Austrian Marshall Plan Foundation Research Report

Real-Time Processing of Ultrasonic Video Streams

conducted within the master program
Information Technology & Systems Management
Salzburg University of Applied Sciences

submitted by:

Peter Friedrich Keuschnigg, BSc

supervised by:

FH-Prof. Univ.-Doz. Mag. Dr. Stefan Wegenkittl
Salzburg University of Applied Sciences

Distinguished Professor Gabor T. Herman
City University of New York

New York City, July 2013

Acknowledgement

First and foremost, I would like to thank the Austrian Marshall Plan Foundation which has funded the research for the masters thesis *Real-Time Processing of Ultrasonic Video Streams* [37]. The Marshall Plan Scholarship allowed me to write the thesis during a research stay at the Graduate Center of the City University of New York.

Furthermore, I would like to thank Dr. Stefan Wegenkittl (Salzburg University of Applied Sciences) for the thesis supervision and for the invaluable support during the creation phase of the thesis, as well as for the assistance during the preceding research and development project on which the thesis is based.

Moreover, I would like to thank Distinguished Professor Gabor T. Herman (Graduate Center of the City University of New York) for the support during my research stay at the City University of New York. He has not only organized a workplace for me in the middle of this incredible city, but also has taken the time to guide me during the creation process of the thesis.

Abstract

This Marshall Plan Scholarship report is a shortened version of the masters thesis *Real-Time Processing of Ultrasonic Video Streams* [37] which has been created during the research stay at Graduate Center of the City University of New York. It presents real-time approaches for image processing of ultrasonic video streams, especially focusing on the problem of object tracking. Existing top-down tracking approaches are being classified and evaluated in terms of real-time techniques. The approach of parallel processing for computation time reduction of object tracking pipelines is being evaluated considering multi-device pipelines such as GPU computing and their technology bottlenecks. Additionally, an information reduction approach is being analyzed in terms of achievable speedup and remaining tracker performance, covering downsampling, regions of interest and sparse processing methods.

Contents

List of Figures

List of Algorithms

List of Abbreviations

API	Application Programming Interface
CAMSHIFT	Continously Adapting Mean-SHIFT
CPU	Central Processing Unit
CT	X-ray Computed Tomography
FIR	Finite Impulse Response
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
OBIA	Object Based Image Analysis
PDF	Probability Density Function
ROI	Region Of Interest
SM	Streaming Multiprocessor
SNR	Signal to Noise Ratio
SP	Streaming Processor
US	UltraSonic or UltraSound

1 Chapter 1

Introduction

Computer aided image processing offers a wide range of algorithms to process medical images. According to Jähne [35], digital image processing comprises for example averaging operations, edge detection or neighborhood based operations. However, depending on the imaging modality, huge amounts of data may be involved. The processing of one image with a pipeline comprising multiple algorithms therefore could take several seconds or minutes, depending on the algorithms and the computational power of the underlying hardware. Nevertheless, time could be a critical factor when processing medical images. Especially in the field of video stream processing, the involved single images or *frames* need to be processed in a way that guarantees a desired framerate τ. This imposes an upper limit τ^{-1} on the processing time that is available for a single frame. Encoding and compression algorithms such as the H.264[1] standard try to overcome this limitation by reducing the amount of stream data. However, medical procedures often require the preservation of small details that may represent key factors for diagnosis and subsequent treatment. Thus, lossy compression algorithms cannot be considered for the input to the given problem. Therefore, this thesis has its focus on alternative strategies for real-time processing of medical video data.

Video streams that are generated by an ultrasonic (US) imaging modality rise additional challenges. Regarding to Birkfellner [8], the US modality has lesser resolution, tends to include artifacts and present a lesser overall image quality compared to other imaging modalities such as x-ray computed tomography (CT). Additionally, US devices often apply image enhancements such as stated by Edelman [22] before providing the output data to a viewing platform or image processing framework. The applied image enhancements tend to differ across various suppliers of US devices. Therefore, frameworks for image processing of US frames face various different tasks such as local image processing (e.g. sharpening, edge detection) or global image processing (e.g. histogram stretching).

1.1 Motivation and Problem Statement

Several different pipeline models for object tracking have been presented by various authors in the last years. Popular applications comprise video surveillance systems or facial recognition for cameras. However, very little research has been focusing on medical

[1] http://www.itu.int/rec/T-REC-H.264

video streams such as generated by the US modality. Whilst many of the object tracking approaches for face tracking and movement detection have proven to be real-time capable, US video stream analysis has shown to be more computationally expensive due to heavy preprocessing requirements. Although some of the existing approaches comprise real-time capability, only a very limited amount of publications present research on common factors that influence the speed of such a pipeline and that may be used for future applications. The key motivation of this thesis therefore is to find patterns in speedup and performance enhancement in existing approaches and implementations and to test their applicability on US video streams. Therefore, different processing pipelines, methods and strategies are being examined in terms of their suitability under real-time constraints.

The key problem that is being analyzed in this thesis, is the question of how image processing for US video streams can be realized to meet real-time requirements. Those requirements include the frame processing time τ^{-1}, the memory consumption or the robustness of the approach which represents also its determinability. In order to provide answers to the research problem, it has been split up into several sub-problems. Those comprise research on state of the art object tracking pipelines and image processing algorithms, a classification of the different methods in terms of parallel processing and the exploration of an information reduction approach in order to limit the amount of processed data.

Real-time processing of medical video data requires different concepts with respect to the applied algorithms for various medical problem cases. Even a simple approach such as parallel processing requires a detailed insight into the utilized algorithms and the bottlenecks of the underlying technologies such as central processing unit (CPU) multithreading or massively parallel processing on a graphics processing unit (GPU) which are being addressed by this thesis.

1.2 Thesis Structure

The report is structured as follows: Chapter 2 comprises background information on the ultrasonic modality and a selection of algorithms that are being used throughout this thesis. Common image and video stream generation is described for the US modality along with the achievable image quality, including possible artifacts and noise. Also, the technology of general purpose graphics processing unit computation (GPGPU) and its applications are being presented to the reader.

Chapter 3 analyzes various object tracking pipelines that have been presented over the last years. Also, an attempt for classification of these pipelines is being made and a selection of pipelines is being presented in detail.

In Chapter 4, image processing algorithms are being analyzed in terms of their degree of complexity. The analysis is based on the perspective of parallel processing. Time expenditure and memory consumption and separability of the algorithms are being analyzed in terms of how the algorithms can be split up in parallel processable parts. Also bottlenecks of current

parallel processing technologies such as GPGPU are being evaluated.

Chapter 5 deals with a different approach to the real-time problem, focusing on information reduction for speedup. A selection of information reduction methods is being presented and their effect on the complexity of an object recognition pipeline is being analyzed.

Chapter 6 summarizes the results and observations.

Chapter 2

Background Information

This chapter contains background information on the subjects of this thesis that is essential for the understanding of the presented key concepts. In the first section, the generation of US images comprising the emergence of image artifacts is being presented. The second section comprises information about GPGPU. The last section introduces a mathematical notation that is being used throughout the remainder of this thesis.

