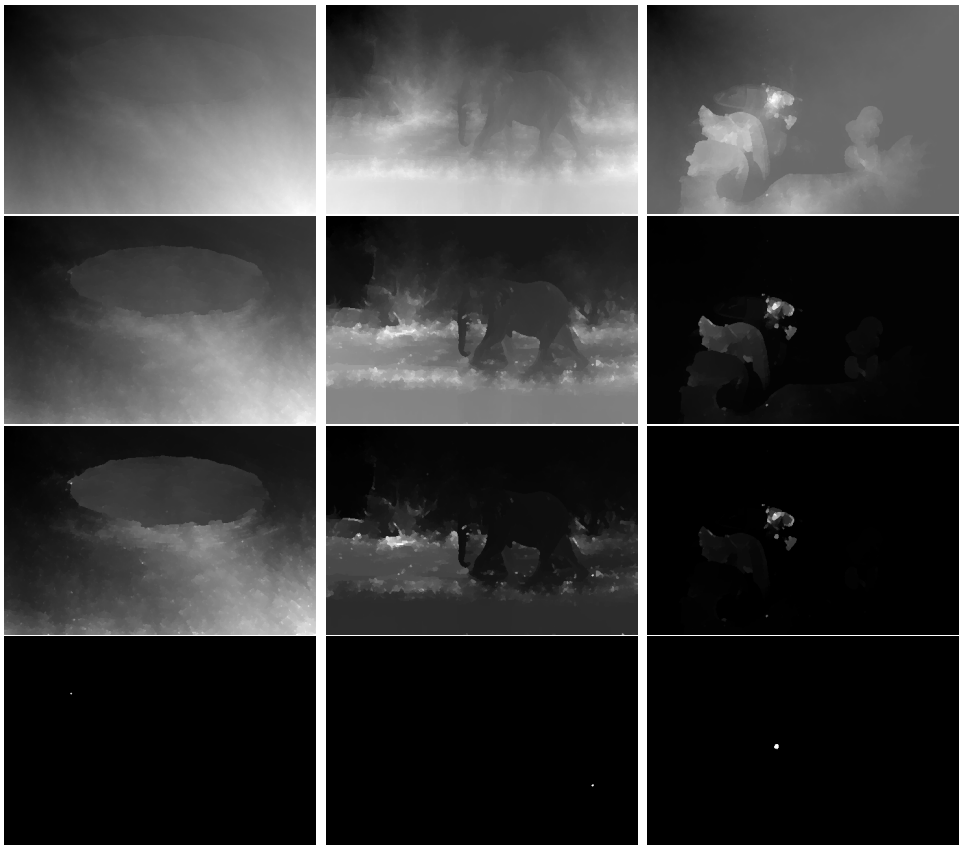


Figure 7.11: The effect of different SDM speeds on the arrival time images. SDM arrival time images with speeds $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} in rows 1–4, respectively. Original, gradient magnitude, and watershed overlaid images in Figure 7.10.



Figure 7.12: The effect of different hierarchical segmentation layers on FDM arrival time images. FDM results based on watershed partitioning (layer 1) in the first row; waterfall (layers 2–6) used for FDM in rows 2–6, respectively. Original, gradient magnitude, and watershed overlaid images in Figure 7.9.



Figure 7.13: The effect of different hierarchical segmentation layers on SDM arrival time images. SDM results based on watershed partitioning (layer 1) in the first row; waterfall (layers 2–6) used for SDM in rows 2–6, respectively. Original, gradient magnitude, and watershed overlaid images in Figure 7.9.

This trend is present but not visualised well for the f_{exp} and some f_{cub} images if the full arrival time range is mapped into the greyscale range.

Next, FDM and SDM qualitative results are compared for segmentations of varying level of granularity in Figures 7.12 and 7.13. Hierarchical segmentation based on $\mathrm{PRW}_{\mathrm{bal}}$ is performed for (the gradient magnitude images of) Lena, abdominal CT slice, and knee MRI slice images. The first, watershed layer, and following five waterfall layers of coarser partitionings are used as segmentation label images for running FDM and SDM. The other algorithmic parameters are set to the experimental default values for these tests.

If a partitioning contains large regions, then the corresponding FDM or SDM result will consist of the same large regions with uniform arrival times across. This is the consequence of the 'dashing' propagation inside partitioning regions. The last two rows of Figures 7.12 and 7.13 illustrate this trend: clearly separated objects are easily identifiable on them. The femur and the tibia in the knee MRI slice, for instance, can be easily extracted from these results. One important drawback of using a coarse high-layer partitioning for FDM or SDM is the automatic reproduction of the partitioning errors in the propagation result. If two pixels have been put in the same partitioning region by mistake, they will inevitably have the same arrival time in the FDM or SDM result. In case of using a fine low-layer partitioning, the risk of such mistakes is lower. At the same time, due to the larger number of boundary crossings in the propagation, when the partitioning regions are smaller, there is an enlarged variability in the arrival time values across the image. In addition, due to its local advancement nature, the propagation rate is similar in different directions. This makes the identification of large complex-shaped objects harder: careful seeding is required to achieve full propagation within the object boundaries before those boundaries are crossed.

7.6.2 Roughly segmented objects

Based on the brief discussion above on FDM parameters and their impact on qualitative results, we carry out a rough semi-automatic segmentation of a few objects in 2D images from BSDS500. Figure 7.14 illustrates some of our results.

The first step is the construction of the gradient magnitude images (second row) of the minimally smoothed greyscale images (first row). Then a hierarchical segmentation of the gradient magnitude image is constructed using the $\mathrm{PRW}_{\mathrm{bal}}$-based waterfall transform (see Section 6.4). Original greyscale images overlaid with layer 3 of the segmentation are shown in the third row of Figure 7.14. Finally, $\mathrm{FDM}_{\mathrm{async}}$ is executed on the gradient magnitude image once per object of interest. The used FDM parameter settings are as follows:

- a single seed position, located differently for each object of interest
- inverse cubic propagation speed $f_{cub}(p) = \frac{1}{image(p)^3+1}$
- partition labels of layer 3 (L3) of hierarchical segmentation produced with $\mathrm{PRW}_{\mathrm{bal}}$
- an arrival time threshold chosen experimentally for each object of interest

For each object, the resulting arrival time image is thresholded with an experimentally chosen value to separate the region, corresponding to the object. Thus, user input is expected for the choice of the single seed position and the arrival time threshold for each object of interest. The rest of the segmentation steps are automated and the implementation parameters are fixed.

The fourth and the fifth rows of Figure 7.14 contain the objects of interest segmented with the aforementioned simple procedure. In these images, the pixels inside the segmented region have the greyscale value of the original image. The pixels outside the segmented region are black.



Figure 7.14: Objects segmented with a rough semi-automatic procedure. Original greyscale images in the first row; gradient magnitude images in the second row; overlay of L3 partitioning in the third row. Segmented objects in the fourth and fifth rows.

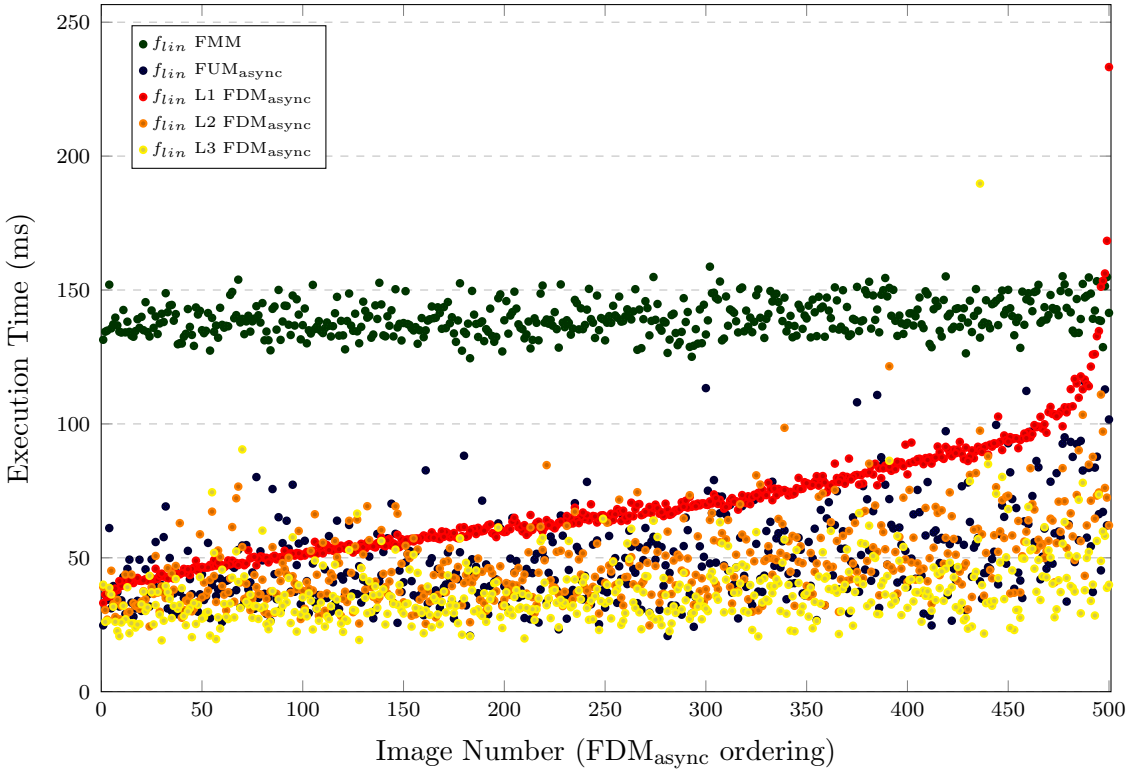
Given the nature of the objects of interest and the simple choice of the FDM parameters, the segmentation quality is very promising. All the considered objects take up a large portion of the image and consist of heterogeneous parts. This means that a more elaborate parameterisation is required to achieve best segmentation quality.

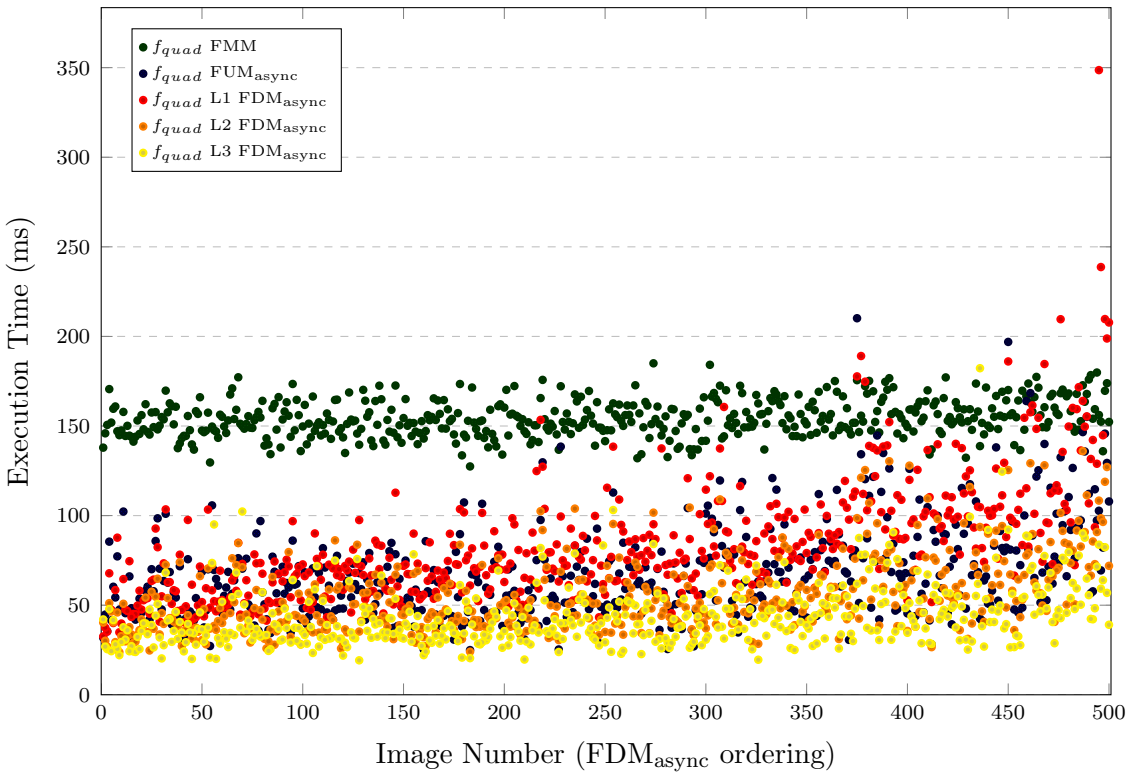
Despite the choice of only a single seed, the regions for the white bears, the sky, and the fighter aircraft are well-approximated. However, poor segmentation is observed for the firemen in the third image. The explanation for this is the corresponding gradient magnitude image. The gradient magnitude values are *not* consistently high along the expected region boundary. In addition to that, curves of high gradient magnitude are located inside the expected shape. For instance, the hat of the fireman on the left is clearly separated from the man's torso; the collar of the right fireman's jacket creates a noticeable line between the jacket and the man's head (refer to the second row of Figure 7.14). As a result, the front propagation initiated in a position inside the jacket region expands to leak into image background before the whole region for the fireman is covered (both for the left and the right firemen).

Thus, an important limitation to the use of simple FDM propagation for segmentation is the characteristics of the gradient magnitude image. The boundaries of the structures of interest need to be distinctively contrasted with little variation inside those regions. The possible solution in order to yield good results, otherwise, is to use a smaller arrival time threshold, reinitiate a new propagation with a seed in the uncovered area, and consider the union of the resulting regions as a final segmentation.

7.7 Comparison against traditional eikonal solvers

We perform quantitative and qualitative comparisons between the FDM algorithm and traditional eikonal solvers. We consider our implementation of the classic sequential $O(N \log N)$ fast marching method (FMM) [104, 108]. We also consider our own


f_{lin} FMM
f_{lin} FUM$_{\text{async}}$
f_{lin} L1 FDM$_{\text{async}}$
f_{lin} L2 FDM$_{\text{async}}$
f_{lin} L3 FDM$_{\text{async}}$
Execution Time (ms)
Image Number (FDM$_{\text{async}}$ ordering)


f_{quad} FMM
f_{quad} FUM$_{\text{async}}$
f_{quad} L1 FDM$_{\text{async}}$
f_{quad} L2 FDM$_{\text{async}}$
f_{quad} L3 FDM$_{\text{async}}$
Execution Time (ms)
Image Number (FDM$_{\text{async}}$ ordering)

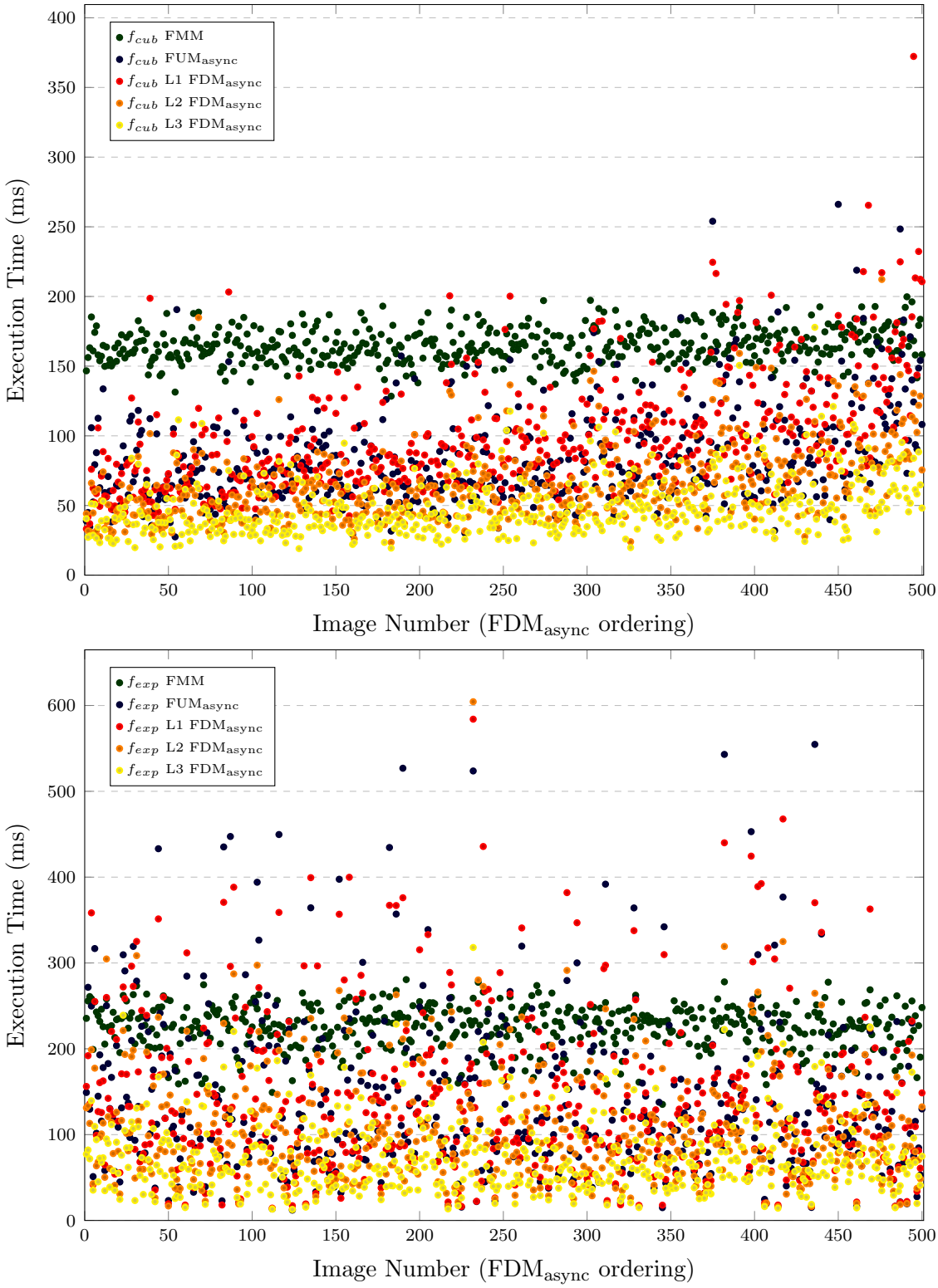


Figure 7.14: The BSDS500 execution times (in ms) of the fast marching and fast updating methods with different propagation speeds, compared against the fast dashing method with the same speeds using hierarchical partitioning layers 1–3.

block-asynchronous GPU implementation of an iterative eikonal solver—fast updating method (FUM). This procedure is based on FDM with two main differences. It excludes the 'dashing', i.e. lines 17–19 of Algorithm 9, and does not require a partitioning label image on input. Hence, it performs traditional front propagation solving the eikonal equation at all points on the grid.

Although our FUM procedure has some conceptual similarities with the fast iterative method [55, 56], there are also principal differences. Fast iterative method (FIM) maintains a list of active points, at each iteration updates only points in that list, updates and adds neighbouring points to the list when local updating converges at the given point. While our FUM considers the whole grid at each iteration, updates points in positive state, and sets the states of the given and neighbouring points for the next iteration.

There are three reasons for our preference of FUM over FIM as a GPU eikonal solver for the comparisons against FDM. First, our tests with the publicly available implementation of FIM [5] on small examples did not produce correct results. Second, that implementation of FIM does not calculate the speed values on the go but rather takes an array of speed values on input. The runtime comparisons between FDM and FIM would be unfair since FDM calculates the speeds in the GPU kernel based on the input image values.[5] Third, since all the implementation parameters and details of FDM and FUM are similar, except the 'dashing' and the use of the partitioning array, the comparison of FDM and FUM illustrates better the effect of the 'dashing' on execution time.

The graphs in Figure 7.14 depict the execution time comparison of $\text{FDM}_{\text{async}}$ against FMM and $\text{FUM}_{\text{async}}$ on BSDS500. Each of the four graphs corresponds to a different speed term: f_{lin}, f_{quad}, f_{cub}, and f_{exp} (see Section 7.4.2). The default settings are used for the seeds and the CUDA block size—the top left corner of the image and

[5]This is done for memory use considerations. Please refer to Section 7.2 for details.