2.1 Ultrasonic Modality

As has been stated by Hoskins et al. [33], US imaging can be subdivided into multiple modes. The A-mode thereby stands for a 1-dimensional line scan where a single transducer emits soundwaves into the body, recoding the echos. The B-mode is a 2-dimensional mode using a linear array of transducers that allow for a planar scan. The most important mode concerning this thesis is the M-mode, that utilizes a successive A or B-mode scan, creating an infinite discrete stream of US frames. According to Hoskins et al. [33], modern US systems use improved concepts such as the Doppler mode that allows for a visualization of blood flow. Sending the sound waves at different angles provides a basis for 3-dimensional US scans. However, due to the fact that only the M-mode in combination with B-mode imaging allows for an infinite 2D video stream, which is also the most commonly used mode in diagnostic ultrasonography, this thesis focuses only on the B-M mode [33].

During the generation of the US images, the physics of the emitted soundwaves allow for the emergence of artifacts in the produced images. According to Edelman [22], these artifacts comprise reverberation effects, ring down artifacts, mirrored objects, reflections, acoustic shadows, enhancement artifacts and statistical noise. All of those artifacts affect the US image quality and increase the difficulty of object segmentation as well as object tracking. Hence, they increase the amount of preprocessing that is needed and for this reason their presence may also increase the required processing time t_{frame} for a single frame.

Regarding to Aldrich [1], *reverberation* effects are caused by sound that is bouncing between tissue boundaries before returning to the receiver and they emerge as equally spaced lines along the ray direction. Resonating air bubbles can produce additional sound that irritates the system and leads to unusually bright areas at the bottom of these bubbles, called *ring down* artifacts. The *enhancement* artifact is similar to the ring down effect but caused by sound traveling through various mediums with different attenuation. *Reflections*

and *mirror images* both create duplicated structures at misleading positions in the resulting images. Those artifacts may pose the most critical threat in medical terms and can only be detected by utilizing additional knowledge (e.g. kidney stones can only appear inside the kidney region). The *attenuation* or shadow artifact produces darker regions below strongly attenuating objects such as kidney stones. However, this effect may not only harmfully modify the image but can also be used for identifying such objects. Speckle *noise* is a random, deterministic and statistically distributed interference pattern in US images that affects the visible image quality and complicates object tracking through movement detection. Some of the potentially emerging artifacts are depicted in Figure 2.1 [1][22].

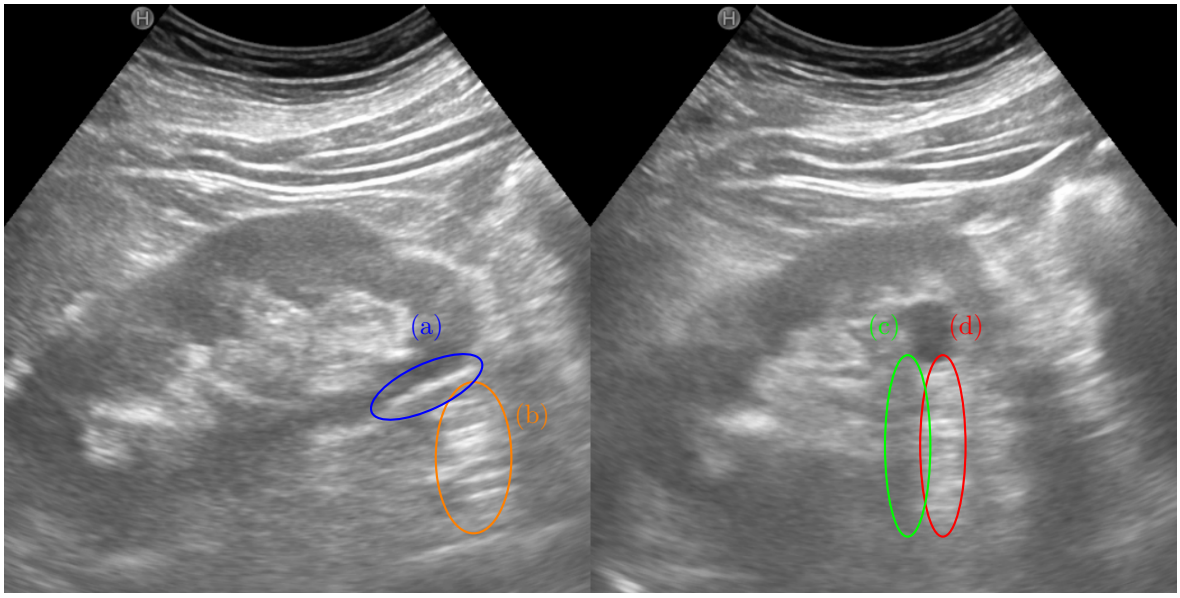

Figure 2.1: Artifacts in US images of the kidney region, showing ring down (a), reverberation (b), attenuation (c) and enhancement (d) effects.

2.2 GPGPU

This section provides a short introduction into the history and principles of general purpose graphics processing unit (GPGPU) computations and massively parallel processing technologies. According to Kirk an Hwu [38], 3D graphics pipeline hardware became available on workstations and PCs in the time between the early 1980s to the late 1990s. Since then, the performance of graphics accelerators has increased continuously. Kirk and Hwu describe GPGPU as intermediate step between using the GPU as graphics accelerator and GPU computing. The term GPGPU therefore refers to using the GPU to solve computational problems via graphics application programming interfaces (API) such as DirectX and OpenGL. However, in order to solve programming problems, algorithms had to be translated into native graphics operations such as vector calculations. The subsequent development of programmable shader processors with increased instruction memory and related APIs such

as NVIDIA CUDA[1] or OpenCL[2] is referred to as GPU computing. Though, for the sake of simplicity, the term GPGPU is being used throughout this thesis also for GPU computing [38].

Concerning its architecture the GPU is a designated device for massively parallel computations. Regarding Sanders and Kandrot [60], applications embrace medical imaging tasks such as 3D US video generation, fluid dynamics computation and environmental science tasks such as molecular simulations. The architecture of a modern GPU is according to Kirk and Hwu [38] built on an array of highly threaded streaming multiprocessors (SM) that allow for a huge number of parallel computations on the internal streaming processors (SP). Each of those SMs is connected to a small but fast accessible cache memory and to the large global memory of the GPU. Along with a high memory bandwidth, the SMs outperform a modern CPU in terms of parallelism. The architecture of a typical G80 GPU is shown in Figure 2.2. However, the GPGPU technology also comprises drawbacks. Compared to the fast internal

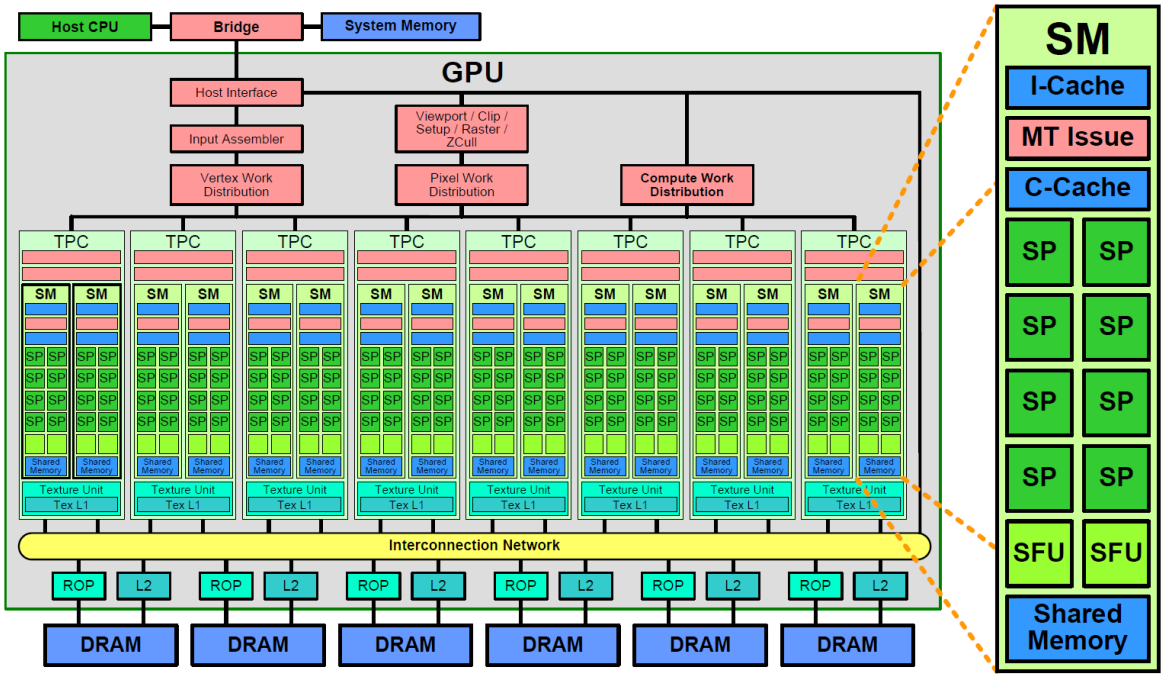

Figure 2.2: Architecture of a G80 GPU according to [63].

memory access, the transfer of data between the main memory of the PC and the global graphics memory is slow. Typically, the program instructions or *kernels* have to be transferred from the CPU to the GPU and are being scheduled to the SMs at run-time. Thereby the CPU is often referred to as *host* and the GPU is referred to as *device*. In the best case scenario, maximal one transfer of the input data and one transfer of the output data between host and device should be designated [38][60].

Another fact is that the program code has to be redesigned and programmers have to apply different concepts in order to take advantage of the parallel computation power.

[1] http://www.nvidia.com/object/cuda_home_new.html
[2] http://www.khronos.org/opencl/

According to McCool et al. [50] two different strategies for scalable parallelism are available. *Data parallelism* thereby describes the task of splitting the data into multiple parts that can be processed by different threads. Multiple threads may execute identical or different procedures at the same time. Regarding the definition of McCool et al. the number of threads increases with the size of the data. Kowalik and Puzniakowski [40] also use the term *single program multiple data* to describe this procedure. The other strategy is *functional decomposition*, which describes the process of splitting a task into sub-tasks that do not depend on each other and letting them process the data in parallel. Functional decomposition is also known as *multiple programs multiple data* [40][50].

2.3 Notation

This section introduces the notation that is being used throughout the thesis. Let $S = (V, \pi)$ be a digital space that is defined by a grid V and an adjacency relation π between elements $g \in V$. Let $I = (V, \pi, \varphi)$ be a digital image that is defined by the digital space (V, π) and a function φ that maps elements $g \in V$ to a field such as $\mathbb{R}$.

A digital 2D grayscale *frame* in an infinite stream of video data is denoted as

$$f = (V, \pi, \varphi_f)$$

where

$$V = \{g_1, g_2, ..., g_{|V|}\}, g_i \in \mathbb{Z}^2$$

represents a set of 2D pixels g_i defined by coordinates $(g_{i,x}, g_{i,y})$. The following neighborhood relations N_4 and N_8 are also defined for any point $g \in \mathbb{Z}^2$:

$$N_4(g) = \{h \in \mathbb{Z}^2 \, : \, |h_x - g_x| + |h_y - g_y| \leq 1\}$$

$$N_8(g) = \{h \in \mathbb{Z}^2 \, : \, max(|h_x - g_x|, |h_y - g_y|) \leq 1\}$$

similar to a definition by Couprie and Bertrand [19]. The edge-adjacency between the pixels in the square grid is represented by

$$\pi = \{(g, h) \, : \, g \in V, h \in V, h \neq g, h \in N_4(g)\}$$

and a mapping of pixels into a grayscale value is defined by

$$\varphi_f \, : \, V \rightarrow [0, 255]$$

An ordered set of frames in time is denoted as $F = (f_1, f_2, ..., f_n), n \in \mathbb{N}$ accordingly. For neighborhood based image filters such as a mean filter or a median filter, a square neighbor-

hood with radius r is represented by

$$S(g,r) = \{h \in \mathbb{Z}^2 \, : \, max(|h_x - g_x|, |h_y - g_y|) \leq r\}$$

and a rectangular neighborhood with r_x and r_y respectively is represented by

$$R(g,r_x,r_y) = \{h \in \mathbb{Z}^2 \, : \, |h_x - g_x| \leq r_x, |h_y - g_y| \leq r_y\}$$

A *segmentation* of V in a frame f is a function σ that maps each $g \in V$ into a set of non-empty distinct objects $O = (o_1, o_2, ..., o_n \, : \, o_i \neq \emptyset, o_i \neq o_j, i \neq j), n \in \mathbb{N}$ and is denoted as

$$\sigma_f \, : \, V \rightarrow O$$

3 Chapter 3

Object Tracking Pipeline Analysis and Classification

Object tracking in US video streams has been continuously researched, as has been shown by various authors [53, 13, 2, 54] in the past few years. In 2008, Nascimento and Marques [53] presented a model based robust approach for shape tracking in US images. Carneiro et al. [13] proposed a method for automatic detection and measurement of fetal anatomies in the same year. However, object tracking is not only an issue in medical image processing but instead comprises a wide range of applications. According to Challa [14], some of the most popular object tracking applications include air space monitoring, video surveillance, weather monitoring and face tracking. Air space monitoring is used to keep track of flying objects in controlled air spaces for security reasons or national defense. Weather monitoring may be used to generate forecasts. Video surveillance and face tracking are related in the security sector but may also be used for e.g. manufacturing processes. Additionally, the military industry provides purpose for object tracking, e.g. for targeting systems. In 2006, Yilmaz et al. [76] have presented a survey about object tracking, comprising a list of applications. According to them, object tracking is used for human identification, automated surveillance, video indexing, human-computer interaction, traffic monitoring and vehicle navigation. Systems for human identification may identify persons by reference to gait or facial features and could be used for enterprise security or airport surveillance. Video indexing could be utilized for automated annotation and search of videos in large database systems. Gesture recognition and eye gaze tracking are both subcategories of human-computer interaction with state of the art applications in the sector of video games. Traffic monitoring systems may be utilized for congestion prevention or police actions. Vehicle navigation keeps track of traffic signs and the course of the road and may be used to warn drivers about dangerous conditions. Nevertheless, Challa and Yilmaz et al. do not claim to provide complete lists but try to give an insight into the large variety of applications. Beside popular applications, there is also literature on specialized applications such as presented by Lin et al. [46] involving object tracking in geo-coordinates [14][76].