Table 7.4: The sums of the runtimes (in sec) of FMM, $\text{FUM}_{\text{async}}$, and $\text{FDM}_{\text{async}}$ (L1–L3) over BSDS500 with the four speed terms considered.

	f_{lin}	f_{quad}	f_{cub}	f_{exp}
FMM	69.8034	77.1823	82.4704	112.4150
$\text{FUM}_{\text{async}}$	24.4510	33.9912	42.8415	69.7458
L1 $\text{FDM}_{\text{async}}$	34.5706	42.0169	49.6268	70.6048
L2 $\text{FDM}_{\text{async}}$	23.9072	27.7530	33.1706	52.3293
L3 $\text{FDM}_{\text{async}}$	18.7696	21.9927	24.5144	37.2653

16×16 2D blocks for $\text{FUM}_{\text{async}}$ and $\text{FDM}_{\text{async}}$. For each propagation speed, the first three layers of hierarchical segmentation, L1–L3, are considered as partitioning label images for $\text{FDM}_{\text{async}}$.

The execution time advantage of the GPU procedures—$\text{FUM}_{\text{async}}$ and $\text{FDM}_{\text{async}}$—over FMM breaks more frequently for the more complex propagation speeds. This is the effect of the iterative nature of the GPU procedures. The eikonal equation is solved only once per pixel in FMM, while both $\text{FUM}_{\text{async}}$ and $\text{FDM}_{\text{async}}$ may solve the eikonal equation multiple times per pixel. In case of the more complex speed terms, the calculation of the speed value at a given pixel becomes more expensive. In addition, the expected number of iterative updates increases as a result of the more complex propagation patterns.

Table 7.4 lists the corresponding sums of runtimes for the BSDS500 dataset. The comparison of these sums indicates that the GPU implementations maintain an overall execution time superiority over the CPU sequential FMM for the different speed terms, despite the occasional slower performance of the GPU procedures on specific images.

Both Figure 7.14 and Table 7.4 show how the choice of the partitioning label image affects the $\text{FDM}_{\text{async}}$ execution times and their comparison against $\text{FUM}_{\text{async}}$. While L1 $\text{FDM}_{\text{async}}$ is generally slower than $\text{FUM}_{\text{async}}$, L2 $\text{FDM}_{\text{async}}$ and L3 $\text{FDM}_{\text{async}}$ execute faster than $\text{FUM}_{\text{async}}$. Based on Table 7.4, the runtime advantage of L3 $\text{FDM}_{\text{async}}$ over $\text{FUM}_{\text{async}}$ is more significant for speeds f_{cub} and f_{exp}.

Table 7.5: The execution times (in sec) of FMM, $\text{FUM}_{\text{async}}$, and $\text{FDM}_{\text{async}}$ with default parameter settings on the 3D test images. The smallest runtime for each 3D test image is highlighted.

	4,000×4,000×8	500×500×512	320×320×1,250
FMM	**294.6566**	407.3912	394.5481
$\text{FUM}_{\text{async}}$	700.5802	265.5912	207.8292
$\text{FDM}_{\text{async}}$	428.9104	**160.9361**	**112.3529**

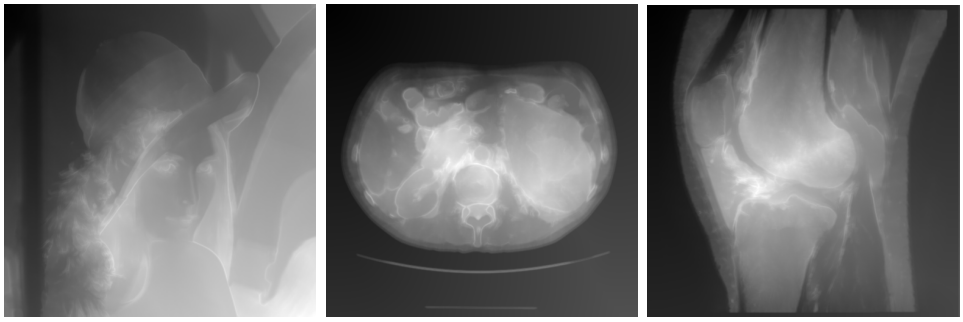


Figure 7.15: FMM arrival time images for Lena, abdominal CT slice, and knee MRI slice sample images using default parameters. Original and gradient magnitude in Figure 7.9.

A further runtime comparison is performed for the 3D test images and results are provided in Table 7.5. All the algorithmic and implementation parameters are set to their defaults, i.e. the speed is f_{lin} and the partitioning label image is L1 for $\text{FDM}_{\text{async}}$. On the 500×500×512 and 320×320×1,250 test images, $\text{FDM}_{\text{async}}$ is much faster than FMM and $\text{FUM}_{\text{async}}$. On the 4,000×4,000×8 image, $\text{FDM}_{\text{async}}$ runs for around 7 minutes, which is still faster than $\text{FUM}_{\text{async}}$ with more than 11.6 minutes. But the best performance on this image is achieved by FMM with just below 5 minutes.

The equivalence of the FMM and $\text{FUM}_{\text{async}}$ procedures is established with a straightforward comparison of the resulting arrival time arrays. These have proved identical throughout these experiments. However, as intended, the $\text{FDM}_{\text{async}}$ results appear different from FMM and $\text{FUM}_{\text{async}}$ due to the 'dashing' propagation inside partitioning regions. In Figures 7.15 and 7.16 we present the arrival time images produced by FMM or, equivalently, by $\text{FUM}_{\text{async}}$, corresponding to the examples discussed in Section 7.6.1. These examples show that FMM results tend to assign high



Figure 7.16: The effect of different FMM speeds on the arrival time images. FMM arrival time images with speeds $f_{lin}, f_{quad}, f_{cub}$, and f_{exp} in rows 1–4, respectively. Original and gradient magnitude in Figure 7.10.



Figure 7.17: Objects segmented with the rough semi-automatic procedure, replacing FDM with FMM. Original greyscale images and gradient magnitude images in Figure 7.14.

arrival time values to region boundaries and there is higher arrival time variability inside regions.

Finally, we apply the same rough semi-automatic segmentation procedure, discussed in Section 7.6.2, replacing FDM with FMM. We also change the experimental arrival time thresholds for the different objects of interest to stop the front propagation just before significant false positive regions are included. The results are presented in Figure 7.17. All six results show noticeable undersegmentation (false negatives), especially near region boundaries. At the same time, increasing the threshold only introduces large false positive regions without solving the problem of false negatives.

7.8 Summary

We introduced and discussed a new hybrid procedure for modelling surface propagation in a partitioned image. Our parallel fast dashing method solves the eikonal equation for propagation across region boundaries and promotes constant arrival times inside partitioning regions. We considered GPU synchronous and block-asynchronous implementations of the fast dashing method and analysed implementation details. An alternative simplified approach of simple dashing method for propagation in a parti-

tioned image was also introduced, which replaces the solution of the eikonal equation with advancement from a single neighbour across region boundaries.

We presented experimental results to reveal how implementation type, algorithmic and implementation parameters affect the runtimes of both novel algorithms on 2D and large 3D images. In addition, we discussed the execution time improvement of SDM over FDM. Further qualitative results presented the effect of the different parameters on the outputs of the two algorithms. A few segmentation attempts, with a rough segmentation procedure that relies on PRW and FDM, showed promising results and proved the segmentation potential of our methods.

A final set of tests was performed to compare FDM against traditional eikonal solvers. This was done in order to show that FDM holds both quantitative and qualitative advantages over the traditional methods and can successfully supersede those in segmentation tasks.

The next chapter presents some results of using the FDM method for medical image segmentation, discusses the method's limitations, and gives recommendations for its use in practice.

Chapter 8

Results

8.1 Experimental setting

In order to illustrate that simple, accurate, and efficient segmentation procedures can be devised based on the proposed algorithms, we describe a real-life experiment and discuss its outcome. With this experiment, we show that

- simple and generic segmentation approaches can be proposed based on the combination of PRW and FDM;
- the GPU implementation makes the processing of large 3D medical image data very efficient, demonstrating interactive runtimes, conceivably usable in clinics;
- high accuracy segmentation can be achieved without any data-specific processing or prior knowledge;
- boundary overlap metrics greatly assist in the validation of segmentation results.

Based on the brief discussion of roughly segmented objects in 2D digital images from Section 7.6.2, the following simple procedure is proposed for medical 3D image segmentation. The procedure consists of the following steps:

1. read a sequence of DICOM images and consider them as a single 3D volume[1];
2. apply minimal 3D smoothing with Gaussian blur ($\sigma = 1$, a single iteration);
3. construct the corresponding gradient magnitude 3D image volume;
4. repeatedly apply the PRW_{bal}-based GPU waterfall transform to construct the L3 of hierarchical partitioning;
5. *experimentally* choose a *single* FDM seed point inside the anatomical structure of interest (this involves a single click from a trained user such as a clinician or a researcher with minimal anatomical training);
6. apply $\text{FDM}_{\text{async}}$ using the L3 partitioning and f_{cub} speed to construct the arrival time image with 0–65535 value range;
7. threshold the arrival time image with an *experimentally* chosen threshold value to produce a labelled region.

Thus, the $\text{FDM}_{\text{async}}$ application relies on the following algorithmic parameters:

- a single 3D seed position, chosen experimentally by the user;
- inverse cubic propagation speed $f_{cub}(p) = \frac{1}{image(p)^3+1}$;
- partition labels of L3 of hierarchical segmentation produced with PRW_{bal};
- an arrival time threshold, chosen experimentally by the user[2].

The proposed segmentation procedure is semi-automatic; user input is expected to seed the FDM propagation and to threshold the resulting arrival time image. This is intentional to keep the generality of the procedure and make it applicable to different types of data, non-restrictive to specific modalities, anatomical structures,

[1]Windowing is not applied for this data. The full greyscale range of the DICOM slices is linearly mapped onto the 0–255 greyscale range.

[2]In our experiments, the thresholds used were in the range 50–1200 for the different images.

image volume shapes and sizes, etc. However, in a clinical setting these parameters could be made specific to a frequently used scan sequence for a specific body part. In addition, the current procedure purposefully excludes any pre- or post-processing techniques (elaborate smoothing or morphological closing, for instance) that could improve its final results.

The main objective of this segmentation procedure is not so much to achieve top performance for a narrow and specific segmentation task. On the contrary, we want to show that promising segmentation results can be attained for a randomly chosen segmentation task, while keeping the procedure simple and efficient. We want to estimate how good the performance of the PRW–FDM combination can be if used without additional processing. **The target of this experiment is the validation of our concept of front propagation in a partitioned domain**. When segmentation for a specific task is the sole focus, this segmentation procedure can be further refined and fully automated for improved target performance.

Using this segmentation procedure, we illustrate the segmentation of femur and tibia bones in nine sagittal knee MRI 3D scans. The in-slice resolution is 528×528 for seven of the sequences and 512×512 for another two. The number of slices is 11 for one sequence and ranges between 26 and 32 for the rest. Thus, most of the considered 3D image volumes contain approximately 7–9 megavoxels.

Per 3D scan, there are two structures of interest—the femur (fem1–fem9) and the tibia (tib1–tib9). For each of the 18 labelled regions, the first FDM result is reported. This means that the seed position has been chosen only once and there have *not* been repeated segmentation attempts (i.e. change of seed position or use of multiple seeds) to improve evaluation scores. In a real-world setting, such mechanisms can be easily utilised to achieve improved performance and segmentation accuracy.

Per structure of interest, two fixed threshold values have been selected based on visual assessment of the resulting regions in 3D. To arrive at satisfactory appearance

of the labelled regions, 2–5 thresholds have been considered. Note, that this process does *not* involve the knowledge of the ground truth segmentations and is fully based on the visual perception of the greyscale MRI scans by the user (the author, in this case), who has only minimal anatomical training. In the initial validation process, the two labelled regions per femur or tibia have been evaluated against the ground truth and only the one with 'better scores' is presented in our quantitative results in the next section. The choice of the 'better scores' is based on a majority vote among the considered evaluation methods (see Section 3.6 and Chapter 5 for details on the evaluation methods).

In some cases, although produced with different thresholds, the two labelled regions are fully identical. This means that there has been no front propagation between the two time points used. This also means that *all* the evaluation scores are also fully identical for the two thresholded regions. In other cases, the absolute difference in initial scores between the two thresholded femur regions reaches up to 7.31% DSC, 3.26 mm HD, or 8.49% SBD. For tibia, the corresponding maximal differences of scores are 3.62% DSC, 8.74 mm HD, or 6.99% SBD.

The next section describes the problems of the available ground truth segmentations and how we process the segmentation masks to make a fairer comparison. Henceforth, we call 'initial' scores those obtained by the direct comparison of our machine segmentation results with the available ground truth segmentations; and 'updated' scores those obtained after processing to compare like with like.

8.2 Performance

Tables 8.1 and 8.3 present the initial evaluation scores of the nine femur and nine tibia segmentations described in the previous section using different overlap-based, size-based, boundary-distance-based, and boundary-overlap-based metrics from Chap-

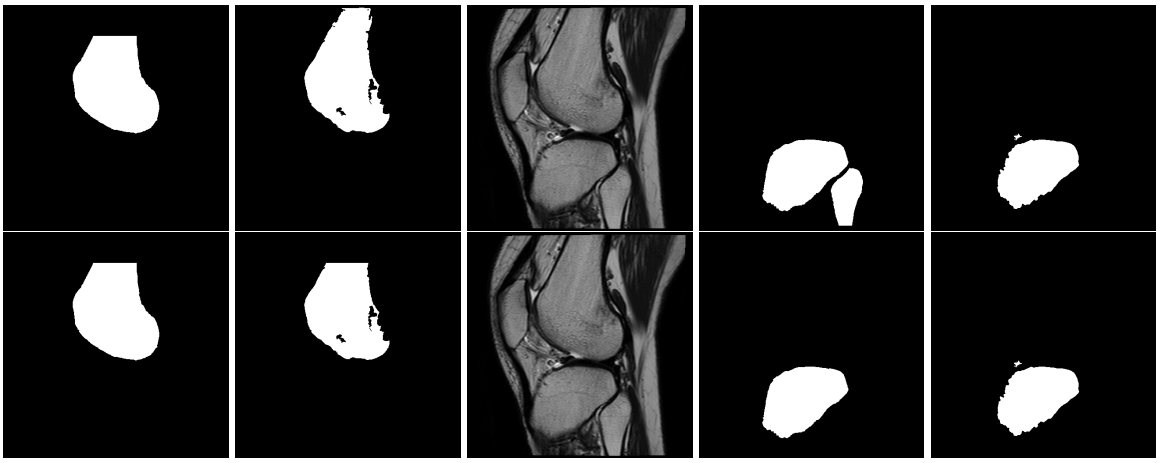


Figure 8.1: The problems of the ground truth regions used (seen in columns 1 and 4), illustrated for a single knee MRI slice. Original greyscale slice is in the middle column. A slice of the ground truth and machine segmentation masks for fem2 (tib2) in first and second (fourth and fifth) columns. First row shows initial results, second row—updated results.

ter 5.

Four out of the nine MRI volumes are illustrated in a slice-by-slice manner in Appendix A along with the labelled ground truth and machine segmentation masks for femur and tibia. These illustrations show, in particular, how the anisotropic spacing (2.5 mm distance between slices versus 0.3 mm intra-slice, for instance) of the 3D images affects the front propagation accuracy in different directions. Since the single FDM seed position is commonly located in the middle slice of the 3D volume, the middle slices are generally better segmented in comparison to the first few and the last few slices. Similar trends of better segmentation quality in middle slices in comparison to the end slices have been reported in the existing literature before [67] even for slice-by-slice 2D segmentation approaches.

Due to a glitch in the expert-provided ground truth where top or bottom rows in the image are excluded from the masks, the scores reported in Tables 8.1 and 8.3 are promising but not particularly high and cannot immediately compete with state-of-the-art segmentation procedures [83] designed specifically for the segmentation of the femur and tibia bones. However, these lower scores *do not only* reflect machine

segmentation errors. The different characteristics of the ground truths used also have an important impact on these scores. Figure 8.1 shows a single slice from the knee MRI dataset, along with the ground truth and machine segmentation masks for femur and tibia.

As can be seen from the illustrated MRI slices in Appendix A and column 1 of Figure 8.1, the ground truth segmentations of femurs appear 'cropped' at the top of the slices. This means that there are no points in the top 55–100 rows in any slice included in the femur ground truth regions. The number of such rows is fixed across slices in the same 3D volume but changes across different 3D volumes. All nine femur ground truths are affected by this problem.

A similar 'cropped' appearance is seen for the ground truths of tib1, tib2, and tib7, when the last 11–79 rows are excluded from the ground truth regions. Furthermore, the ground truths of tib1, tib2, and tib3 include the regions corresponding to the fibula bone in addition to the tibia. The femur and tibia ground truths in columns 1 and 4 of Figure 8.1 showcase all three mentioned problems. This also illustrates how hard it is for a human to produce consistently quantifiable contours (see Section 3.5.4).

Table 8.1: Match of our segmentation against gold standards per femur with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm.

Femur	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD	SBJ	SBTP	SBTN	SBP
fem1	86.97	76.94	85.74	99.17	88.23	2.83	23.64	1.82	71.81	63.08	64.52	58.18	86.11
fem2	87.71	78.11	86.76	99.15	88.67	2.16	20.68	1.64	73.81	66.23	68.50	67.00	84.67
fem3	86.30	75.90	80.13	99.67	93.49	14.29	15.33	0.97	81.71	76.75	77.74	40.11	89.67
fem4	87.34	77.52	88.18	99.04	86.51	1.93	23.33	1.81	72.65	64.30	67.06	64.05	84.35
fem5	77.07	62.69	71.35	99.00	83.78	14.84	19.93	2.71	57.36	49.36	53.46	57.00	68.14
fem6	85.44	74.58	85.42	99.06	85.46	0.06	23.63	1.86	71.52	63.81	66.53	63.90	82.02
fem7	78.03	63.97	78.34	98.39	77.72	0.79	31.47	3.42	61.95	55.89	58.49	61.40	69.71
fem8	86.89	76.81	91.26	98.92	82.91	10.08	29.39	2.13	75.78	69.07	72.22	68.00	83.60
fem9	87.67	78.05	89.11	99.15	86.28	3.28	27.98	1.74	75.67	68.56	71.08	68.87	85.09

Table 8.2: Match of our updated ('cropped') segmentation against gold standards per femur with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm.

Femur	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD	SBJ	SBTP	SBTN	SBP
fem1_{upd}	91.79	84.83	85.74	99.92	98.76	13.19	11.88	0.96	78.04	68.65	70.02	60.97	93.63
fem2_{upd}	91.89	85.00	86.76	99.84	97.67	11.17	12.35	1.03	79.01	71.02	73.21	70.08	90.68
fem3_{upd}	86.30	75.90	80.13	99.67	93.49	14.29	15.33	0.97	81.71	76.75	77.74	40.11	89.67
fem4_{upd}	92.11	85.38	88.18	99.77	96.41	8.54	8.75	1.00	78.47	69.56	72.33	67.09	91.16
fem5_{upd}	80.31	67.10	71.35	99.54	91.85	22.31	19.26	2.26	60.90	52.50	56.68	58.98	72.39
fem6_{upd}	89.95	81.74	85.42	99.71	94.99	10.08	11.39	1.09	76.57	68.32	71.23	66.22	87.82
fem7_{upd}	85.17	74.17	78.34	99.60	93.30	16.04	16.98	1.71	70.51	63.73	66.45	66.58	79.40
fem8_{upd}	94.23	89.08	91.26	99.86	97.39	6.29	10.20	0.68	84.49	77.14	80.39	72.79	93.30
fem9_{upd}	93.30	87.45	89.11	99.89	97.92	9.00	10.00	0.73	82.43	74.79	77.31	72.66	92.74
mean	89.45	81.18	84.03	99.76	95.75	12.32	12.91	1.16	76.90	69.16	71.71	63.94	87.86

Table 8.3: Match of our segmentation against gold standards per tibia with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm.

Tibia	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD	SBJ	SBTP	SBTN	SBP
tib1	84.56	73.25	76.85	99.75	94.00	18.25	29.17	2.47	64.17	55.86	60.03	67.53	74.82
tib2	88.61	79.56	80.91	99.88	97.94	17.39	30.25	2.25	71.57	63.42	64.49	72.82	84.37
tib3	74.94	59.93	61.81	99.83	95.17	35.05	31.87	2.87	69.61	65.66	68.09	52.44	73.98
tib4	88.49	79.35	84.96	99.63	92.33	7.98	15.34	1.36	67.23	58.42	63.84	59.70	78.05
tib5	91.33	84.05	84.56	99.97	99.29	14.84	11.00	0.88	75.57	66.74	67.76	70.45	90.36
tib6	86.20	75.74	78.41	99.83	95.70	18.06	18.71	1.45	71.68	63.31	65.19	64.73	84.67
tib7	80.49	67.35	82.14	99.02	78.90	4.11	23.10	2.43	62.03	54.12	55.75	60.94	75.11
tib8	90.64	82.88	88.78	99.68	92.58	4.10	11.53	1.04	74.27	65.92	69.13	65.06	85.31
tib9	93.42	87.65	89.64	99.90	97.53	8.09	11.45	0.61	80.76	72.61	75.12	72.24	92.06

Table 8.4: Match of our updated ('cropped') segmentation against (modified to exclude fibula) gold standards per tibia with several similarity or difference metrics; tib1, tib2, tib3, tib7 affected. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm.

Tibia	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD	SBJ	SBTP	SBTN	SBP
tib1_{upd}	90.02	81.85	85.00	99.83	95.66	11.14	12.41	1.04	73.16	63.68	68.51	64.12	85.24
tib2_{upd}	93.35	87.53	88.13	99.96	99.23	11.19	12.17	0.87	80.26	71.09	72.45	70.38	94.51
tib3_{upd}	90.07	81.93	85.49	99.83	95.17	10.17	13.55	0.60	83.17	78.45	81.36	43.81	88.39
tib4	88.49	79.35	84.96	99.63	92.33	7.98	15.34	1.36	67.23	58.42	63.84	59.70	78.05
tib5	91.33	84.05	84.56	99.97	99.29	14.84	11.00	0.88	75.57	66.74	67.76	70.45	90.36
tib6	86.20	75.74	78.41	99.83	95.70	18.06	18.71	1.45	71.68	63.31	65.19	64.73	84.67
tib7_{upd}	89.76	81.42	82.14	99.96	98.94	16.98	9.19	1.09	71.42	62.61	63.98	66.69	86.51
tib8	90.64	82.88	88.78	99.68	92.58	4.10	11.53	1.04	74.27	65.92	69.13	65.06	85.31
tib9	93.42	87.65	89.64	99.90	97.53	8.09	11.45	0.61	80.76	72.61	75.12	72.24	92.06
mean	90.36	82.49	85.23	99.84	96.27	11.39	12.82	0.99	75.28	66.98	69.70	64.13	87.23

Tables 8.2 and 8.4 show the updated scores of our femur and tibia segmentations when a fairer comparison is attempted. In order to compare like with like, we have 'cropped' the machine segmented regions to match the corresponding ground truths. In the case of each femur, the number of 'cropped' rows in the ground truth is identified and the same number of top rows is excluded from the machine segmentation of that femur. In the case of the tibias, the machine segmentations of tib1, tib2, and tib7 are also 'cropped' for a fixed number of rows at the bottom of each slice, in accordance to the respective ground truths. Finally, we have processed every single slice of the ground truth masks for tib1, tib2, and tib3 to exclude the regions that belong to the fibula.

The updated femur segmentation evaluation scores are in the 80.31–94.23% DSC, 8.75–19.26 mm HD, and 60.9–84.49% SBD ranges. Before the update the corresponding score ranges were only 77.07–87.71% DSC, 15.33–31.47 mm HD, and 57.36–81.71% SBD. For tibia, the worst scores upgrade from 74.94% DSC, 31.87 mm HD, and 62.03% SBD to 86.2% DSC, 18.71 mm HD, and 67.23% SBD when we update the segmented regions. These examples crucially showcase how important the quality of the ground truth, produced with manual segmentation, is and how much it affects evaluation scores.

Despite the large sizes of the 3D images reaching almost 9 megavoxels, the segmentation of a femur or a tibia with the described steps 1–7 always takes less than a minute. In comparison, the segmentation of an abdominal organ with the sequential segmentation procedure from Chapter 4 in an image of similar size would take at least 15–20 minutes. Marčan et al. [82] report approximately 30 minutes of execution time for the segmentation of femur and tibia from a 3D knee MRI volume of similar size using a sequential IPF- and region-growing-based algorithm.

8.3 The state-of-the-art context

In addition to the segmentation of femur and tibia from knee 3D MRI scans discussed above, we also consider the segmentation of femur from knee 2D MRI slices. For that, we choose another dataset: ten individual slices from ten 3D image volumes in the training set of the SKI10 grand challenge [51]. Our choice of the individual slices fully matches that in [96] (full details of the scan and slice numbers and the windowing parameters can be found in Table A.2 in [96]).

This experiment is relevant for multiple reasons. First, the new data displays more variability and lower contrast appearance of the structures of interest (SKI10 is designed to present difficulties for many algorithms). Second, it allows us to see how the performance of our segmentation procedure improves when switching from 3D to 2D processing (especially important when dealing with anisotropic data). Next, windowing is also utilised to diminish the effect of low contrast boundaries between the structure of interest and surrounding tissues. Finally, the results confirm the potential of PRW-FDM-based segmentation procedures to achieve fast and accurate performance on benchmarking data, like the SKI10 grand challenge.

The experimental setting and the segmentation procedure for these knee 2D MRI slices is very similar to what was described in Section 8.1. However, there are also a few key differences:

- instead of a 3D volume, only a single slice is considered, and all the processing steps 1–7 are applied in 2D;
- in step 1, a 2D slice is read and *windowed* using the windowing parameters chosen for that slice and reported in Table A.2 by Pirovano [96];
- instead of two, only one threshold of the FDM arrival time image is chosen for numeric validation.

Table 8.5: Match of our segmentation against gold standards per femur slice from SKI10 with several similarity or difference metrics. Metrics DSC–RVD, SBD–SBP are measured in %; HD and ASSD—in mm.

2DFemur	DSC	JSC	TPVF	TNVF	Prec	RVD	HD	ASSD	SBD	SBJ	SBTP	SBTN	SBP
M	97.01	94.20	98.23	99.05	95.82	2.51	8.81	0.78	76.06	69.88	78.83	53.97	77.32
N	98.11	96.29	97.94	99.50	98.28	0.35	5.86	0.52	79.50	71.64	79.50	49.80	84.22
P	93.90	88.51	91.51	99.11	96.43	5.10	12.11	1.33	69.28	61.63	72.80	43.33	71.00
Q	95.73	91.81	97.64	98.07	93.89	3.99	4.83	1.08	66.22	59.47	68.84	38.16	68.44
R	97.04	94.25	96.06	99.47	98.04	2.03	7.18	0.74	73.37	65.26	71.29	53.32	81.66
S	95.86	92.05	96.00	98.71	95.72	0.29	10.55	1.42	71.97	66.33	75.62	50.82	72.68
T	99.18	98.37	99.46	99.68	98.90	0.57	2.34	0.19	86.40	79.81	89.51	57.71	87.36
U	98.42	96.89	97.13	99.93	99.75	2.63	2.76	0.44	67.53	58.69	59.98	47.44	86.10
V	97.48	95.09	98.59	99.09	96.39	2.28	11.72	0.80	75.16	67.23	75.60	51.29	80.70
W	98.16	96.39	98.73	99.32	97.61	1.15	3.60	0.50	80.30	74.12	84.75	51.41	79.69
mean	97.09	94.39	97.13	99.19	97.08	2.09	6.98	0.78	74.58	67.41	75.67	49.73	78.92

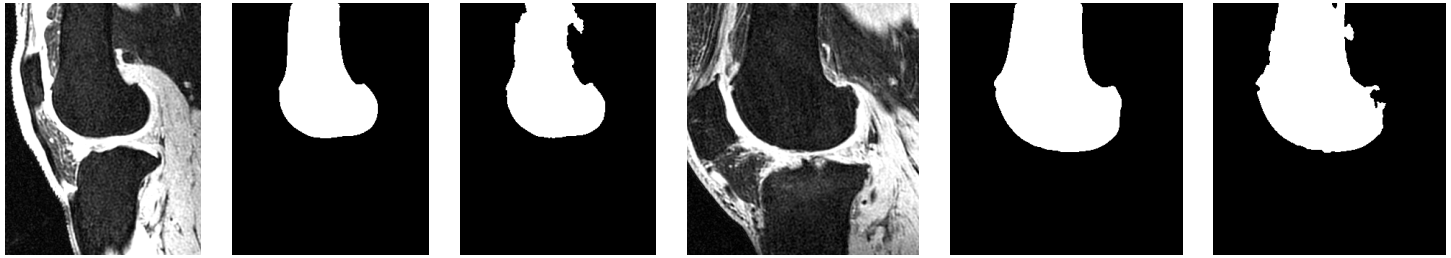


Figure 8.2: Single-slice femur segmentation results for M and N image slices from SKI10. Original greyscale slices are in the first and fourth columns. The ground truth (machine segmentation) masks in second and fifth (third and sixth) columns.

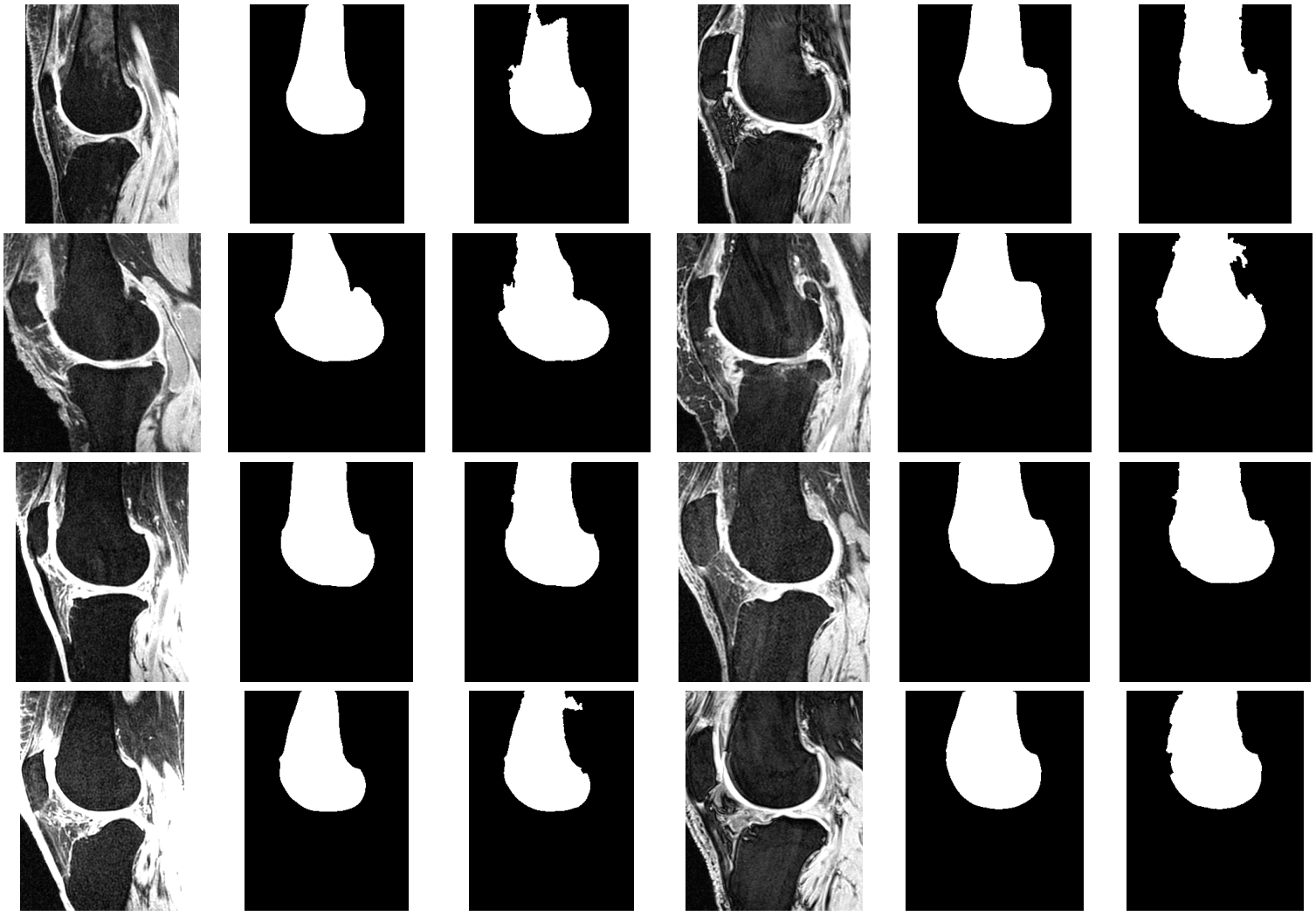


Figure 8.2: Single-slice femur segmentation results for P–W image slices from SKI10. Original greyscale slices are in the first and fourth columns. The ground truth (machine segmentation) masks in second and fifth (third and sixth) columns.

The evaluation scores of the ten single-slice femur segmentations (M–W) and their averages are provided in Table 8.5. The slices and the corresponding ground truth and machine segmentation masks can be found in Figure 8.2. The average DSC score is 97.09%; HD is 6.98 mm; SBD is 74.58%. While the very high DSC scores indicate that the overall femur regions are approximated well, the significantly lower SBD scores show that most boundary curves are dislocated by a few pixels. This is visible in the comparison of the pairs of masks in Figure 8.2. The boundary-distance-based metrics indicate the extent of the dislocations. Our results for femur slices M–W outperform those reported for the same slices in [96] (comparison based on DSC and TPVF).

The official SKI10 challenge scores participating teams based on the segmentation results for femur bone, tibia bone, femur cartilage, and tibia cartilage. The bones are scored with the ASSD and Root-Mean-Square Symmetric Surface Distance (RMSD)[3] evaluation metrics [51] using resampled segmentations with doubled number of slices (for an approximately isotropic spacing of 0.4×0.4×0.5 mm) and 18-neighbourhood to identify boundary points.

Table 8.6 presents the average femur bone segmentation scores over the 50 test 3D images of the overall top ten performing submissions to SKI10 [13] as of December 2017. It also includes the reference inter-operator variability used in the scoring system to rank the submissions.

In order to assess the potential of our methods to be used for the development of application-specific segmentation procedures for participation in a challenge like SKI10, we compare the average ASSD score from our results to the ones reported in Table 8.6. Our average score of 0.78 mm ASSD is only just below a couple of scores in Table 8.6 (0.77 mm ASSD for rank 7 and 0.76 mm ASSD for rank 9). In comparison to the reference inter-operator variability, our score is around 1.73 times the reference

[3]RMSD is defined similarly to ASSD, only replacing the linear averaging with root-mean-square averaging.

Table 8.6: Average femur bone test segmentation scores of the overall top ten performing submissions to SKI10 as of December 2017 and the reference inter-operator variability used in the scoring system.

Rank	Team	ASSD (mm)	RMSD (mm)
1	ZIB	0.43	0.72
2	AMC_MIRL	0.41	0.71
3	Imorphics	0.42	0.74
4	ZIB	0.57	0.88
5	UPMC_IBML	0.63	1.05
6	SNU_SPL	0.70	1.09
7	Biomediq	0.77	1.43
8	UPMC_IBML	0.66	1.06
9	shan_unc	0.76	1.22
10	AMC_MIRL	0.67	1.13
–	Reference	0.45	0.77

of 0.45 mm ASSD, which we believe is a promising start for future experiments.

A direct comparison is unfair, for a few reasons. We have segmented ten 2D slices from ten images in the training set of the challenge (ground truth is available for download only for training data). The top-submission scores are for 3D segmentations of the 50 test images. We rely on a 2D 8-neighbourhood; they use a 3D 18-neighbourhood and resampled data. Also, we evaluate raw segmentation results—no segmentation improvement mechanisms are employed in addition to the PRW–FDM combination. Section 9.2 briefly discusses how data-specific automatic segmentation procedures can be devised on the basis of this combination.

In each of the ten cases, the runtime of the segmentation procedure was less than a second. This estimate obviously excludes the time taken by the user to choose a single seed point and to threshold the arrival time image. If a convenient graphical user interface (GUI) is provided, this user input can easily take less than a minute, bringing the overall time required for the segmentation of a single anatomical structure (to a certain accuracy degree) from a 2D slice or a 3D volume to around 1–2 minutes.

SKI10 is an ongoing challenge, thus a comparative discussion of the different

submissions has not be published yet. Hence, we do not have much information about the execution times of the top performing submissions. From the brief descriptions available on the challenge webpage [13], most of the submissions rely on learning techniques. The few reported execution times are in the range of 15–40 minutes.

8.4 Discussion

Visual assessment of the results provided in Figures 8.1, 8.2, and Appendix A allows us to separate the main inaccuracy types for this segmentation procedure, and hence FDM.

The most salient problem in 3D propagation is the effect of the anisotropic spacing of the input data on the propagation rates in different directions. Very often the machine segmentation stretches across a few extra slices or shows inaccurate masks in end slices in comparison to the ground truth. One possible solution to this problem is the more elaborate seeding of the FDM propagation to include a few seeds spread across different slices. Also, the produced masks can be post-processed by a user to remove any labelled regions in the extra slices. Other alternative approaches are also possible, such as performing a seeded 2D FDM propagation in each of the relevant slices.

The variable level of contrast along the boundary of the structure of interest causes 'leaks' in the propagation front. This is the trend of including extra regions (adjacent false positives) in the labelled mask before covering the whole region of interest (i.e. leaving false negatives inside). Some of the slices in Figure 8.2 include similar cases. When this happens, it is impossible to choose an optimal arrival time threshold. One solution is to stop the propagation before the 'leakage' happens, seed a new propagation inside the undersegmented (false negative) area, and consider the union of the original and the newly labelled regions as the final result. The great

runtime efficiency of the proposed algorithms allows for fast interactive segmentation when multiple FDM propagations are initialised and performed.

Another problem is the appearance of 'holes' in the mask slices, when a non-labelled region appears surrounded by labelled region. This can be the effect of local inhomogeneity in the input data drastically slowing down the front propagation into the inhomogeneous region. Alternatively, this can be the visual effect of projecting a 3D region without 'holes' onto 2D slices. In any case, better input smoothing techniques or multiply seeding can improve the segmentation outcome.

Lastly, based on input data, gradient magnitude image quality, and propagation patterns, the boundaries of thresholded FDM results can appear unsmooth and uneven. Such cases are flagged by our boundary-overlap-based evaluation metrics. The segmentation results can be automatically post-processed with primitive morphological operations, like morphological closing, to smooth out the mask boundaries.

A final note relates to the comparison between the results in Tables 8.2, 8.4, and 8.5. Although, if compared directly, these scores are indicative of the relative quality of the segmentation results, the experimental differences need to be kept in mind. Tables 8.2 and 8.4 correspond to 3D segmentations of data with 26–32 slices of resolution 528×528 or 512×512 and typical voxel spacing of 0.3 mm by 0.3 mm by 2.5 mm. By contrast, Table 8.5 corresponds to 2D single-slice data with approximate resolution of 270×350 and pixel spacing of 0.4 mm by 0.4 mm (the slice thickness of 1 mm is irrelevant to 2D experiments).

Similarly, our results for the SKI10 data cannot be immediately compared against the top scores reported on the challenge webpage: those are 3D score averages across the whole dataset and rely on data resampling before the score calculation. We believe that our current results are a promising starting point for future large-scale experiments on benchmarking data.

Chapter 9

Conclusions

9.1 Concluding remarks

This thesis has presented novel results concerning four different aspects of medical image segmentation.

We start with describing an automated sequential 3D procedure for the segmentation of kidneys from abdominal CT sequences. This proposed procedure combines the hierarchical partitioning of 3D images with front propagation to label kidney regions. For the hierarchical partitioning we use the IPF data structure by Golodetz [43]. We rely on FMM [107] for front propagation. While the validation of this segmentation procedure shows valuable results, it also reveals shortcomings of both the segmentation and validation procedures.

Our concern about the lack of availability of reliable validation mechanisms for image segmentation leads to our survey of existing evaluation approaches and methods. We propose a family of hybrid evaluation metrics called boundary overlap metrics for the comparison of segmented masks against ground truth regions. Although we do not introduce any novel types of validation, our contribution makes the most common evaluation form of comparison against ground truth more informative and

useful. Our experimental results on the comparison of boundary overlap metrics against other well-known evaluation metrics have demonstrated that there is a niche for our metrics.