All the mentioned applications provide for different scenarios for the object tracking systems. The complexity of object tracking is influenced by many factors. As has been stated in Chapter 2 of this thesis, US video streams comprise a large number of artifacts that lower the image quality. Whereas video surveillance applications may deal with a static background and moving target objects, the US video application includes at least patient

movement such as respiration which conducts a non-linear transform on the frames, thus leading to a separate movement of both, background and target objects. Additionally, the transducer may be moving as well, leading to background movement, partially visible objects or objects moving completely outside the capture region. Since a B-scan of the US modality is a planar scan, a movement of the transducer may position the sectional plane such that it does not intersect with the target object anymore whereas the background remains mostly constant. Many of the presented applications also involve creating a model of the desired object. Although, this can be a simple task for applications such as player tracking in sports casts using static templates, deformable objects in US videos require dynamic models, thus complicating the modeling task. Other factors that exacerbate the tracking process comprise occlusion effects when tracking multiple objects. Player objects in sports video streams may be subject to occlusion as described by Yilmaz et al. [76]. However, these occlusion effects may also occur in US videos, depending on the application.

Despite the differences between the listed applications, common characteristics can be extracted. All of them start with a set of input frames F_{input} in a finite or infinite video stream. According to the type of application and the involved artifacts, pre-processing may be required e.g. to remove noise, hence this step is optional. It is followed by a segmentation of objects $O_{seg} = \{o_1, ..., o_n\}, n \in \mathbb{N}$ and a subsequent extraction of features such as edges or histograms according to the objects. These two stages may be repeated multiple times in order to refine the results. Object based image analysis (OBIA) as described by Blaschke et al. [9] could be seen as multiple iterations of segmentation and feature extraction. Afterward, the extracted object descriptions and related features are submitted to a model that is used to select relevant objects and predict their future locations. In a last step, the predicted objects are being marked in output frames $F_{output} \subseteq F_{input}$. A visualization of this abstract process can be found in Figure 3.1. This pipeline is similar to the image engineering framework as has been stated by Patil and Deore [56], consisting of image processing, image analysis and image understanding. Pre-processing or image processing thereby represents the lowest layer, followed by a middle layer consisting of image segmentation, object detection and feature extraction algorithms which are part of image analysis. The highest level, named image understanding, eventually comprises the modeling and prediction tasks.

However, according to Nummiaro et al. [55] this pipeline represents only one of two different classes of trackers, namely *bottom-up* approaches. There is also the class of *top-down* trackers which utilize a sort of reverse approach to the bottom-up trackers. Thus, a top-down approach generates object hypotheses from an initial model first and tries to verify them by using the input images F_{input} before marking the objects in the output images F_{output} and updating the model. A visualization of a top-down pipeline is shown in Figure 3.2.

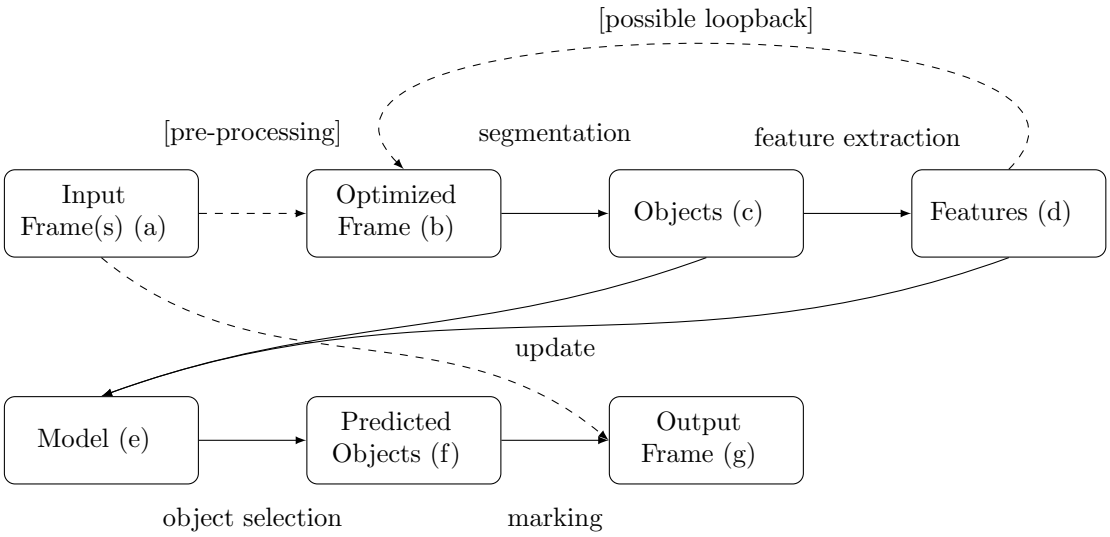

Figure 3.1: Generic bottom-up object tracking pipeline, showing the processing steps between the input frame (a) and the output frame (g) including optional stages such as pre-processing (a→b).

3.1 Detailed Pipeline Analysis

The abstract generic pipeline is instanced each time in a different way by a wide range of approaches for each of the fields of application. In 2012, Kolarov et al. [39] have made an attempt to categorize various object tracking approaches. Therein, they followed a basic scheme that has been introduced by Yilmaz et al. [76] before. They have split the problem of object tracking in a first stage by the form of object representation and they distinguish between shape based and appearance based representation. According to Kolarov et al. [39] and Yilmaz et al. [76], shape based representation includes models such as object contours, skeletal models, geometric shapes and points. Object contours thereby refer to the boundary of an object with the object silhouette inside that region. Together with skeletal representations which are area reduced versions of contours, they are suitable for tracking complex nonrigid objects. Point based representations mostly rely on the centroids of objects and therefore can be used only to track relatively small objects. Related tracking methods therefore could be based on techniques such as Kalman filtering [45, 34], particle filtering [29, 58, 49] or active contour methods [44, 30]. Furthermore, Kolarov et al. [39] and Yilmaz et al. [76] mention probability densities, templates, active appearance models and multiview appearance models among the appearance based object representation. Whereas probability densities comprise only parametric densities, nonparametric densities or histogram information, templates add geometric shape or contour information of a single view. Therefore these models are suitable only for non-deformable objects whose appearance does not change considerably during tracking. Active appearance models and multiview active appearance models need a training phase for simultaneous modeling of the object shape and appearance in a single view or

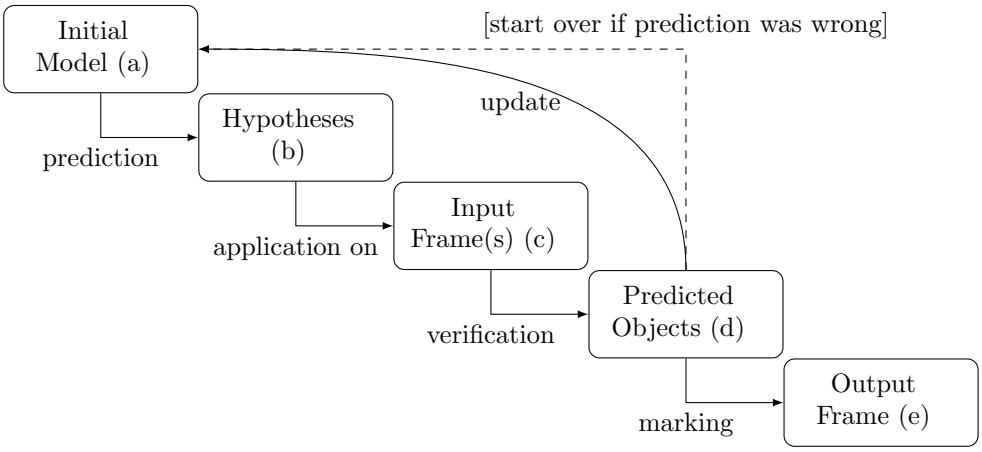

Figure 3.2: Generic top-down object tracking pipeline, showing the processing steps between the initial model (a) and the output frame (e). Therein, hypotheses (b) are being generated and applied to the input frames (c) in order to predict objects and to verify them (d). Finally, the objects are being marked in output frames (e).