The main shortcomings of the kidney segmentation procedure are the execution speed and the local errors in the front propagation results. In order to significantly reduce the runtime of hierarchical partitioning of large 3D images we propose the path reducing watershed algorithm and PRW-based waterfall algorithm for parallel GPU implementation. In extensive experiments with hundreds of 2D images and large 3D images we have achieved runtimes of a few sec, and successfully overcome the state-of-the-art GPU watershed algorithms.

We develop the FDM algorithm, and the simplified SDM, to model front propagation in partitioned images, thus enabling the propagation step to utilise pixel grouping information available after partitioning. The tests have analysed the effects of the different parameters on the algorithm performance, revealed fast execution patterns, and proved runtime and accuracy advantages over traditional eikonal solvers.

Combining the proposed PRW and FDM algorithms for GPU implementation, we have segmented femur and tibia from knee MRI sequences in a simple semi-automatic way. The results have proved very efficient with runtimes under a minute for image volumes of 7–9 megavoxels. The quantitative outcome is highly promising given the very basic and straightforward nature of the human operator input (single seed point and simple thresholding).

If compared against learning methods which are currently the top-performing approaches in the field, our segmentation procedures hold the advantage of not requiring any training data and potentially make the process of generating training data more efficient. Accurate manual segmentations are very hard to produce and take many user hours. At the same time, image-based segmentation approaches like ours are more prone to local errors since no information about the expected shape is incorpo-

rated in the segmentation procedure.

9.2 Future work

There are multiple dimensions of interest to extend the current work presented in this thesis. A few important ones are flagged here.

9.2.1 Evaluation methods

In the future, it would be interesting to perform extended segmentation ranking experiments to compare boundary-overlap-based metrics against the discussed and other existing evaluation metrics. An important target of such comparisons is to check whether SBD, or other boundary-overlap-based, rankings correlate better than other metrics with human-produced rankings.

An additional aspect of interest is to estimate how accurate a boundary overlap metric can be if not every point on the region boundaries is taken into account. For certain boundary shapes, this may be an option in order to improve execution time. The execution time can also be improved by attempting parallel GPU implementation of boundary overlap metrics.

Initial results for other metrics from our family in addition to SBD, reported in Section 5.4, suggest that these can also be used in segmentation evaluation. A caveat is that they inherit some of the limitations of the corresponding overlap-based metrics applied in the small neighbourhoods. Large-scale systematic experiments can follow with wider ranges of segmentation results, as well as further analysis of the strengths and weaknesses of this new family of metrics in comparison to metrics recently proposed in the literature.

A separate topic for future work is the possibility to use the boundary-overlap-based metrics in additional tasks related to medical image segmentation. In partic-

ular, it would be interesting to explore the potential of these methods to be used as loss functions in machine learning algorithms.

9.2.2 Watershed and waterfall

In many domains, the watershed algorithms of choice are the ones which successfully incorporate markers in order to solve the oversegmentation problem. Our path reducing watershed algorithm does not classify as such since it constructs paths of steepest descent in a top–down manner. It would be interesting to check if in such domains PRW-based waterfall can be a fast parallel substitute to currently employed marker-based watersheds.

A few modifications of the waterfall transform have been proposed in the existing literature, including the enhanced, standard, and P algorithms [21, 20]. The development of parallel algorithms for these waterfall modifications and their extensions on GPU devices is an interesting vector of future research.

We have described in Section 6.4 how watershed lines can get dislocated when hierarchical segmentation is constructed using our PRW-based waterfall transform. This is one important obstacle to using this parallel waterfall algorithm for IPF construction since the rules of region inclusion are not immediately applicable. In the future, the different instruments for overcoming this obstacle can be researched into. Two different starting points intuitive to us are the enforcement of watershed line locations to prevent dislocation or, alternatively, some rules of region inclusion based on the location of regional minima.

9.2.3 Extended class of fast dashing methods

We have shown in Section 7.5 that SDM has a large computational advantage over FDM. At the same time, based on visual inspection, the difference between arrival time images produced with FDM and SDM is small. It would be interesting to com-

pare the accuracy of these two methods for the task of medical image segmentation. If SDM proves to produce not much inferior results, then it can be utilised to improve the segmentation runtimes even further, especially in the framework of interactive segmentation software.

In addition, it is possible to extend or modify the FDM and SDM definitions to produce new front propagation methods in partitioned domains. For instance, the unconditional 'dashing' propagation inside partitioning regions can be replaced with constant speed propagation. This can prove useful if the available partitioning includes mistakes. Instead of including or excluding the whole partitioning region, the arrival times can be thresholded to include only part of that partitioning region.

9.2.4 Fully automated GPU-based segmentation

Finally, fully automated and fast segmentation procedures based on parallel GPU algorithms can be developed for various specific medical image segmentation tasks using the basis of PRW and FDM. Similar to our sequential kidney segmentation algorithm in Chapter 4, the seeding of the front propagation can be done automatically by filtering for candidate regions in the hierarchical segmentation of the image. For each body part and each anatomical structure of interest, a separate set of filtering rules needs to be built based on anatomical knowledge, specifications of imaging modality, and initial small experiments. For example, in the case of femurs in sagittal knee MRI data, the filtering rules can require the region to be a fairly large centrally located one in the top half of the slices.

For the production of segmentation masks from arrival time images, smart automatic thresholding techniques can be utilised. These can, for instance, recognise the arrival at region boundaries by analysing the histogram of the arrival time image. Thus the requirement for user input will be eliminated. Furthermore, the use of the PRW and FDM methods can be part of a longer chain of processing steps in a seg-

mentation procedure. FDM seeds can be extracted from an initial approximation of the region of interest, acquired with other approaches. And conversely, FDM results can be piped into further processing with alternative segmentation algorithms. There is large potential to develop fully automated and accurate segmentation procedures by combining our methods with machine learning algorithms, atlases, shape priors, etc.

In addition, our current results can also be the foundation for easy-to-use GUI segmentation software intended for efficient interactive segmentation. A simplest GUI would include slice-by-slice visualisation of the input images, mouse clicks on which would create seed points. For thresholding, a slide bar of the arrival time range would be sufficient to choose a simple threshold. As a consequence of the extremely fast execution times of our methods, straightforward and fast interactive segmentation of large image volumes would be possible in this way.

Appendix A

Knee MRI Segmentation: Qualitative Results

In Chapter 8 a fast and simple segmentation procedure was applied on femur and tibia bones in nine knee MRI 3D image volumes. Below the labelled region masks for four out of the nine knee volumes, corresponding to the scores reported in Tables 8.1 and 8.3, are presented in a slice-by-slice manner.

Figures A.1, A.2, A.3, and A.4 illustrate the labelled masks for fem3–tib3, fem5–tib5, fem7–tib7, and fem9–tib9 pairs respectively. The middle column contains the original greyscale slices. The first and the second columns correspond to the femur ground truth and machine segmentation. The tibia ground truth and machine segmentation are in the fourth and the fifth columns.

Note that in all cases the machine segmentation was produced with an FDM propagation from a single seed position. This seed is commonly located in the middle slice of the 3D volume. Hence, due to the anisotropic nature of the image volumes, the accuracy of the intra-slice propagation is significantly better than that of the inter-slice propagation. This trend is easily noticeable in the figures below. The first few and the last few slices are generally more poorly segmented than the middle slices.

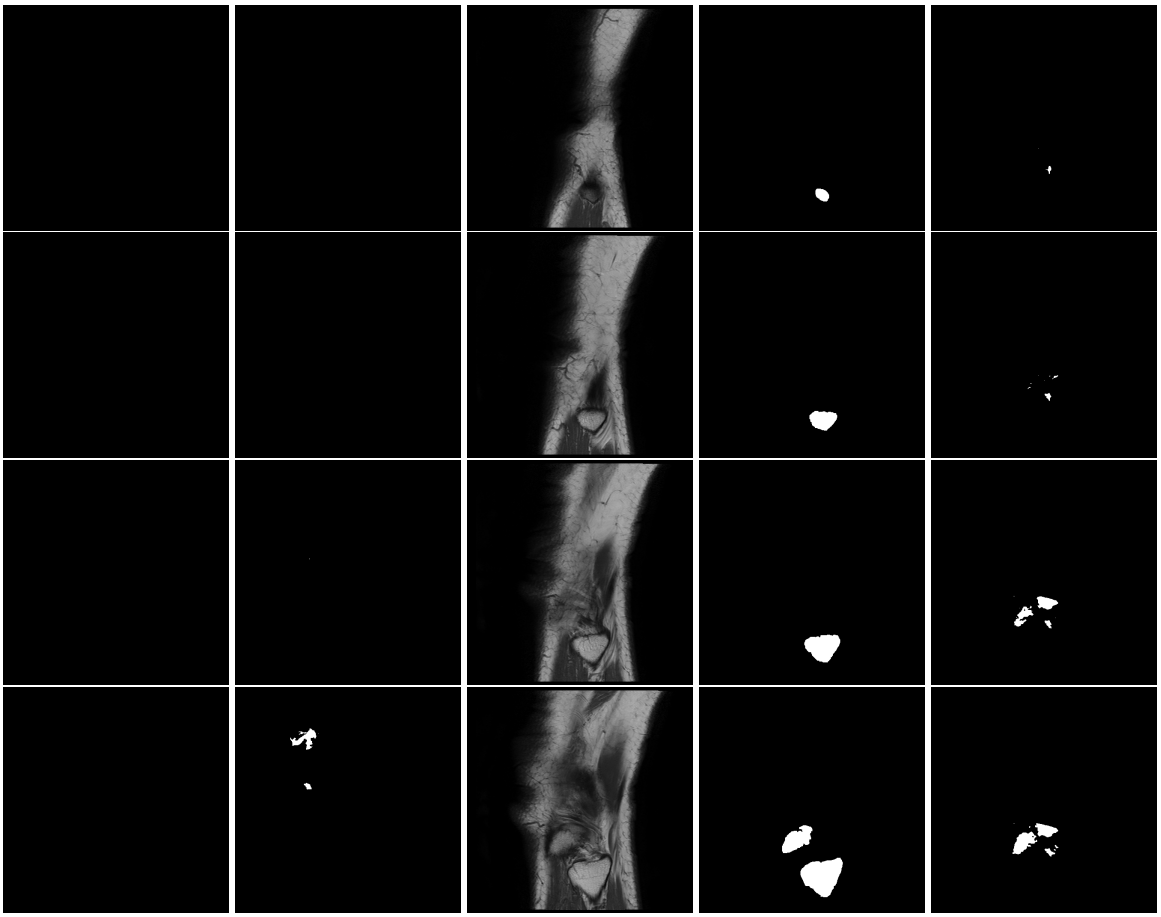


Figure A.1: The femur and tibia segmentation results for fem3 and tib3, first four slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem3 (tib3) in first and second (fourth and fifth) columns.

All four presented femur ground truths are cropped from top, i.e. do not include top rows in any of the image slices. The ground truth of tib7 is also cropped, i.e. does not include bottom rows in any of the image slices. Finally, the fibula bone is included in the ground truth of tib3. These are all important factors that introduce some bias in the evaluation and bring down the scores of these machine segmentation results.

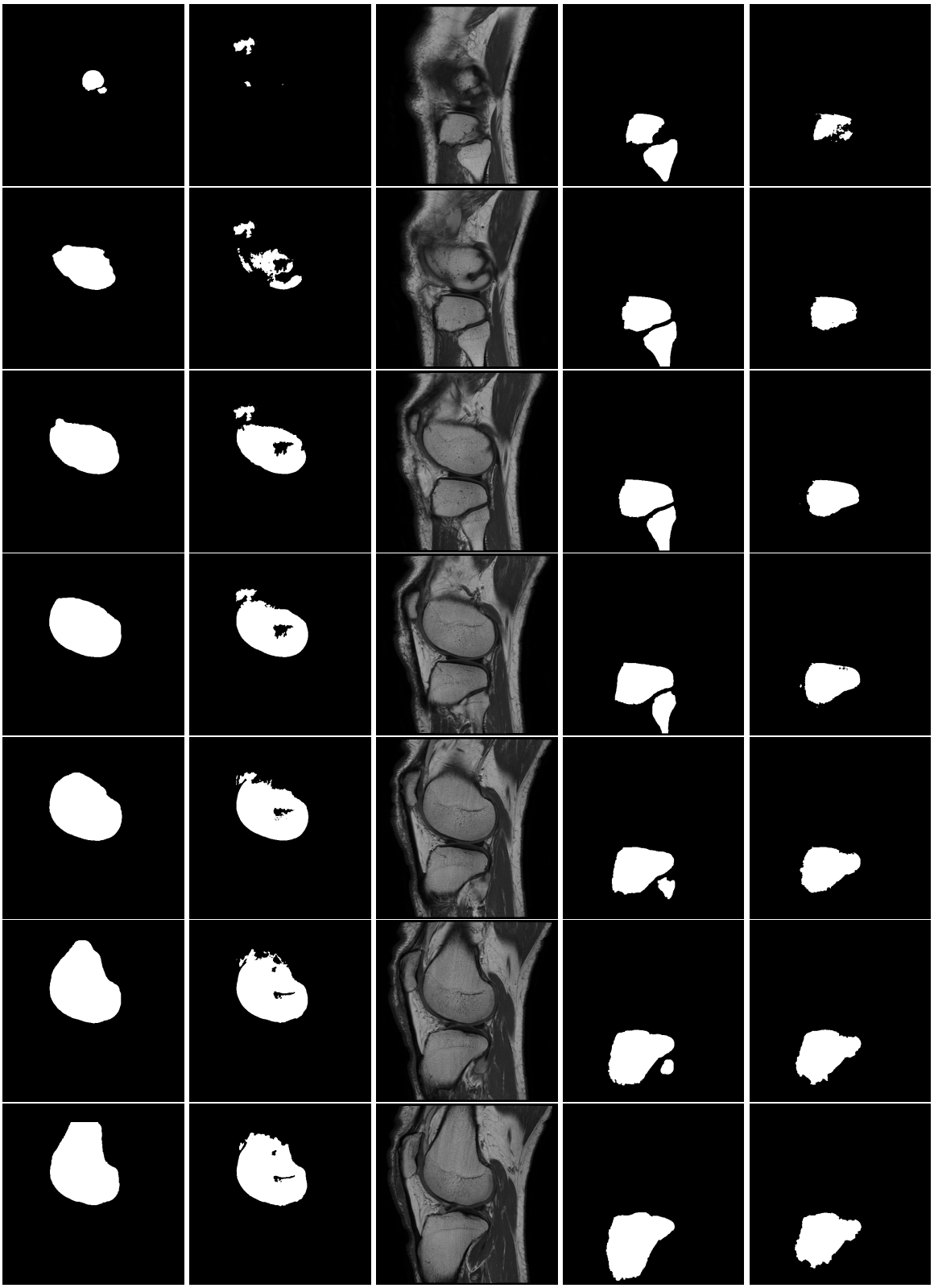


Figure A.1: The femur and tibia segmentation results for fem3 and tib3, last seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem3 (tib3) in first and second (fourth and fifth) columns.

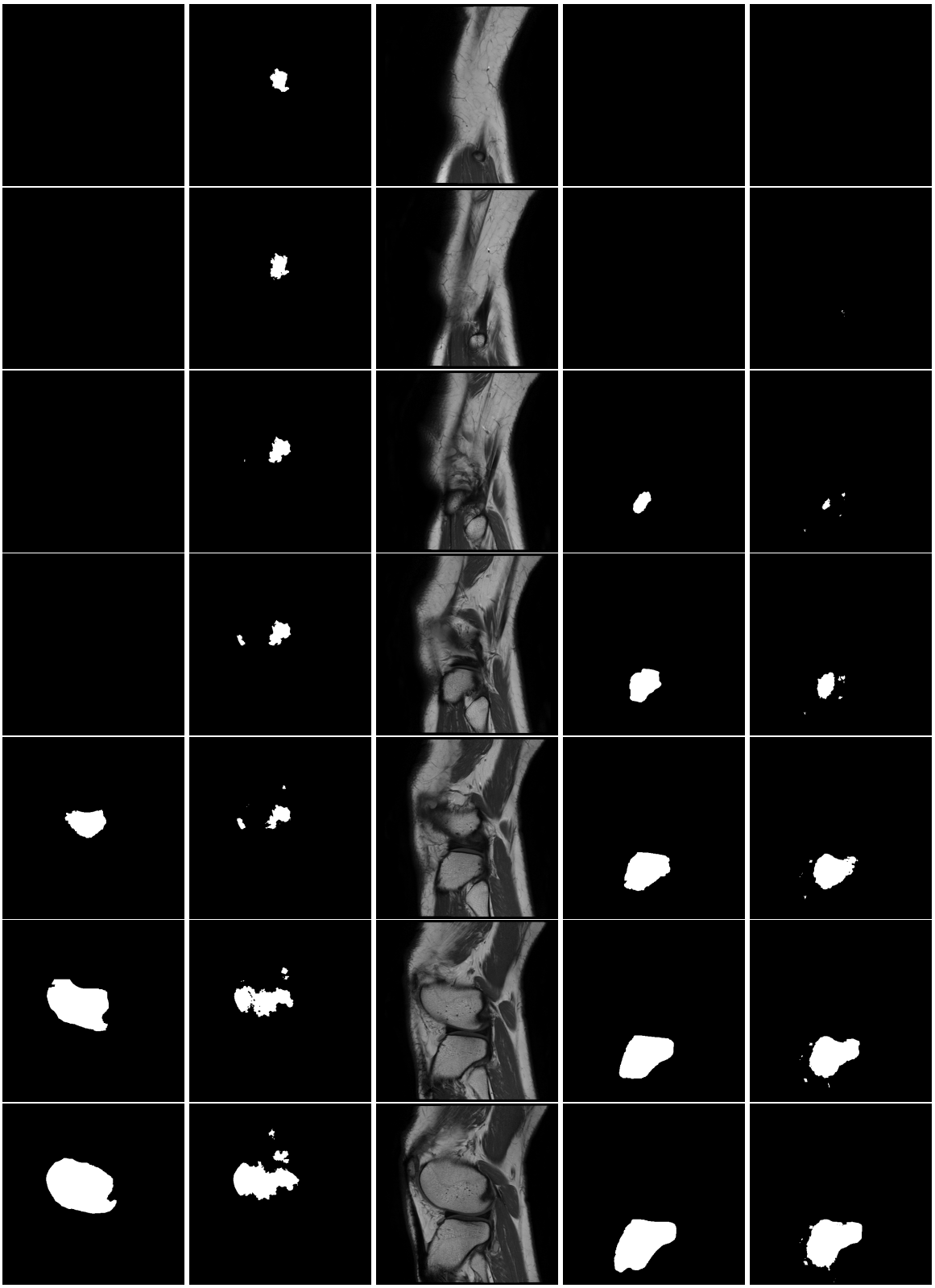


Figure A.2: The femur and tibia segmentation results for fem5 and tib5, first seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns.