multiple views. Associated tracking methods comprise sparse variants such as based on point descriptors (e.g. SIFT, SURF, KLT) [6, 68] or dense manifestations such as histogram based trackers (e.g. mean shift, CAMSHIFT) [17, 10, 25]. In Figure 3.3 the basic attempt for a categorization is visualized in tree form [39, 76]. Nevertheless, this attempt is not the only way of a categorization of object tracking approaches. Yang et al. [75] for example distinguish between deterministic and stochastic methods [39][76].

3.1.1 Filtering

The different approaches to object tracking comprise pipelines that include various filters which are being applied to one or more of the frames $f_i \in F_{input}$ or to outcome of intermediate steps of the pipeline such as $o_i \in O_{seg}$. Those filters can be of various type such as image processing filters or machine learning algorithms. Especially if pre-processing is required, image processing algorithms are involved. Authors such as Jähne [35] or Bankman [4] categorize image processing algorithms into *global* and *local* applicable filters. In global image processing thereby every output value is dependent on all values of the image, thus comprising algorithms such as histogram generation or mean value computation. Let $g_i \in V$ be an indexed pixel in the grid of a frame f consisting of n pixels. The computation of the mean value μ of all pixels in V can be defined as

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\varphi_f(g_i)$$

Thus, the algorithm generates one output value that depends on all pixels of the image. Considering local image processing methods, every output value g'_i depends only on a single pixel g_i or on a neighborhood, e.g. $N_8(g_i)$. Local image processing comprises algorithms such as a intensity changes or mean filtering. A change of the image intensity by a value of Δ can

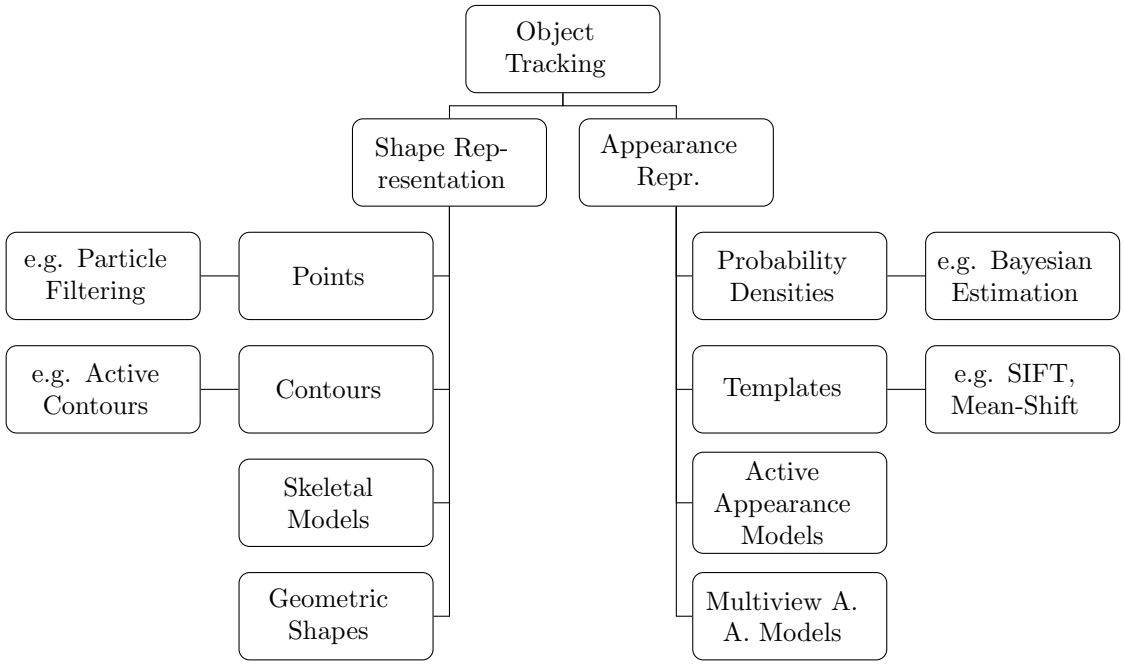

Figure 3.3: Categorization of object tracking techniques as proposed in [39] and [76].

be defined as

$$\forall g_i \in V \,:\, \varphi_f(g_i') = \varphi_f(g_i) + \Delta$$

and hence only dependent on one observation. A neighborhood based mean filter can be defined as

$$\forall g_i \in V \,:\, \varphi_f(g_i') = \frac{1}{|N_8|} \sum_{g_j \in N_8(g_i)} \varphi_f(g_j)$$

and therefore depends on a set of pixels that can range from a single pixel g_i to the whole set of pixels V. Both types require different strategies when considering parallel processing, as is being discussed in Chapter 4 [35][4].

The following subsections present three different top-down approaches to object tracking that are being used throughout this thesis as examples of object tracking pipelines in contrast to the bottom-up approach. Since there is no general way of a categorization and since there is a vast amount of different approaches to the subject of object tracking, this part of the thesis only considers three approaches that have been popular in terms of publications over the last years. The chosen approaches have also been used in the field of ultrasound video streams already and may therefore be suitable for the given problem of nephrolith tracking. The following subsections introduce the approaches of mean-shift tracking, particle based tracking and active contour tracking, focusing especially on the pipeline models behind.

3.1.2 Mean-Shift Tracking

The mean-shift method has been presented originally by Fukunaga and Hostetler [26] in 1975. It is based on the idea of a non-parametric estimation of a density gradient that

is related to a probability density function (PDF) which is not well defined in terms of the available knowledge about the function or its form. The algorithm represents an iterative procedure of mode seeking as has been stated in 1999 by Comanicu and Meer [18]. They have used the algorithm for image segmentation in the spatial domain of gray-scale images.

In 2003, Comanicu et al. [17] have presented an approach for kernel based object tracking which utilizes the mean-shift method. This approach falls into the section of appearance representation based object tracking and into the subsection of template based representation as has been stated in Section 3.1. As this method is based on a density gradient, it is suitable for gradient based optimization. They have chosen the histogram for representing a non-parametric density estimation. A *target model* has been defined as $\hat{q} = \{\hat{q}_u\}_{u=1...m}$ of m-bin histograms and target candidates at position y have been defined as $\hat{p}(y) = \{\hat{p}_u(y)\}_{u=1...m}$. In order to measure similarity between the model and a candidate, Comanicu et al. [17] have proposed a metric $d(y)$ that is shown in Equation 3.1 based on the Bhattacharyya coefficient [7].

$$d(y) = \sqrt{1 - \sum_{u=1}^{m} \sqrt{\hat{p_u}(y)\hat{q_u}}} \tag{3.1}$$

By using this metric, the distance between the model and a candidate can be minimized as they have described in [17]. According to them, the tracking pipeline consists of 6 steps which are visualized in Figure 3.4.

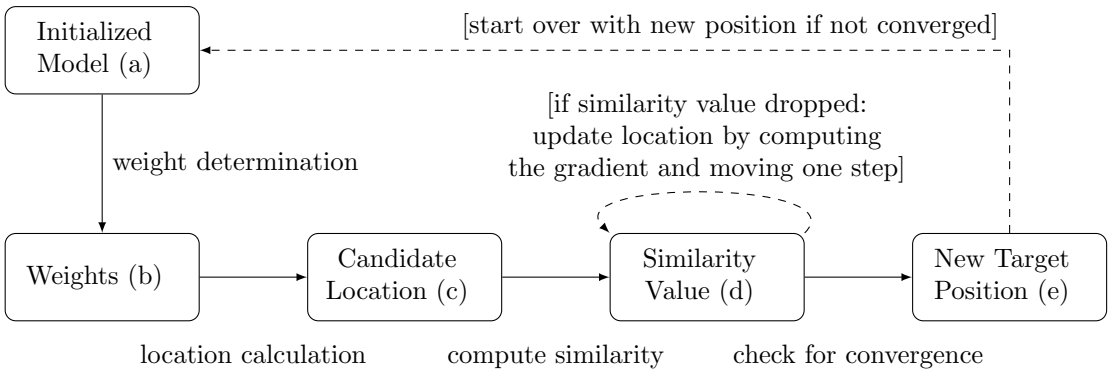

Figure 3.4: Mean shift pipeline according to [17]. (a) Initialization of the model with position $\hat{y}_0$. (b) Derivation of weights. (c) Search for the next location for a candidate $\hat{y}_1$. (d) Computation of similarity between candidate and model. (e) As long as similarity is lower than at the original position, set $\hat{y_1} \leftarrow \frac{1}{2}(\hat{y}_0 + \hat{y}_1)$. (f) Leave the pipeline in case of convergence or start over with $\hat{y}_0 \leftarrow \hat{y}_1$ otherwise.

There have been adaptions such as the continuously adapting mean-shift (CAMSHIFT) algorithm that has been presented by Bradski [12] for special applications such as face tracking. Thereby, the original mean-shift algorithm is embedded into a parent pipeline that adjusts a region of interest (ROI) regarding to the mean-shift output and updates the model accordingly. A simplified version of the CAMSHIFT pipeline is shown in Figure 3.5.

Mean-shift and CAMSHIFT have been extended by various authors in order to improve

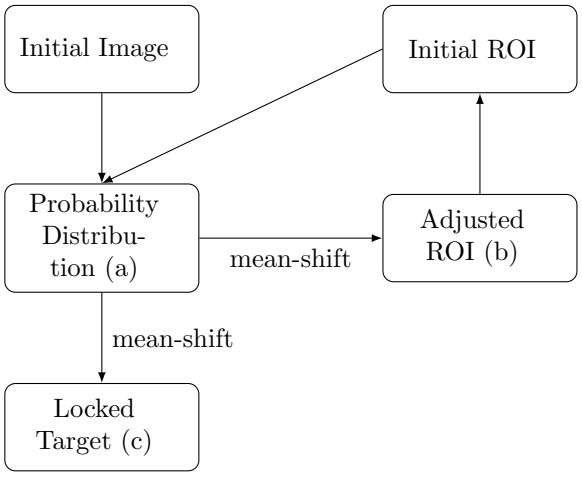

Figure 3.5: CAMSHIFT pipeline according to [12]. This mean-shift extension comprises an additional ROI that is used for the calculation of the PDF (a). After the mean-shift step, the output includes an updated ROI (b) and the locked target (c).

the processing time or the robustness of the approach. Bleiweiss and Werman [10] for example have added the use of image depth information to the histogram information in order to make the approach more robust to effects of occlusion. Santner et al. [61] have merged it with a tracking-by-detection approach in order to avoid the drifting problem of on-line learning. Exner et al. [25] have proposed an efficient GPU implementation of the CAMSHIFT algorithm. Moreover, the algorithms have also been adapted to specific applications such as video surveillance, as has been done by Hsia et al. [34]. Thereby, the CAMSHIFT approach has been embedded into another pipeline that is responsible for pre-processing and creating time-difference images, as can be seen in Figure 3.6. It can be seen in the depicted pipelines that the usage of the mean-shift approach can lead to complex pipeline systems, depending on which extension has been chosen. The complexity also depends on the application and whether there is pre-processing required. Nesting of pipelines additionally introduces increased complexity for programmers, software designers and software architects. Hidden internal dependencies might slow down the overall approach when not considered during the development of the tracking application.

3.1.3 Particle Filter Based Tracking

Particle filter based tracking approaches use according to the categorization in Section 3.1 a point based object representation. Therefore, those approaches are clearly separated from appearance based approaches such as mean-shift, that has been discussed in Section 3.1.2. Regarding to Kroese et al. [41], particle filtering is a subset of Monte Carlo methods or more specifically a subset of sequential Monte Carlo methods. It has applications in combinatorial problems, Bayesian likelihood estimation and complex simulations. Especially the application of Bayesian likelihood estimation is relevant for object tracking.

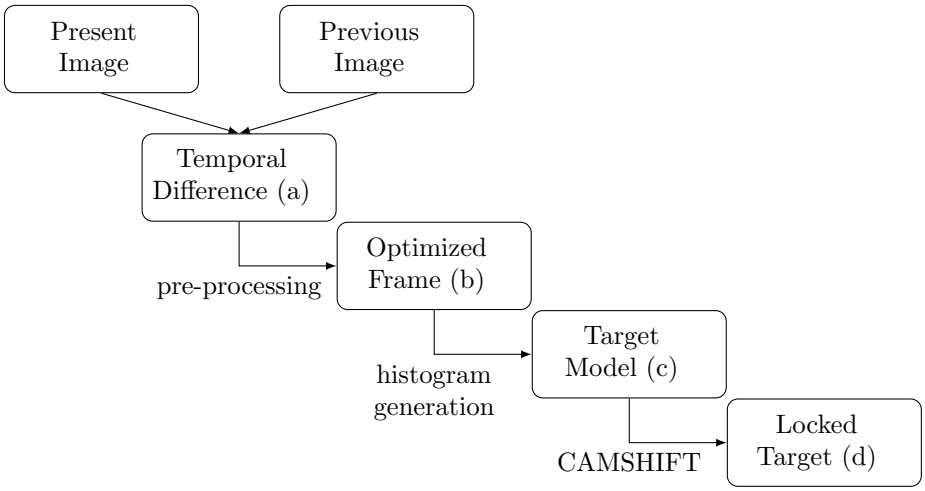

Figure 3.6: Video surveillance pipeline based on CAMSHIFT according to [34]. Special characteristics of this pipeline are the calculation of the temporal difference (a) and the use of pre-processing (b). In (c) the histogram is being generated and used for the CAMSHIFT algorithm to lock the target (d).

Thereby, particles which represent random values of a state-space variable are being generated, used as a description in a model of the tracker and finally weighted to determine their likelihood of being a representative particle. Therefore, a set of particles approximately describes a PDF of the target. Since that PDF approximation can be used in other tracking approaches such as in the mean-shift method as an alternative to histogram based target models, particle filter based tracking is not only a standalone tracking approach [41].