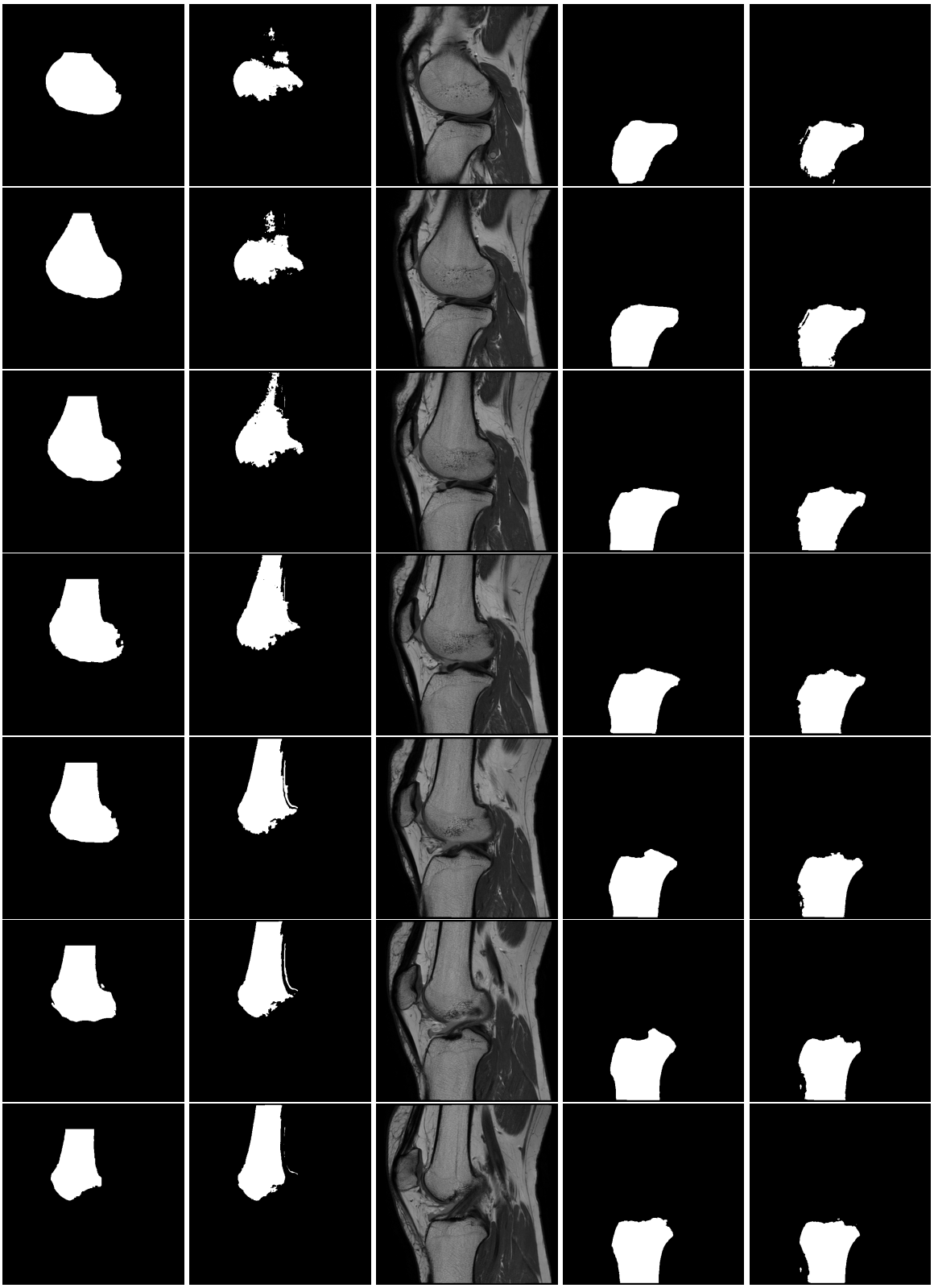


Figure A.2: The femur and tibia segmentation results for fem5 and tib5, second seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns.

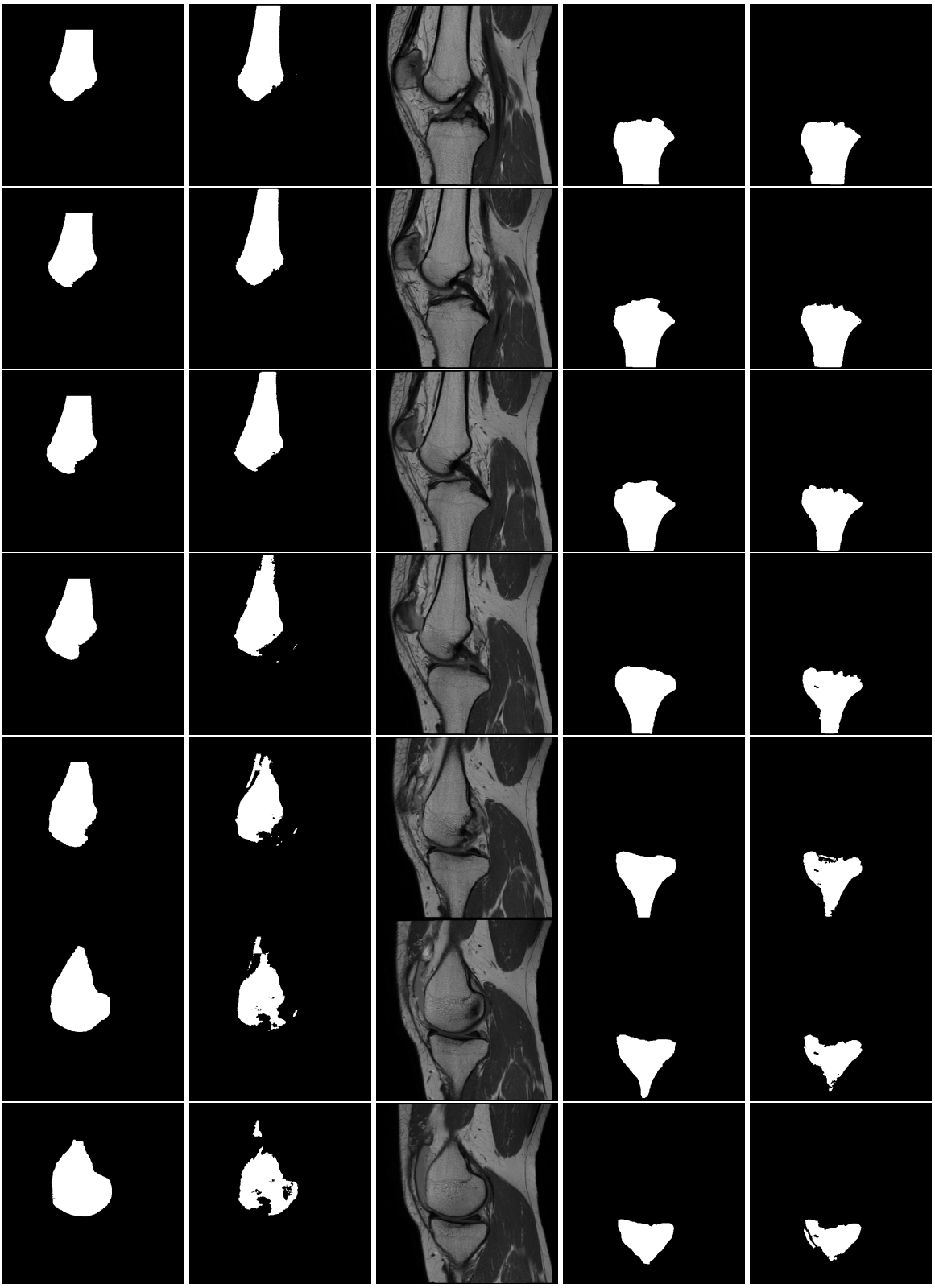


Figure A.2: The femur and tibia segmentation results for fem5 and tib5, third seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns.

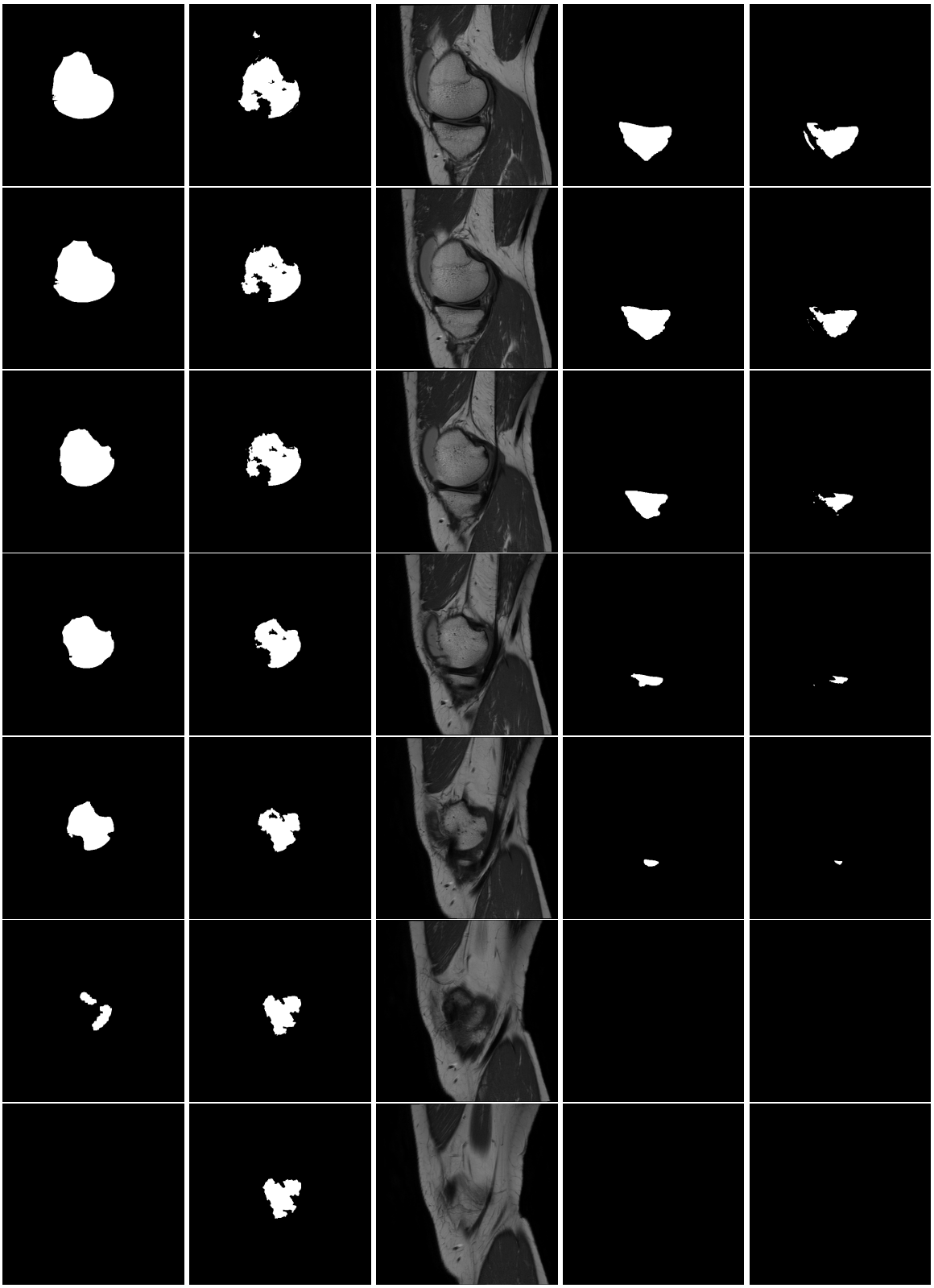


Figure A.2: The femur and tibia segmentation results for fem5 and tib5, fourth, and last, seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem5 (tib5) in first and second (fourth and fifth) columns.

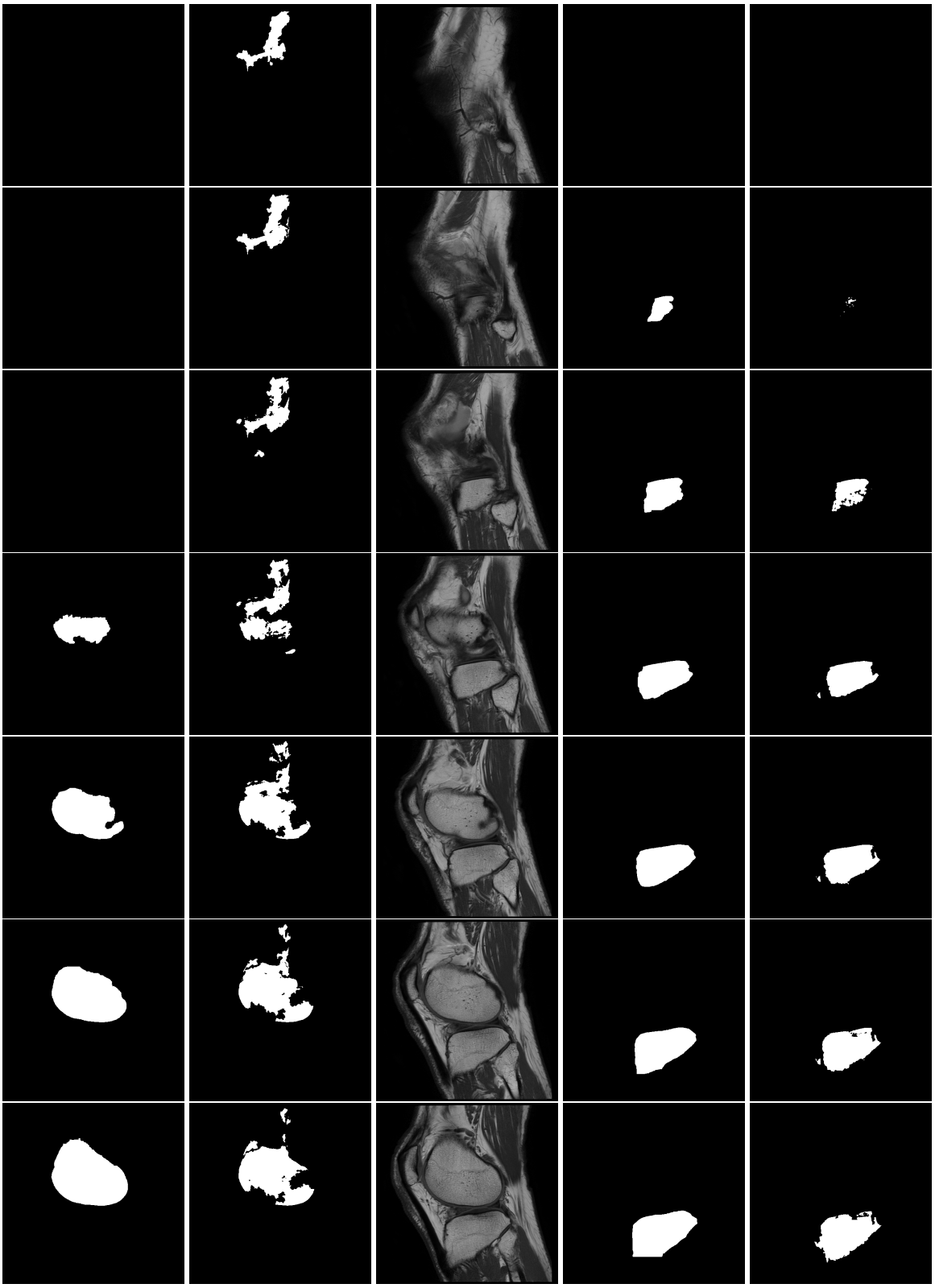


Figure A.3: The femur and tibia segmentation results for fem7 and tib7, first seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns.

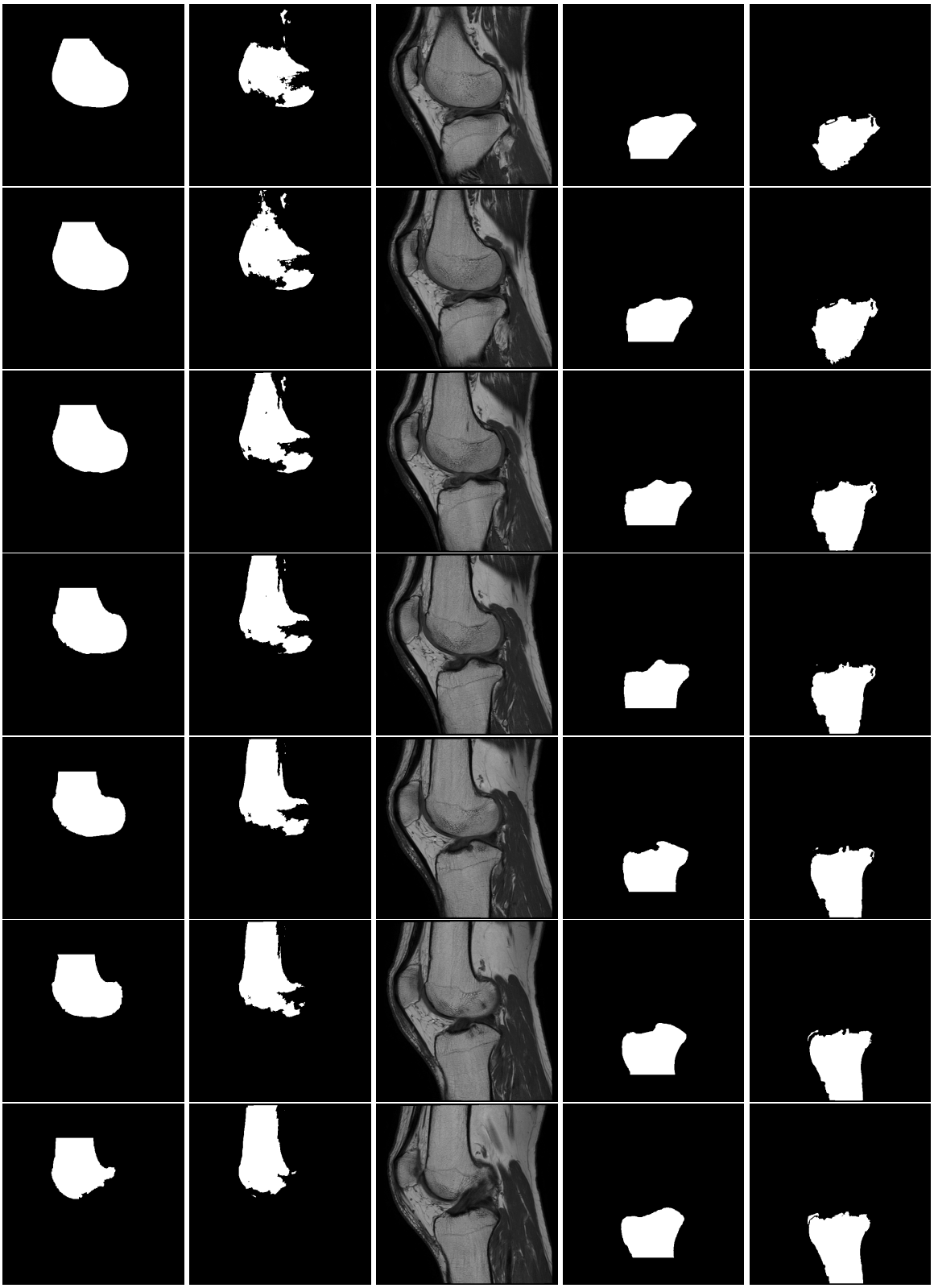


Figure A.3: The femur and tibia segmentation results for fem7 and tib7, second seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns.

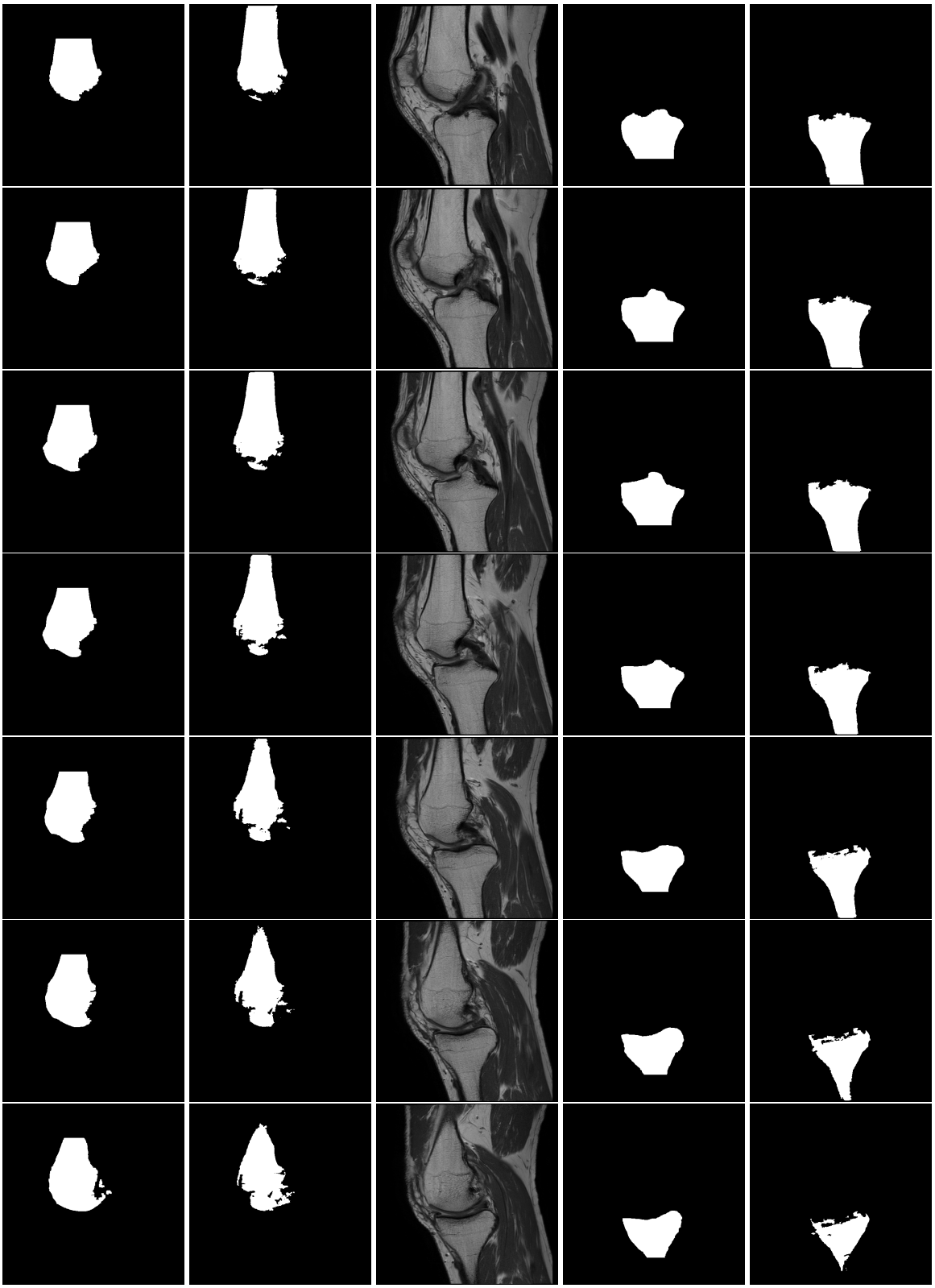


Figure A.3: The femur and tibia segmentation results for fem7 and tib7, third seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns.