Kroese [41] has defined a sequential Monte Carlo algorithm in a generic form that samples approximately from a given density function. It consists of four major steps: initialization, importance sampling, selection and a stopping condition. There are also more advanced versions of such a particle filter but this simple form is sufficient for showing the pipeline behind this approach. The initialization step comprises a, possibly random, sampling of a measurement vector from the initial distribution such as the frame histogram. In the next step, a second vector is sampled from an importance distribution and weights are being calculated as well as normalized. The importance sampling is a critical step in the pipeline, leading to more particles being sampled in important regions. However, unbiasedness of the estimator is being preserved by including the importance distribution not directly but in the form of normalized weights. Importance sampling theory has been discussed extensively in the literature, e.g. by Kroese [41]. A practical implementation from Lozano and Otuska [49] for face tracking utilizes a feature map of a 3D human face model as an importance distribution. Using this weighted sample, replacement particles are then being sampled from the initial distribution. Next, the whole process is being repeated beginning from the second step, until a specific number of iterations has been made. A simple pipeline that represents the initialization part of the tracker visualizes this algorithm in Figure 3.7 [41].

In 2008, Lozano and Otsuka [49] have presented a combination of a particle filter and

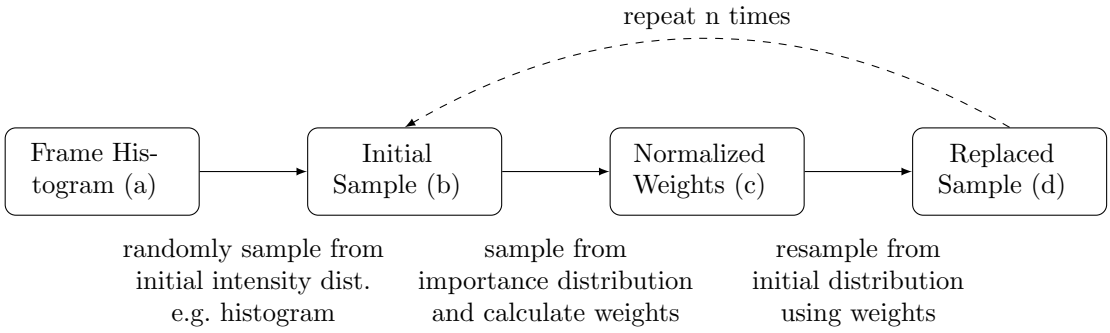

Figure 3.7: Sequential Monte Carlo pipeline according to the algorithm of [41] as an initialization for the tracker. An initial distribution (a), e.g. frame histogram, as well as an importance distribution (b), e.g. feature map of a template model, are being sampled. Subsequently, weights (c) are being calculated and normalized before the initial distribution is being resampled using the weights (d).

an active appearance based model for a real-time multiple face tracking system. In their approach, they have utilized the massively parallel computation power of GPGPU in order to realize the particle filter. Thus, they have shown that a particle filter can be designed efficiently multithreaded. They also state that particle filters require high arithmetic throughput and have low communication and storage costs, which makes them suitable for parallel computation. Their tracker comprises three steps: an initialization phase, the tracking itself and a subsequent output generation. The initialization stage is performed continuously in a separate thread on the CPU. It is responsible for scanning the incoming frames for new faces, utilizing a Viola and Jones boosting algorithm [70] for detection. It also extracts a sparse description template that is used as initial model in the tracking stage. The tracking stage itself comprises the particle filter and runs partially on the CPU and on the GPU. Lozano and Otsuka [49] claim that only the weighting part of the algorithm has been moved to the GPU because it is the computationally most expensive part. The output generation phase eventually comprises an averaging of the best results of the tracker before marking the output. A visualization of this implementation is given in Figure 3.8 [49].

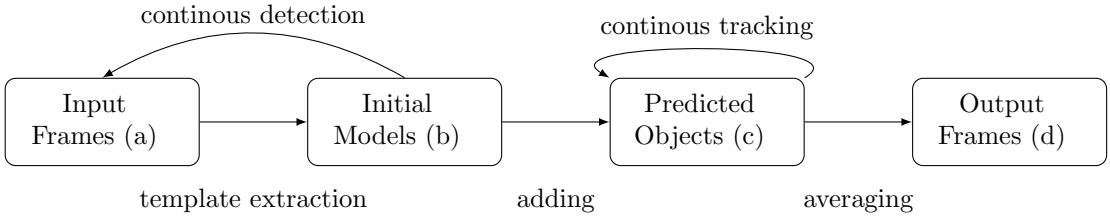

Figure 3.8: Particle filter based tracking pipeline according to [49]. Input frames (a) are being continuously searched for new faces and models (b) are being extracted to initialize continuous tracking of those objects (c) before averaging the results and marking them in output frames (d).

In 2003, Nummiaro et al. [55] have presented another application of particle filters

for object tracking. Instead of an active appearance representation such as has been used by Lozano and Otsuka [49], they proposed a combination of a particle filter and the Bhattacharyya coefficient [7] which is commonly being used in mean-shift approaches. They claimed to have brought together efficient tracking components. Following that, in 2004 Deguchi et al. [20] proposed a combination of a particle filter and the whole mean-shift algorithm. Instead of an improved performance however, they have searched for a more robust tracking solution. This approach includes tracking with the mean-shift method and subsequent avoidance of local maxima by using a small number of samples of a particle filter [55][20].

3.1.4 Active Contour Tracking

Active contour based tracking falls into the category of contour based object representation regarding the categorization in Section 3.1. Active contour models or *snakes* haven been mentioned first by Kass et al. [36] in 1988. They have used energy minimization as a framework for designing low level energy functions which provide in their local minima a set of alternative solutions available to high level image processing. Snakes are commonly being used for edge detection, line detection and contour detection as well as for motion tracking of those objects. Gunn and Nixon [30] claim that snakes incorporate a global continuity and curvature based edge detection combined with a local edge strength feature. Leymarie [44] describes snakes as deformable curves or contours that are composed of elastic materials making them resistant to stretching and bending. According to Kass et al. [36], snakes represent a general technique to match a deformable model to a given frame by minimizing its energy function. The original snake energy term E_{snake} of a contour $v(s)$ as shown in Equation 3.2 represents according to Kass et al. a controlled continuity spline under the influence of internal and external forces.

$$E_{snake}(v(s)) = \int_0^1 [E_{int}(v(s)) + E_{img}(v(s)) + E_{con}(v(s))]ds \tag{3.2}$$

Thereby, E_{int} represents the internal energy of the snake, consisting of two weighted terms that make the snake behave like a membrane and like a thin plate respectively. The image forces E_{img} ensure that the snake moves towards line, edge or contour features of the image. The external constraint forces E_{con} eventually locate the snake near the desired local minimum. The local optimization task can be carried out by various different algorithms such as gradient descent or downhill simplex methods, thus presenting an opportunity for introducing parallel computable strategies. A simplified pipeline for active contour based tracking is shown in Figure 3.9 [36].