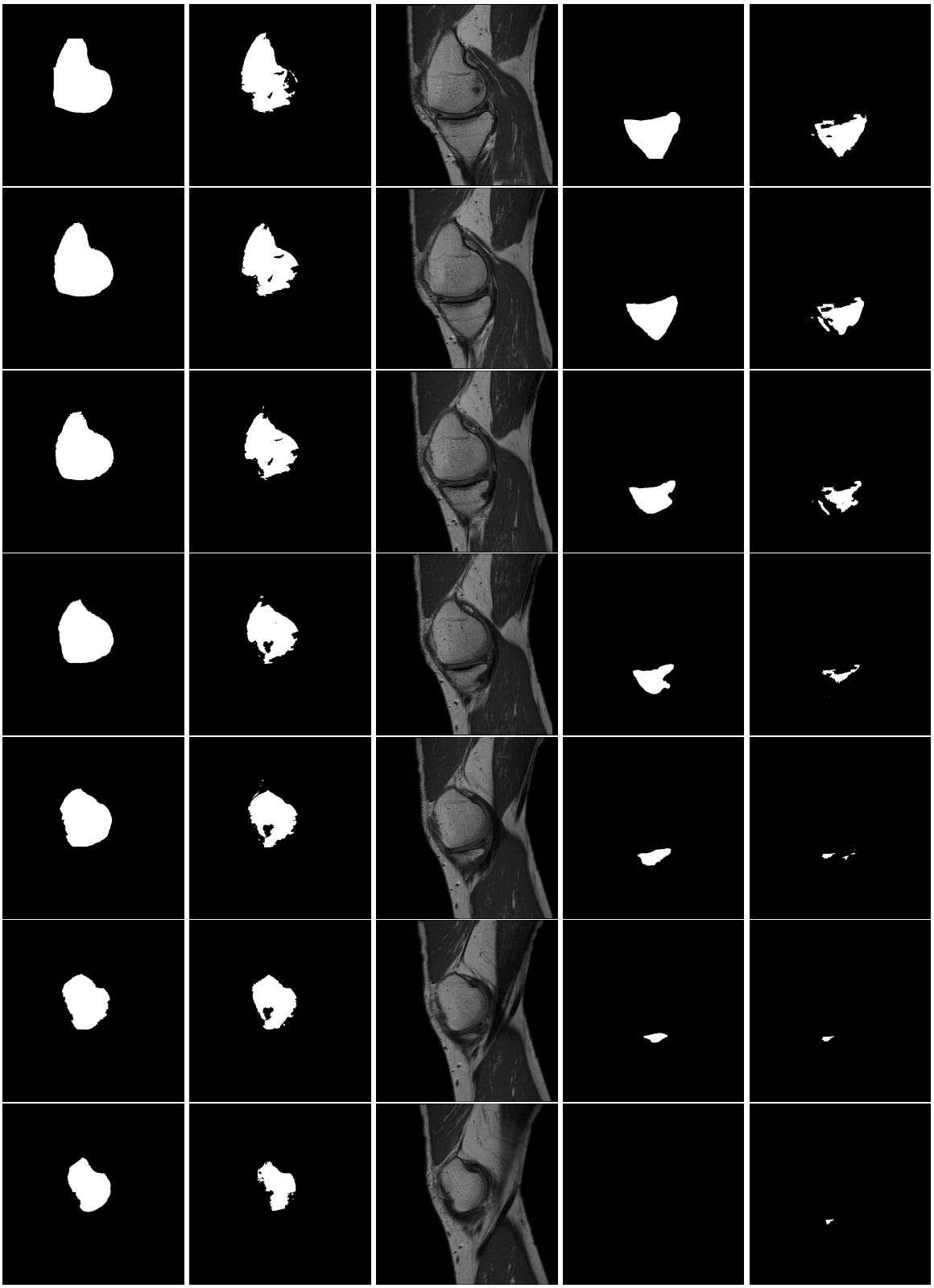


Figure A.3: The femur and tibia segmentation results for fem7 and tib7, fourth seven slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns.

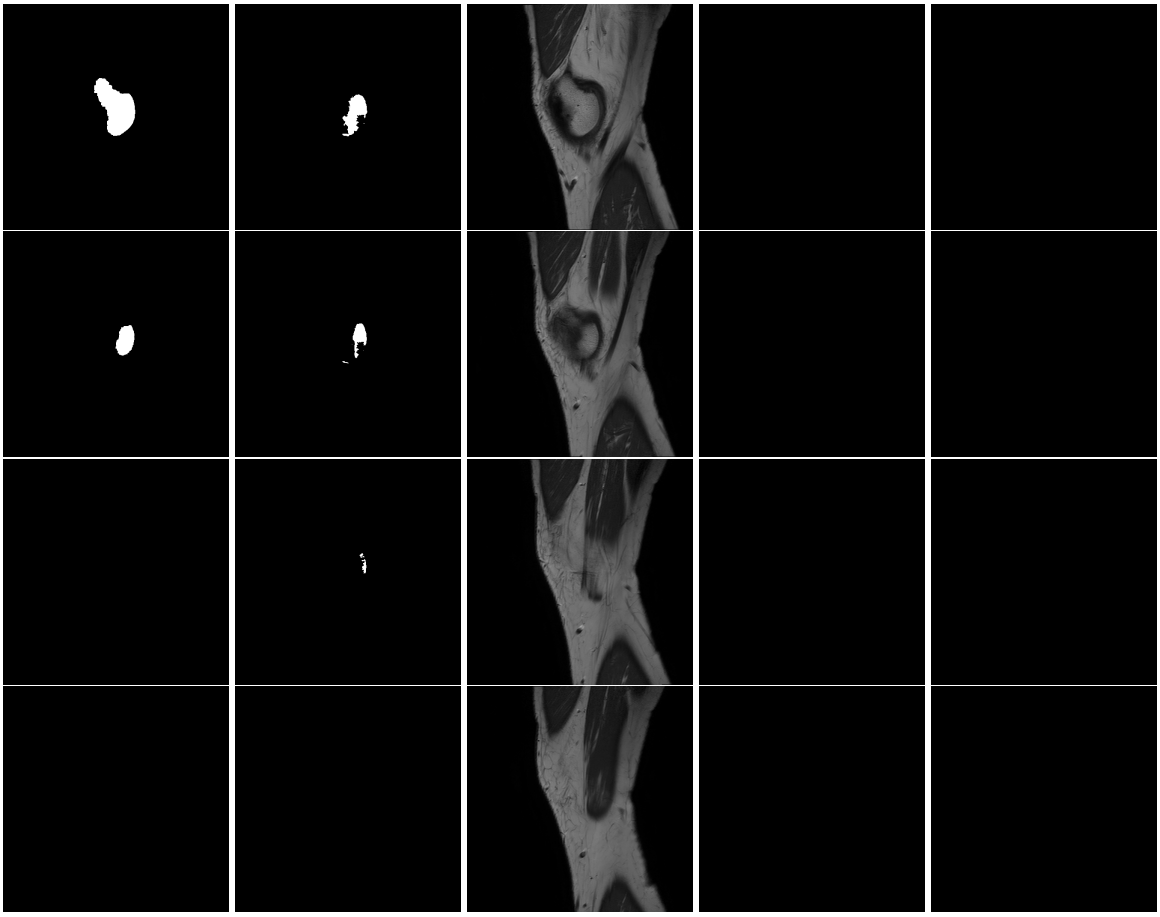


Figure A.3: The femur and tibia segmentation results for fem7 and tib7, last four slices. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem7 (tib7) in first and second (fourth and fifth) columns.



Figure A.4: The femur and tibia segmentation results for fem9 and tib9, first slice. Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns.

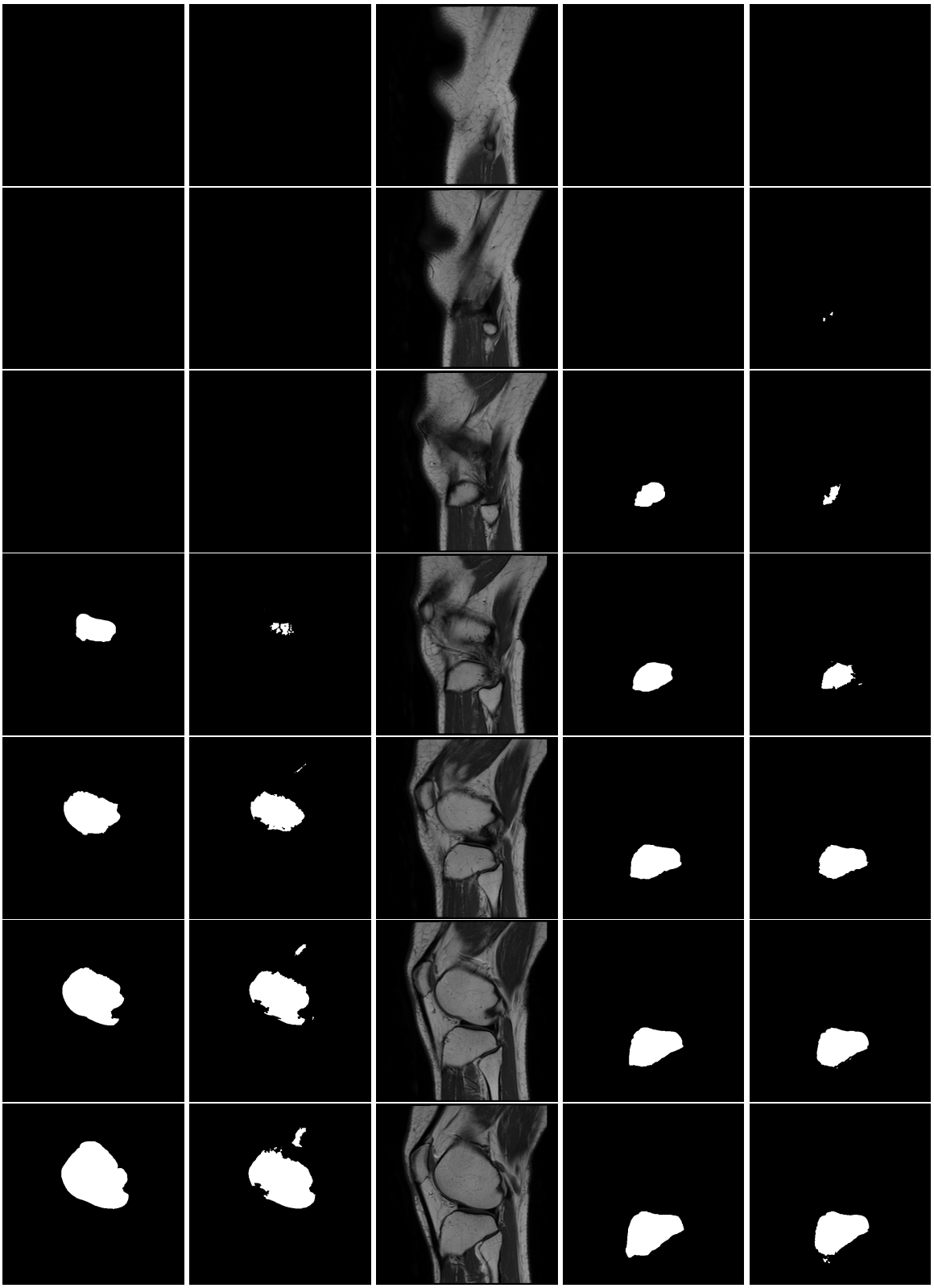


Figure A.4: The femur and tibia segmentation results for fem9 and tib9, next seven slices (2–8). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns.

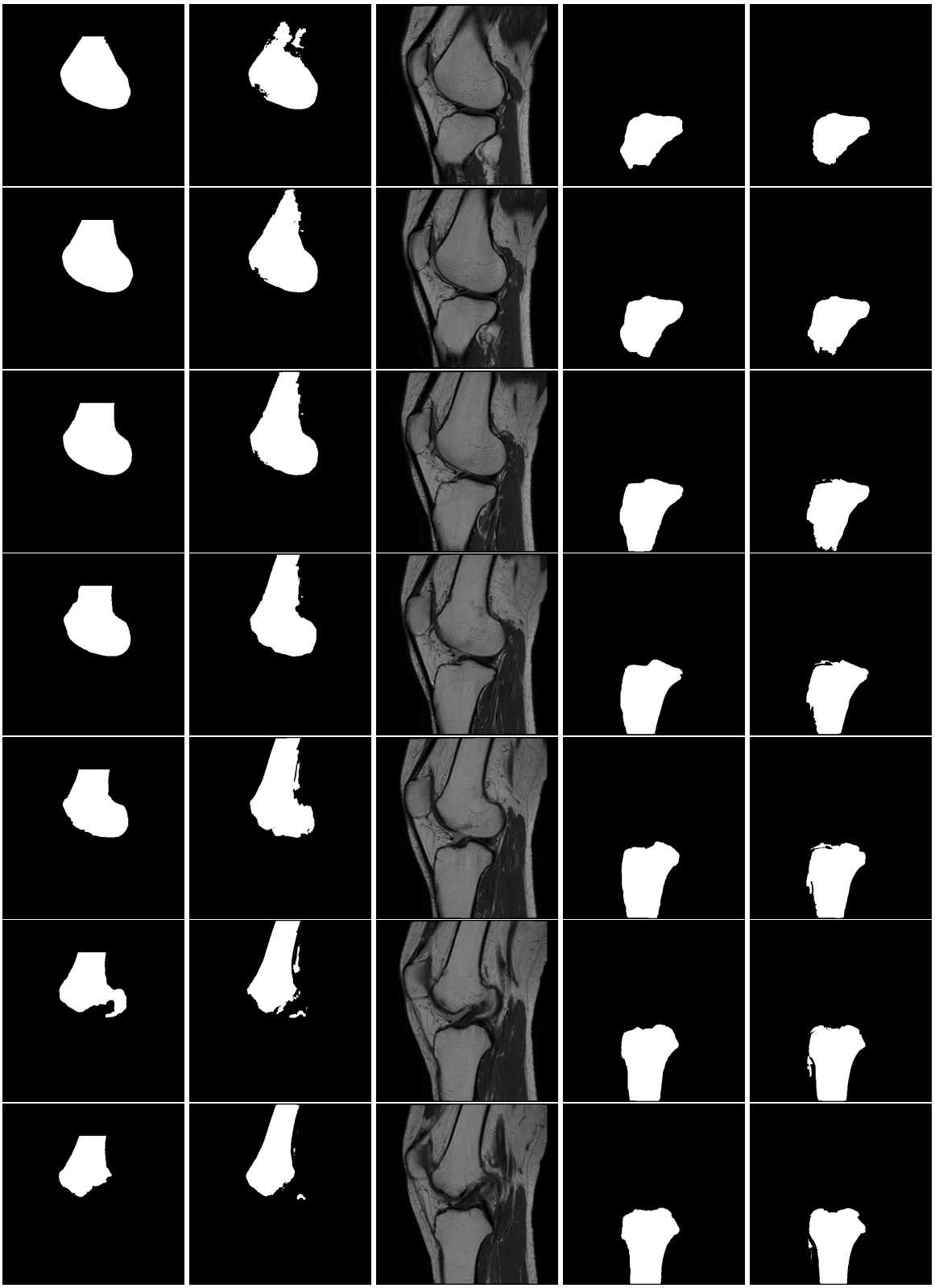


Figure A.4: The femur and tibia segmentation results for fem9 and tib9, next seven slices (9–15). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns.

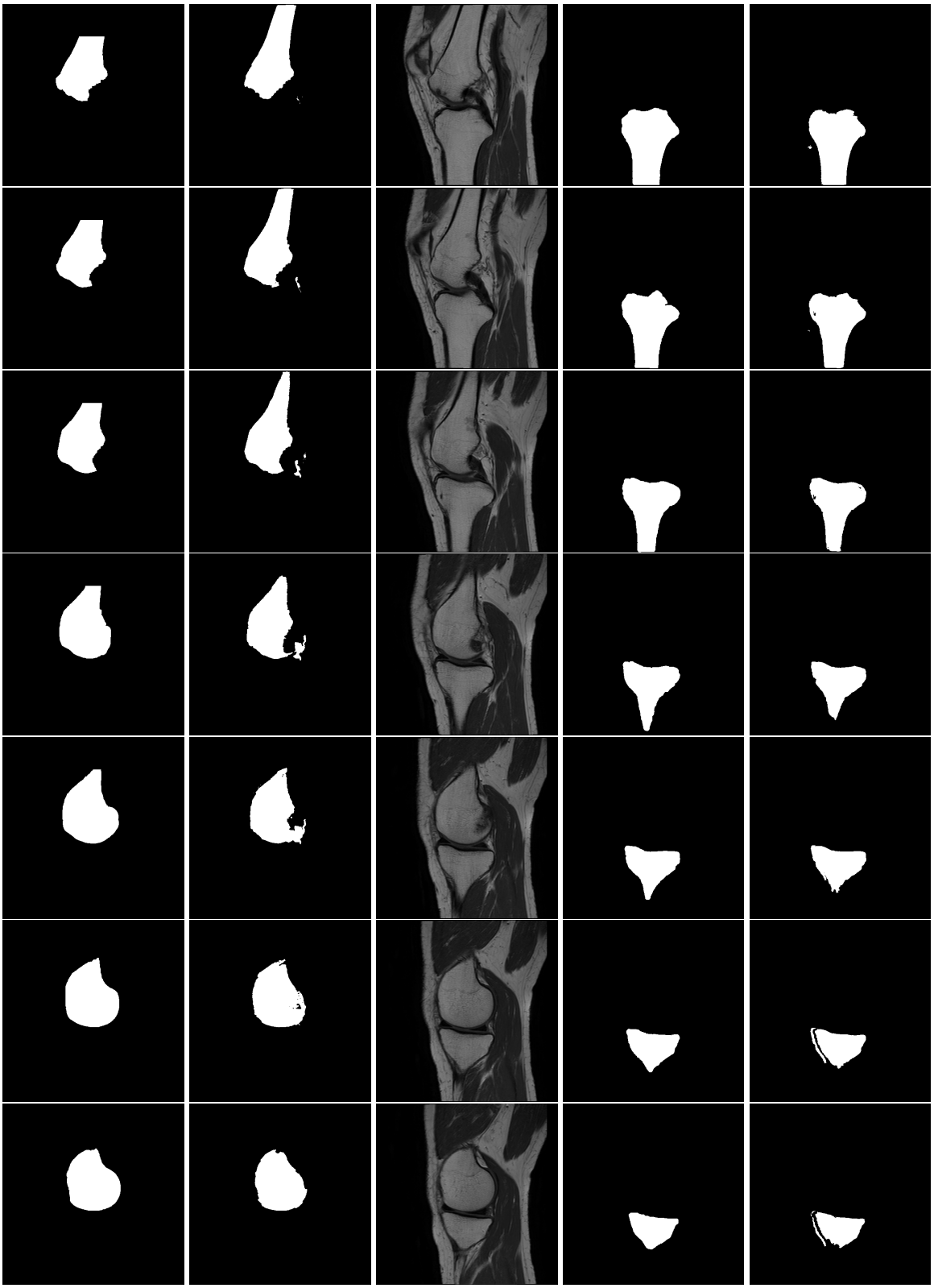


Figure A.4: The femur and tibia segmentation results for fem9 and tib9, next seven slices (16–22). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns.

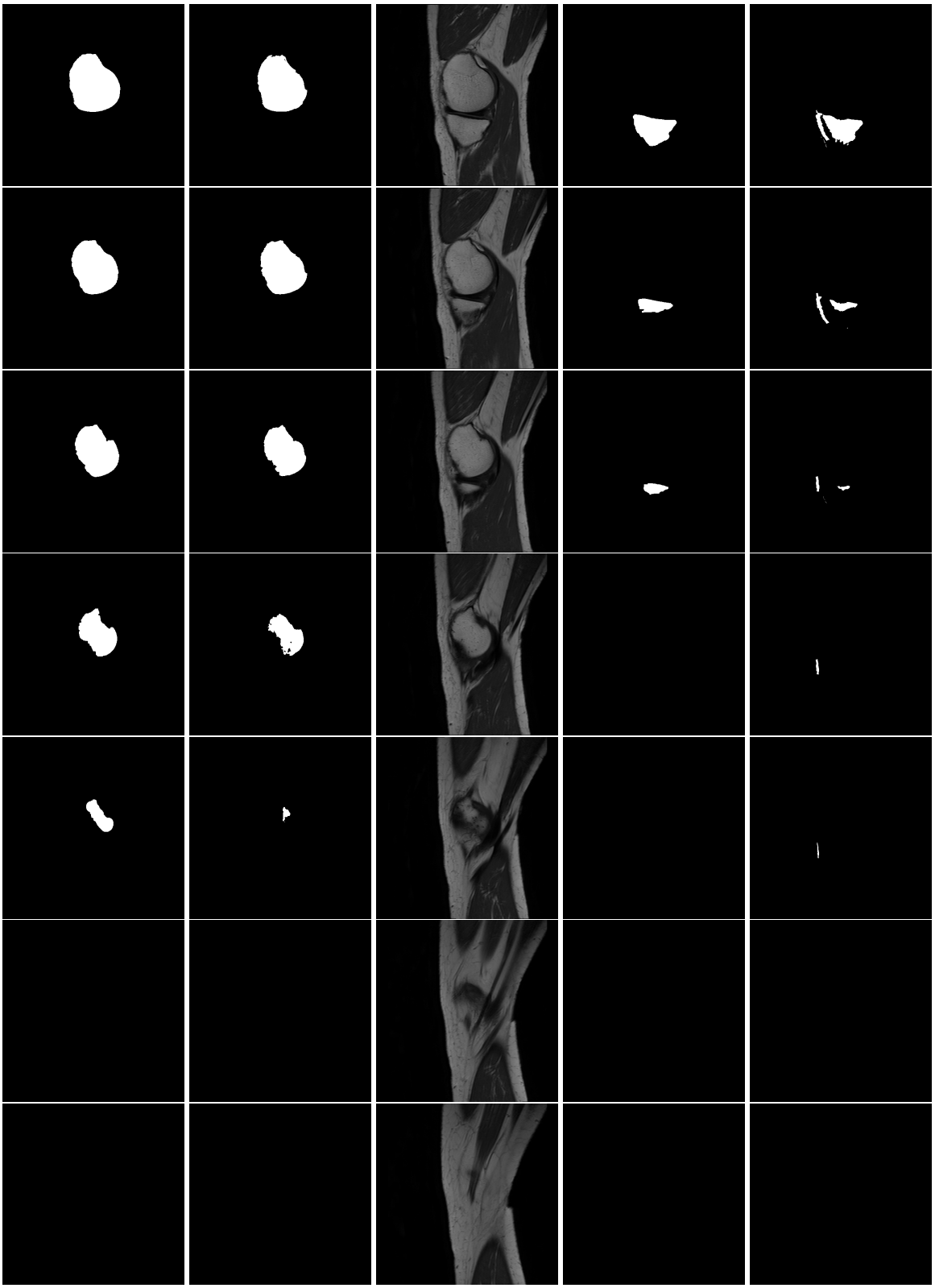


Figure A.4: The femur and tibia segmentation results for fem9 and tib9, last seven slices (23–29). Original greyscale slices are in the middle column. Ground truth and machine segmentation masks for fem9 (tib9) in first and second (fourth and fifth) columns.

Appendix B

Software

Brief descriptions are given for the different pieces of software developed in the context of this thesis. Not every single piece of code is mentioned: I have limited the list to the ones that may be interesting to anyone who wishes to extend the work presented in this thesis.

B.1 OxMedIS

Millipede is image segmentation, feature labelling, and visualisation software mainly developed by S. Golodetz [43]. I initially extended this software as part of my MSc in Computer Science dissertation [133].

Later, updated software called *OxMedIS* [10] (abbreviated from 'Oxford Medical Image Segmentation') was developed on the basis of *millipede*. Different people have made contributions to extend the functionality of *OxMedIS*. My personal contribution to this software has included, in particular, the implementation of the sequential 3D segmentation procedure for kidneys in abdominal CT scans (Chapter 4).

B.2 Evaluation methods

eval_pbm_images.py is a Python script for the evaluation of several 2D masks against a gold standard using overlap-based, boundary-distance-based, and boundary-overlap-based metrics.

eval_pbm3_images.py is a Python script for the evaluation of several 3D masks against a gold standard using overlap-based, boundary-distance-based, and boundary-overlap-based metrics.

boundary_distance.cu is a CUDA GPU implementation of the HD and ASSD metrics for the comparison of 3D masks. My Python non-optimised implementations of these metrics were too slow (of the order of a few hours for images of 3 megavoxels).

B.3 Partitioning CUDA GPU programmes

3mywatershed.cu is the PRW_{sync} implementation applicable both on 2D and 3D images.

2Dasync3myws.cu is the $\text{PRW}_{\text{async}}$ implementation for 2D images.

3Dasync3myws.cu is the $\text{PRW}_{\text{async}}$ implementation for 3D images.

sym2Dasync3myws.cu is the PRW_{bal} implementation for 2D images.

sym3Dasync3myws.cu is the PRW_{bal} implementation for 3D images.

2Dmywaterfall.cu is a hierarchical partitioning implementation for 2D images that performs a fixed number of waterfall transforms based on PRW_{bal}.

3Dmywaterfall.cu is a hierarchical partitioning implementation for 3D images that performs a fixed number of waterfall transforms based on PRW_{bal}.

mywatershed, 2mywatershed, 4mywatershed are three different watershed algorithms that were developed initially, compared, and improved upon in the process of designing the PRW algorithm.

B.4 Front propagation CUDA GPU programmes

fastdashing.cu is the FDM_{sync} implementation applicable on both 2D and 3D images.

2Dfastdashing.cu is the $\text{FDM}_{\text{async}}$ implementation for 2D images.

3Dfastdashing.cu is the $\text{FDM}_{\text{async}}$ implementation for 3D images.

simpledashing.cu is the SDM_{sync} implementation applicable on both 2D and 3D images.

2Dsimpledashing.cu is the $\text{SDM}_{\text{async}}$ implementation for 2D images.

3Dsimpledashing.cu is the $\text{SDM}_{\text{async}}$ implementation for 3D images.

2Dfastiterative.cu is the $\text{FUM}_{\text{async}}$ implementation for 2D images[1].

3Dfastiterative.cu is the $\text{FUM}_{\text{async}}$ implementation for 3D images.

2Dfastmarching.cpp is the FMM sequential C++ implementation for 2D images.

3Dfastmarching.cpp is the FMM sequential C++ implementation for 3D images.

Sequential Python prototypes of my GPU algorithms (both partitioning and front propagation) were developed first, and only after initial tests with these, were the CUDA programmes written.

[1]The naming difference is a result of legacy issues.

B.5 Other tools

I also implemented a number of different tools in Python mostly to assist in the file manipulation. Some of these tools are designed to convert 2D and/or 3D images between file formats; to crop machine segmentation masks at top or bottom of slices according to the corresponding ground truth masks; to smooth input and produce gradient magnitude images; overlay input image with watershed lines; threshold arrival time image and produce region mask, etc.

Appendix C

Publications

My current publications based on the work presented in this thesis are as follows.

C.1 Journal

A family of boundary overlap metrics for the evaluation of medical image segmentation

Varduhi Yeghiazaryan and Irina Voiculescu

Journal of Medical Imaging, SPIE, 2018. *Basis of Chapter 5, under review.*

Hierarchical image segmentation with path reducing watershed on the GPU

Varduhi Yeghiazaryan and Irina Voiculescu

In preparation to be submitted to IEEE Transactions on Medical Imaging.

C.2 Conference

Path Reducing Watershed for the GPU

Varduhi Yeghiazaryan and Irina Voiculescu

In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV).* IEEE, March, 2018. *Basis of Chapter 6, to appear.*

Boundary overlap for medical image segmentation evaluation

Varduhi Yeghiazaryan and Irina Voiculescu

In *Proceedings of SPIE 10135, Medical Imaging 2017: Image-Guided Procedures, Robotic Interventions, and Modeling*, 101351J. SPIE, March, 2017. *Basis of Chapter 5*

Automated 3D renal segmentation based on image partitioning

Varduhi Yeghiazaryan and Irina D. Voiculescu

In *Proceedings of SPIE 9784, Medical Imaging 2016: Image Processing*, 97842E. SPIE International Society for Optics and Photonics, February, 2016. *Basis of Chapter 4*

Parallel fast dashing in partitioned medical images on the GPU

Varduhi Yeghiazaryan and Irina Voiculescu

In preparation to be submitted to MICCAI 2018.

C.3 Internal

An overview of current evaluation methods used in medical image segmentation

Varduhi Yeghiazaryan and Irina Voiculescu

Technical report No. CS-RR-15-08. Department of Computer Science. Oxford, UK. 2015.

The use of fast marching methods in medical image segmentation

Varduhi Yeghiazaryan and Irina Voiculescu

Technical report No. CS-RR-15-07. Department of Computer Science. Oxford, UK. 2015.

Experiments on the use of fast marching for feature identification

Varduhi Yeghiazaryan and Irina Voiculescu

In *Proceedings of the Oxford University Department of Computer Science Student Conference 2014*. 2014. **Best Abstract Award**

C.4 Other presentations

Novel Parallel Watershed for Faster Image Partitioning

Varduhi Yeghiazaryan and Irina Voiculescu

Presentation at Oxford Computer Science Conference 2017.

The sensitivity to boundary errors of image segmentation evaluation measures

Varduhi Yeghiazaryan and Irina Voiculescu

Presentation at Oxford Computer Science Conference 2016.

Common evaluation measures in medical image segmentation: experimental comparison

Varduhi Yeghiazaryan and Irina Voiculescu

Presentation at Oxbridge Women in Computer Science Conference 2016.

Fast marching method for organ identification in CT images

Varduhi Yeghiazaryan and Irina Voiculescu

Presentation at Oxbridge Women in Computer Science Conference 2014.

C.5 Posters

Boundary overlap for medical image segmentation evaluation

Varduhi Yeghiazaryan and Irina Voiculescu

Poster at Oxbridge Women in Computer Science Conference 2017. **Best Poster Award**

Poster at Oxford Biomedical Imaging Festival 2017.

Towards unsupervised segmentation of the knee from MRI images based on image partition forests

Marija Marčan, Varduhi Yeghiazaryan and Irina Voiculescu

Poster at Oxford Biomedical Imaging Festival 2015.

Comparison of evaluation methods for medical image segmentation

Varduhi Yeghiazaryan and Irina Voiculescu

Poster at Oxford University Department of Computer Science Student and Research Staff Conference 2015. **Best Poster Award**

Poster at Oxford Biomedical Imaging Festival 2015.

Progress towards unsupervised medical image segmentation

Varduhi Yeghiazaryan and Irina Voiculescu

Poster at Oxbridge Women in Computer Science Conference 2015.

Bibliography

[1] 3D Slicer. www.slicer.org. Accessed: 17/6/2018.

[2] BSDS500. www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html. Accessed: 4/9/2017.

[3] Compute unified device architecture (CUDA). www.nvidia.com/object/cuda_home_new.html. Accessed: 4/9/2017.

[4] DW. adessowiki.fee.unicamp.br/adesso/wiki/watershed/ismm2011_dw/view/. Accessed: 4/9/2017.

[5] FIM. github.com/SCIInstitute/StructuredEikonal. Accessed: 8/11/2017.

[6] ISLES challenge 2015. www.isles-challenge.org/ISLES2015/. Accessed: 1/12/2017.

[7] ITK-SNAP. www.itksnap.org/pmwiki/pmwiki.php. Accessed: 17/6/2018.

[8] Lena wiki. en.wikipedia.org/wiki/Lenna#/media/File:Lenna.png. Accessed: 4/9/2017.

[9] Mimics^{TM}. biomedical.materialise.com/mimics. Accessed: 1/12/2017.

[10] OxMedIS. www.cs.ox.ac.uk/projects/OxMedIS/index.html. Accessed: 19/12/2017.

[11] PubMed. www.ncbi.nlm.nih.gov/pubmed. Accessed: 17/6/2018.

[12] Simpleware™. simpleware.com/software. Accessed: 1/12/2017.

[13] SKI10 grand challenge. ski10.org/index.php. Accessed: 1/12/2017.

[14] Standard test images. www.ece.rice.edu/~wakin/images/. Accessed: 4/9/2017.

[15] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[16] Jakob Andreas Bærentzen. On the implementation of fast marching methods for 3D lattices. Technical report, 2001.

[17] Ulas Bağci, Xinjian Chen, and Jayaram K. Udupa. Hierarchical scale-based multiobject recognition of 3-D anatomical structures. *IEEE Transactions on Medical Imaging*, 31(3):777–789, 2012.

[18] Serge Beucher. *Segmentation d'images et morphologie mathématique*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris, 1990.

[19] Serge Beucher. Watershed, hierarchical segmentation and waterfall algorithm. In Jean Serra and Pierre Soille, editors, *Mathematical Morphology and Its Applications to Image Processing, Proc. ISMM'94*, pages 69–76. Kluwer Ac. Publ., Nld, 1994.

[20] Serge Beucher. Towards a unification of waterfalls, standard and p algorithms. 2013.

[21] Serge Beucher and Beatriz Marcotegui. P algorithm, a dramatic enhancement of the waterfall transformation. 2009.

[22] Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. In E. Dougherty, editor, *Mathematical*

Morphology in Image Processing, pages 433–481. Marcel Dekker, New York, 1993.

[23] Wojciech Bieniecki. Oversegmentation avoidance in watershed-based algorithms for color images. In *Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004.*, pages 169–172. IEEE, 2004.

[24] Andreas Bieniek and Alina Moga. A connected component approach to the watershed segmentation. *Computational Imaging and Vision*, 12:215–222, 1998.

[25] Paola Campadelli, Elena Casiraghi, and Andrea Esposito. Liver segmentation from computed tomography scans: A survey and a new algorithm. *Artificial Intelligence in Medicine*, 45(2-3):185–196, 2009. Computational Intelligence and Machine Learning in Bioinformatics.

[26] Paola Campadelli, Elena Casiraghi, and Stella Pratissoli. Fully automatic segmentation of abdominal organs from CT images using fast marching methods. In *21st IEEE International Symposium on Computer-Based Medical Systems, 2008. CBMS'08*, pages 554–559. IEEE, June 2008.

[27] Paola Campadelli, Elena Casiraghi, and Stella Pratissoli. A segmentation framework for abdominal organs from CT scans. *Artificial Intelligence in Medicine*, 50(1):3–11, 2010.

[28] Rubén Cárdenes, Rodrigo de Luis-García, and Meritxell Bach-Cuadra. A multidimensional segmentation evaluation for medical image data. *Computer Methods and Programs in Biomedicine*, 96(2):108–124, 2009.

[29] Xinjian Chen and Ulas Bagci. 3D automatic anatomy segmentation based on iterative graph-cut-ASM. *Medical Physics*, 38(8):4610–4622, August 2011.

[30] Xinjian Chen, Jayaram K. Udupa, Ulas Bagci, Ying Zhuge, and Jianhua Yao. Medical image segmentation by combining graph cuts and oriented active appearance models. *IEEE Transactions on Image Processing*, 21(4):2035–2046, 2012.

[31] Xinjian Chen, Jianhua Yao, Ying Zhuge, and Ulas Bagci. 3D automatic anatomy segmentation based on graph cut-oriented active appearance models. In *17th IEEE International Conference on Image Processing (ICIP)*, pages 3653–3656, September 2010.

[32] Yu Chen, Zhuangzhi Yan, and Yungao Chu. Cellular automata based level set method for image segmentation. In *IEEE/ICME International Conference on Complex Medical Engineering*, pages 171–174. IEEE, 2007.

[33] Yufei Chen, Zhicheng Wang, Jinyong Hu, Weidong Zhao, and Qidi Wu. The domain knowledge based graph-cut model for liver CT segmentation. *Biomedical Signal Processing and Control*, 7(6):591–598, 2012.

[34] Jean Cousty, Gilles Bertrand, Laurent Najman, and Michel Couprie. Watershed cuts: Minimum spanning forests and the drop of water principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(8):1362–1374, 2009.

[35] William R. Crum, Oscar Camara, and Derek L. G. Hill. Generalized overlap measures for evaluation and validation in medical image analysis. *IEEE Transactions on Medical Imaging*, 25(11):1451–1461, November 2006.

[36] Rémi Cuingnet, Raphael Prevost, David Lesage, Laurent D. Cohen, Benoît Mory, and Roberto Ardon. Automatic detection and segmentation of kidneys in 3D CT images using random forests. In Nicholas Ayache, Hervé Delingette, Polina Golland, and Kensaku Mori, editors, *Medical Image Computing and*

Computer-Assisted Intervention – MICCAI 2012, volume 7512 of *Lecture Notes in Computer Science*, pages 66–74. Springer Berlin Heidelberg, 2012.

[37] Miles Detrixhe, Frédéric Gibou, and Chohong Min. A parallel fast sweeping method for the eikonal equation. *Journal of Computational Physics*, 237:46–55, 2013.

[38] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[39] Nicolas Forcadel, Carole Le Guyader, and Christian Gout. Generalized fast marching method: applications to image segmentation. *Numerical Algorithms*, 48(1):189–211, July 2008.

[40] Hong-Seng Gan, Tian-Swee Tan, Ahmad Helmy Abdul Karim, Khairil Amir Sayuti, and Mohammed Rafiq Abdul Kadir. Interactive medical image segmentation with seed precomputation system: Data from the osteoarthritis initiative. In *IEEE Conference on Biomedical Engineering and Sciences (IECBES)*, pages 315–318. IEEE, December 2014.

[41] Guido Gerig, Matthieu Jomier, and Miranda Chakos. Valmet: A new validation tool for assessing and improving 3D object segmentation. In Wiro J. Niessen and Max A. Viergever, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2001*, volume 2208 of *Lecture Notes in Computer Science*, pages 516–523. Springer Berlin Heidelberg, 2001.

[42] Payel Ghosh, Sameer K. Antani, L. Rodney Long, and George R. Thoma. Unsupervised grow-cut: Cellular automata-based medical image segmentation. In *First IEEE International Conference on Healthcare Informatics, Imaging and Systems Biology*, pages 40–47. IEEE, 2011.

[43] Stuart Golodetz. *Zipping and Unzipping: The Use of Image Partition Forests in the Analysis of Abdominal CT Scans.* DPhil thesis, University of Oxford, 2011.

[44] Stuart Golodetz, Irina Voiculescu, and Stephen Cameron. A proposed decision-support system for (renal) cancer imaging. In *Proceedings of FBIT '07*, pages 361–366, Jeju City, South Korea, October 2007.

[45] Stuart Golodetz, Irina Voiculescu, and Stephen Cameron. Automatic spine identification in abdominal CT slices using image partition forests. In *Proceedings of 6th International Symposium on Image and Signal Processing and Analysis, 2009. ISPA 2009.*, pages 117–122, Salzburg, Austria, September 2009. IEEE.

[46] Stuart Golodetz, Irina Voiculescu, and Stephen Cameron. Progress on a decision-support system for abdominal CT scans (work-in-progress paper). In Lucia Lo Bello and Giancarlo Iannizzotto, editors, *Proceedings of HSI 2009*, pages 116–119, Catania, Italy, May 2009. Best Paper Award (Telemedicine and e-Health).

[47] Stuart Golodetz, Irina Voiculescu, and Stephen Cameron. Simpler editing of graph-based segmentation hierarchies using zipping algorithms. *Pattern Recognition*, 70:44–59, 2017.

[48] Stuart M. Golodetz, Chris Nicholls, Irina D. Voiculescu, and Stephen A. Cameron. Two tree-based methods for the waterfall. *Pattern Recognition*, May 2014.

[49] Raphael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Prentice Hall, 2008. 3rd edition.

[50] Vicente Grau, A. U. J. Mewes, M. Alcañiz, Ron Kikinis, and Simon K. Warfield. Improved watershed transform for medical image segmentation using prior information. *IEEE Transactions on Medical Imaging*, 23(4):447–458, April 2004.

[51] Tobias Heimann, Bryan J. Morrison, Martin A. Styner, Marc Niethammer, and S. Warfield. Segmentation of knee images: A grand challenge. In *Proc. MICCAI Workshop on Medical Image Analysis for the Clinic*, pages 207–214, 2010.

[52] Tobias Heimann, Bram van Ginneken, Martin A. Styner, Yulia Arzhaeva, et al. Comparison and evaluation of methods for liver segmentation from CT datasets. *IEEE Transactions on Medical Imaging*, 28(8):1251–1265, August 2009.

[53] Sumin Hong and Won-Ki Jeong. A group-ordered fast iterative method for eikonal equations. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):318–331, February 2017.

[54] Michal Hučko and Miloš Šrámek. Streamed watershed transform on GPU for processing of large volume data. In *Proceedings of the 28th Spring Conference on Computer Graphics*, SCCG '12, pages 137–141, New York, NY, USA, 2012. ACM.

[55] Won-Ki Jeong and Ross T. Whitaker. A fast eikonal equation solver for parallel systems. In *SIAM conference on Computational Science and Engineering*, 2007.

[56] Won-Ki Jeong and Ross T. Whitaker. A fast iterative method for eikonal equations. *SIAM Journal on Scientific Computing*, 30(5):2512–2534, 2008.

[57] Claude Kauffmann and Nicolas Piche. Cellular automaton for ultra-fast watershed transform on GPU. In *ICPR*, pages 1–4, 2008.

[58] Claude Kauffmann and Nicolas Piché. A cellular automaton framework for image processing on GPU. In *Pattern Recognition*. InTech, 2009.

[59] Claude Kauffmann and Nicolas Piché. Seeded ND medical image segmentation by cellular automaton on GPU. *International Journal of Computer Assisted Radiology and Surgery*, 5(3):251–262, 2010.

[60] Ender Konukoglu, Ben Glocker, Antonio Criminisi, and Kilian M. Pohl. WESD–weighted spectral distance for measuring shape dissimilarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(9):2284–2297, 2013.

[61] André Körbes and Roberto de Alencar Lotufo. Analysis of the watershed algorithms based on the breadth-first and depth-first exploring methods. In *2009 XXII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, pages 133–140. IEEE, October 2009.

[62] André Körbes, Giovani B Vitor, Janito V Ferreira, and Roberto de Alencar Lotufo. A proposal for a parallel watershed transform algorithm for real-time segmentation. In *Proceedings of Workshop de Visão Computacional WVC*, 2009.

[63] André Körbes, Giovani Bernardes Vitor, Roberto de Alencar Lotufo, and Janito Vaqueiro Ferreira. Advances on watershed processing on GPU architecture. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 260–271. Springer, 2011.

[64] Hans Lamecker, Thomas Lange, and Martin Seebass. Segmentation of the liver using a 3D statistical shape model. Technical Report 4-9, ZIB, Takustr.7, 14195 Berlin, 2004.

[65] Lay Khoon Lee, Siau Chuin Liew, and Weng Jie Thong. A review of image segmentation methodologies in medical image. In *Advanced Computer and Communication Engineering Technology: Proceedings of the 1st International*

Conference on Communication and Computer Engineering, pages 1069–1080. Springer International Publishing, 2015.

[66] Bing Nan Li, Chee Kong Chui, Stephen Chang, and S. H. Ong. Integrating spatial fuzzy clustering with level set methods for automated medical image segmentation. *Computers in Biology and Medicine*, 41(1):1–10, 2011.

[67] Mikhiel Lim and Ilker Hacihaliloglu. Structure-enhanced local phase filtering using l0 gradient minimization for efficient semiautomated knee magnetic resonance imaging segmentation. *Journal of Medical Imaging*, 3(4):044503, 2016.

[68] Seong-Jae Lim, Yong-Yeon Jeong, and Yo-Sung Ho. Automatic liver segmentation for volume measurement in CT images. *Journal of Visual Communication and Image Representation*, 17(4):860–875, 2006.

[69] Marius George Linguraru, John A. Pura, Vivek Pamulapati, and Ronald M. Summers. Statistical 4D graphs for multi-organ abdominal segmentation from multiphase CT. *Medical Image Analysis*, 16(4):904–914, 2012.

[70] Marius George Linguraru, Jianhua Yao, Rabindra Gautam, James Peterson, Zhixi Li, W. Marston Linehan, and Ronald M. Summers. Renal tumor quantification and classification in contrast-enhanced abdominal CT. *Pattern Recognition*, 42(6):1149–1161, 2009.

[71] Yan Liu, H. D. Cheng, Jianhua Huang, Yingtao Zhang, and Xianglong Tang. An effective approach of lesion segmentation within the breast ultrasound image based on the cellular automata principle. *Journal of Digital Imaging*, 25(5):580–590, 2012.

[72] Roberto Lotufo and Alexandre Falcão. The ordered queue and the optimality of the watershed approaches. In John Goutsias, Luc Vincent, and Dan S.

Bloomberg, editors, *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 341–350. Springer US, Boston, MA, 2000.

[73] Zhen Ma, João Manuel R.S. Tavares, Renato Natal Jorge, and T. Mascarenhas. A review of algorithms for medical image segmentation and their applications to the female pelvic cavity. *Computer Methods in Biomechanics and Biomedical Engineering*, 13(2):235–246, 2010.

[74] Ravi Malladi and James A. Sethian. Level set and fast marching methods in image processing and computer vision. In *Proceedings of International Conference on Image Processing*, volume 1, pages 489–492. IEEE, September 1996.

[75] Ravi Malladi and James A. Sethian. An O(N log N) algorithm for shape modeling. *Proceedings of the National Academy of Sciences*, 93(18):9389–9392, 1996.

[76] Ravi Malladi and James A. Sethian. Level set methods for curvature flow, image enchancement, and shape recovery in medical images. In Hans-Christian Hege and Konrad Polthier, editors, *Visualization and Mathematics*, pages 329–345. Springer Berlin Heidelberg, 1997.

[77] Ravi Malladi and James A. Sethian. A real-time algorithm for medical shape recovery. In *Sixth International Conference on Computer Vision*, pages 304–310. IEEE, January 1998.

[78] Ravi Malladi, James A. Sethian, and Baba C. Vemuri. Evolutionary fronts for topology-independent shape modeling and recovery. In Jan-Olof Eklundh, editor, *Computer Vision - ECCV'94*, volume 800 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 1994.

[79] Ravi Malladi, James A. Sethian, and Baba C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, February 1995.

[80] Ravikanth Malladi, James A. Sethian, and Baba C. Vemuri. Topology-independent shape modeling scheme. In *SPIE's 1993 International Symposium on Optics, Imaging, and Instrumentation*, pages 246–258. International Society for Optics and Photonics, 1993.

[81] Ravikanth Malladi, James A. Sethian, and Baba C. Vemuri. A fast level set based algorithm for topology-independent shape modeling. *Journal of Mathematical Imaging and Vision*, 6(2-3):269–289, 1996.

[82] Marija Marčan, Elise C. Pegg, Hemant G. Pandit, and Irina Voiculescu. Calibrating the segmentation of the knee in order to expedite patientspecific instruments. submitted.

[83] Marija Marcan and Irina Voiculescu. Unsupervised segmentation of MRI knees using image partition forests. In *Proceedings of SPIE Medical Imaging*. SPIE International Society for Optics and Photonics, 2016.

[84] Beatriz Marcotegui and Serge Beucher. Fast implementation of waterfall based on graphs. In Christian Ronse, Laurent Najman, and Etienne Decencière, editors, *Mathematical Morphology: 40 Years On, Proc. ISMM'05*, pages 177–186. Springer Netherlands, Dordrecht, 2005.

[85] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[86] Arnold Meijster and Jos BTM Roerdink. A disjoint set algorithm for the watershed transform. In *9th European Signal Processing Conference (EUSIPCO 1998)*, pages 1–4. IEEE, 1998.

[87] Bjoern H. Menze, Andras Jakab, Stefan Bauer, et al. The multimodal brain tumor image segmentation benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, 2015.

[88] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, 1990.

[89] Fernand Meyer. Topographic distance and watershed lines. *Signal processing*, 38(1):113–125, 1994.

[90] Eric N. Mortensen and William A. Barrett. Toboggan-based intelligent scissors with a four-parameter edge model. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 452–458, 1999.

[91] Alireza Norouzi, Mohd Shafry Mohd Rahim, Ayman Altameem, Tanzila Saba, Abdolvahab Ehsani Rad, Amjad Rehman, and Mueen Uddin. Medical image segmentation methods, algorithms, and applications. *IETE Technical Review*, 31(3):199–213, 2014.

[92] Stanley Osher and James A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[93] Víctor Osma-Ruiz, Juan I Godino-Llorente, Nicolás Sáenz-Lechón, and Pedro Gómez-Vilda. An improved watershed algorithm based on efficient computation of shortest paths. *Pattern Recognition*, 40(3):1078–1090, 2007.

[94] Lei Pan, Lixu Gu, and Jianrong Xu. Implementation of medical image segmentation in CUDA. In *2008 International Conference on Information Technology and Applications in Biomedicine*, pages 82–85, May 2008.

[95] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[96] Edoardo Pirovano. Applications of reinforcement learning to medical image segmentation, 2017. Computer Science Project Report.

[97] Pablo Quesada-Barriuso, Dora B. Heras, and Francisco Argüello. Efficient GPU asynchronous implementation of a watershed algorithm based on cellular automata. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 79–86, July 2012.

[98] Pablo Quesada-Barriuso, Dora B. Heras, and Francisco Argüello. Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata. *Computers and Electrical Engineering*, 39(8):2638–2655, 2013.

[99] Christian Rasch and Thomas Satzger. Remarks on the O(N) implementation of the fast marching method. *IMA Journal of Numerical Analysis*, 29(3):806–813, 2009.

[100] Jos BTM Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta informaticae*, 41:187–228, 2000.

[101] R Romero-Zaliz and JF Reinoso-Gordo. An updated review on watershed algorithms. In *Soft Computing for Sustainability Science*, pages 235–258. Springer, 2018.

[102] László Ruskó, György Bekes, and Márta Fidrich. Automatic segmentation of the liver from multi- and single-phase contrast-enhanced CT images. *Medical Image Analysis*, 13(6):871–882, 2009.

[103] Andrea Schenk, Guido Prause, and Heinz-Otto Peitgen. Efficient semiautomatic segmentation of 3D objects in medical images. In Scott L. Delp, Anthony M. DiGoia, and Branislav Jaramaz, editors, *Medical Image Computing and Computer-Assisted Intervention*, volume 1935 of *Lecture Notes in Computer Science*, pages 186–195. Springer Berlin Heidelberg, 2000.

[104] James A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[105] James A. Sethian. Theory, algorithms, and applications of level set methods for propagating interfaces. *Acta Numerica*, 5:309–395, 1996.

[106] James A. Sethian. Adaptive fast marching and level set methods for propagating interfaces. *Acta Mathematica Universitatis Comenianae*, 67(1):3–15, 1998.

[107] James A. Sethian. Fast marching methods. *SIAM Review*, 41(2):199–235, 1999.

[108] James A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 1999.

[109] James A. Sethian. Evolution, implementation, and application of level set and fast marching methods for advancing fronts. *Journal of Computational Physics*, 169(2):503–555, 2001.

[110] Neeraj Sharma and Lalit M Aggarwal. Automated medical image segmentation techniques. *Journal of Medical Physics*, 35(1):3–14, 2010.

[111] Avraham Sherman. June bug: Building and analyzing physical 3D models from medical scans. Master's thesis, University of Oxford, 2017.

[112] Rushin Shojaii, Javad Alirezaie, and Paul Babyn. Automatic lung segmentation in CT images using watershed transform. In *IEEE International Conference on Image Processing 2005*, volume 2, pages II–1270–3, September 2005.

[113] Erik Smistad, Thomas L. Falch, Mohammadmehdi Bozorgi, Anne C. Elster, and Frank Lindseth. Medical image segmentation on GPUs – a comprehensive review. *Medical Image Analysis*, 20(1):1–18, 2015.

[114] Han Sun, Jingyu Yang, and Mingwu Ren. A fast watershed algorithm based on chain code and its application in image segmentation. *Pattern Recognition Letters*, 26(9):1266–1274, 2005.

[115] Michał Świercz and Marcin Iwanowski. Fast, parallel watershed algorithm based on path tracing. In Leonard Bolc, Ryszard Tadeusiewicz, Leszek J. Chmielewski, and Konrad Wojciechowski, editors, *Computer Vision and Graphics: International Conference, ICCVG*, pages 317–324. Springer Berlin Heidelberg, 2010.

[116] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging*, 15(1):1–28, 2015.

[117] Amy S. Tidwell. Advanced imaging concepts: A pictorial glossary of CT and MRI technology. *Clinical Techniques in Small Animal Practice*, 14(2):65–111, 1999.

[118] John N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[119] Jayaram K. Udupa, Vicki R. LeBlanc, Ying Zhuge, Celina Imielinska, Hilary Schmidt, Leanne M. Currie, Bruce E. Hirsch, and James Woodburn. A framework for evaluating image segmentation algorithms. *Computerized Medical Imaging and Graphics*, 30(2):75–87, 2006.

[120] Joël Van Neerbos, Laurent Najman, and Michael Wilkinson. Towards a parallel topological watershed: first results. In Pierre Soille, Martino Pesaresi, and Georgios K. Ouzounis, editors, *Mathematical Morphology and Its Applications to Image and Signal Processing: 10th International Symposium, ISMM 2011*, Lecture Notes in Computer Science, pages 248–259. Springer Berlin Heidelberg, 2011.

[121] Vladimir Vezhnevets and Vadim Konouchine. "GrowCut" - interactive multi-label N-D image segmentation by cellular automata. In *Proceedings of Graphicon*, pages 150–156, 2005.

[122] Luc Vincent and Pierre Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.

[123] Giovani Bernardes Vitor, Janito Vaqueiro Ferreira, and André Körbes. Fast image segmentation by watershed transform on graphical hardware. In *XXX Iberian Latin American Congress on Computational Methods in Engineering - CILAMCE 2009*, volume 1, pages 1–14, November 2009.

[124] Giovani Bernardes Vitor, André Körbes, Roberto de Alencar Lotufo, and Janito Vaqueiro Ferreira. Analysis of a step-based watershed algorithm using CUDA. *International Journal of Natural Computing Research (IJNCR)*, 1(4):16–28, 2010.

[125] Giovani Bernardes Vitor, André Körbes, Roberto de Alencar Lotufo, and Janito Vaqueiro Ferreira. Analysis of a step-based watershed algorithm using CUDA. In *Nature-Inspired Computing Design, Development, and Applications*, pages 321–335. IGI Global, 2012.

[126] Björn Wagner, Andreas Dinges, Paul Müller, and Gundolf Haase. Parallel volume image segmentation with watershed transformation. In Arnt-Børre Salberg, Jon Yngve Hardeberg, and Robert Jenssen, editors, *Image Analysis: 16th Scandinavian Conference, SCIA 2009*, pages 420–429. Springer Berlin Heidelberg, 2009.

[127] Björn Wagner, Paul Müller, and Gundolf Haase. A parallel watershed-transformation algorithm for the GPU. In *Workshop on Applications of Discrete Geometry and Mathematical Morphology*, pages 111–115, 2010.

[128] Simon K. Warfield, Kelly H. Zou, and William M. Wells. Simultaneous truth and performance level estimation (STAPLE): An algorithm for the validation of image segmentation. *IEEE Transactions on Medical Imaging*, 23(7):903–921, 2004.

[129] Robin Wolz, Chengwen Chu, Kazunari Misawa, Kensaku Mori, and Daniel Rueckert. Multi-organ abdominal CT segmentation using hierarchically weighted subject-specific atlases. In Nicholas Ayache, Hervé Delingette, Polina Golland, and Kensaku Mori, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2012*, volume 7510 of *Lecture Notes in Computer Science*, pages 10–17. Springer Berlin Heidelberg, 2012.

[130] Suping Wu and Yingshuai Hu. Parallelization research on watershed algorithm. In *International Conference on Automatic Control and Artificial Intelligence (ACAI 2012)*, pages 1524–1527, March 2012.

[131] Ziji Wu and John M. Sullivan. Multiple material marching cubes algorithm. *International Journal for Numerical Methods in Engineering*, 58(2):189–207, 2003.

[132] Liron Yatziv, Alberto Bartesaghi, and Guillermo Sapiro. O(N) implementation of the fast marching algorithm. *Journal of Computational Physics*, 212(2):393–399, 2006.

[133] Varduhi Yeghiazaryan. Feature identification on CT images using the fast marching method. Master's thesis, University of Oxford, 2013.

[134] Varduhi Yeghiazaryan and Irina Voiculescu. An overview of current evaluation methods used in medical image segmentation. Technical Report CS-RR-15-08, Department of Computer Science, University of Oxford, Oxford, UK, 2015.

[135] Varduhi Yeghiazaryan and Irina Voiculescu. The use of fast marching methods in medical image segmentation. Technical Report CS-RR-15-07, Department of Computer Science, University of Oxford, Oxford, UK, 2015.

[136] Varduhi Yeghiazaryan and Irina D. Voiculescu. Automated 3D renal segmentation based on image partitioning. In *Proc. SPIE 9784, Medical Imaging 2016: Image Processing*, page 97842E. SPIE International Society for Optics and Photonics, 2016.

[137] Futoshi Yokota, Toshiyuki Okada, Masaki Takao, Nobuhiko Sugano, Yukio Tada, Noriyuki Tomiyama, and Yoshinobu Sato. Automated CT segmentation of diseased hip using hierarchical and conditional statistical shape models. In Kensaku Mori, Ichiro Sakuma, Yoshinobu Sato, Christian Barillot, and Nassir Navab, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, volume 8150 of *Lecture Notes in Computer Science*, pages 190–197. Springer Berlin Heidelberg, 2013.

[138] Hui Zhang, Sharath Cholleti, Sally A. Goldman, and Jason E. Fritts. Meta-evaluation of image segmentation using machine learning. In *IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition, volume 1, pages 1138–1145. IEEE, 2006.

[139] Hui Zhang, Jason E. Fritts, and Sally A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Computer Vision and Image Understanding*, 110(2):260–280, 2008.

[140] Yu Jin Zhang. A survey on evaluation methods for image segmentation. *Pattern Recognition*, 29(8):1335–1346, 1996.

[141] Yu Jin Zhang. A review of recent evaluation methods for image segmentation. In *Sixth International Symposium on Signal Processing and its Applications*, volume 1, pages 148–151. IEEE, 2001.

[142] Yu Jin Zhang. Image segmentation evaluation in this century. *Encyclopedia of Information Science and Technology. Beijing, Tsinghua University, China*, pages 1812–1817, 2009.

[143] Hongkai Zhao. A fast sweeping method for eikonal equations. *Mathematics of computation*, 74(250):603–627, 2005.

[144] Hongkai Zhao. Parallel implementations of the fast sweeping method. *Journal of Computational Mathematics*, pages 421–429, 2007.