As has been stated by Leymarie [44], active contour tracking comprises the problem of selecting an initial position of the snake in the first frame. Sargin et al. [62] referred to this problem as *tracing* in contrast to the adjustment of the contour points which they refer to as *tracking*. The tracing can be solved by either utilizing an automatic segmentation of the desired contour in the first frame or by using a user defined shape. In subsequent frames, the

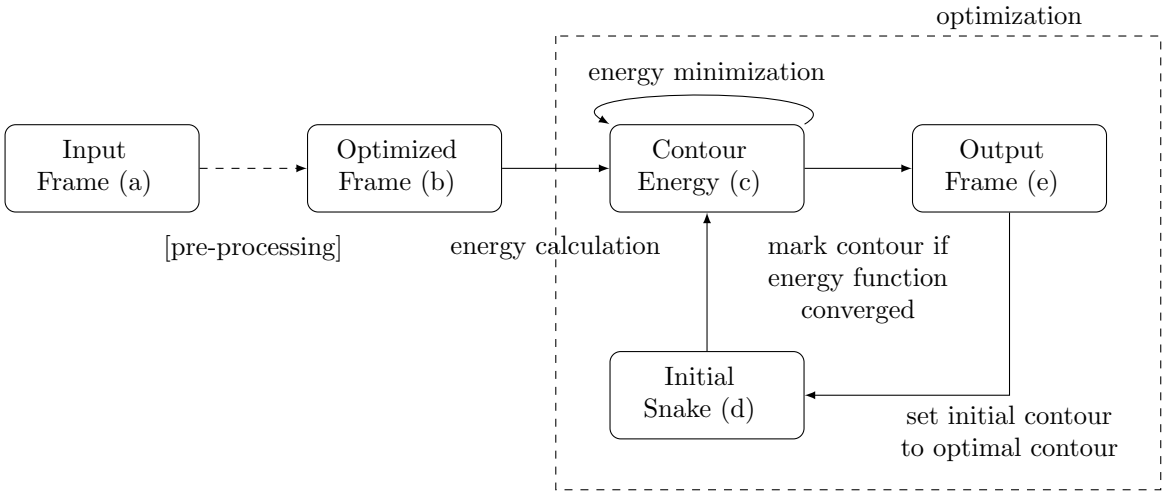

Figure 3.9: Active contour based tracking pipeline. Starting with an input frame (a) and an initial contour (d), the contour energy (c) is being calculated and minimized by an optimizing algorithm. Beforehand, the input frame may be subject to pre-processing. When the energy function converged to a local minimum, the contour is being marked in an output frame and set as the initial contour for subsequent frames.

optimal snake position of the preceding frame may be used as an initial position. However, large deformations between frames may lead to a situation where the snake loses its focus around the desired object and drifts off. Robust implementations therefore may be required to repeatedly validate the initial position. Gunn and Nixon [30] have presented a robust snake implementation which is based on a dual active contour procedure. Their approach utilizes two contours, one of them inside and one outside the feature of interest. In a first stage, both contours are searching for a energy minimum where their energy function converges and they behave stationary. A second stage then applies additional forces in order to alternately push the contours towards each other until both contours have found the same optimal solution. The authors claimed to provide therewith an approach that is robust to weak local minima.

There have been various criticisms and extension of the original formulation of active contours. In 2011, Sargin et al. [62] have criticized the applicability of active contour based tracking to open contours such as often found in biomedical image analysis. They proposed an alternative by mapping contour positions to a hidden Markov model. Shih and Rose [65] presented a similar approach for tracking cell contours, including a merging and splitting detection. Gai and Stevenson [27] addressed in their paper the level set method as a variation of the snake model in combination with geodesic active contours. A level set based real time tracker has also been presented by Shi and Karl [64], utilizing a fast level set implementation. However, with reservations most approaches can still be mapped into the simple pipeline model presented in Figure 3.9.

4 Chapter 4

Parallel Processing

In order to speed up the processing of a video stream pipeline, a common approach is to use massively parallel computations such as carried out by GPGPU. In this chapter, parallel executable versions of image processing filters are being introduced, their computational complexity is being analyzed and effects on object tracking pipelines are being discussed subsequently.

The parallel approach has been shown in the past by Webb [73] to work well on local image processing filters but also to provide issues for global image filters. According to Webb, some global image operations cannot be executed in parallel due to a specific order of execution that is required for a correct result. However, he also showed that reversible global operations such as the computation of image histograms can be executed in parallel by using a split and merge strategy. Thereby the global image region is split into multiple sub-regions. Each of the parallel processing threads is then being executed on one of the sub-regions and the results are being merged subsequently. A visualization of the split and merge strategy for a small pixel grid is shown in Figure 4.1 . However, there are limitations for the split and

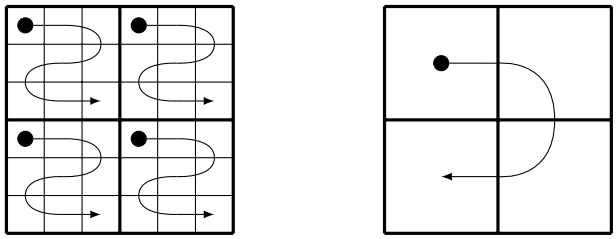

Figure 4.1: Split and merge strategy as presented in [73] for a 6x6 image and 4 threads, showing the processing after the split (left) and the merge operation (right).

merge strategy. If there is a number of threads $n_{threads}$ and a image to process consisting of a number of values n_{values} such that $n_{threads} = n_{values}$, the split and merge strategy will have no effect on the processing time. This due to the fact that the merging operation in that specific case would have at least a complexity of $\mathcal{O}(n_{threads})$. Hence, $n_{threads}$ has to be chosen carefully and related to both, thread capacity and the number of operations to perform. Webb states that for a sequential combination of partial results, the optimal number of threads is

$$n_{threads} = \sqrt{\frac{t_{process} \cdot n_{values}}{3 \cdot t_{combine}}}$$

where $t_{process}$ is the time that is required for processing one result value and $t_{combine}$ stands for the time that is required for combining two results [73].

4.1 Parallel Image Processing

The following examples show how the processing time and memory consumption of image filters are being affected by the application of a parallel computing design. In order to proof the concept of Webb [73] for image processing algorithms, local, neighborhood-local and global image filters are being analyzed. Both, time and memory consumption are being measured by using the big O notation, e.g. $\mathcal{O}(n)$, to describe their limiting behavior. However, the big O notation is being used in this chapter to present theoretical growth behavior of the algorithms for a number of threads $n_{threads} \to \infty$ and a number of pixels $n_{pixels} \to \infty$, in order to show the independence of the kernel functions from both variables. All examples of algorithms are given in pseudo-code. Therefore, no constraints of specific programming languages or bottlenecks of specific technologies such as GPGPU or of their underlying hardware, e.g. G80 GPU, have to be considered. Local image processing is being represented by an intensity alteration function. The neighborhood-local image processing examples comprise mean filtering and median filtering operations. Global image processing algorithms are being represented by histogram generation. All of these filters represent simple low-level algorithms that are being used often in a pre-processing step of an object tracking pipeline. In case of a subsequent processing step depending on results of preceding stages and therefore presenting a strictly iterative behavior, these parts of a pipeline by definition cannot efficiently be parallelized. Thus they are not suitable for a parallel processing approach. Depending on the structure of the algorithm, some high-level machine learning procedures or prediction steps may present such a behavior. In the following subsections, presented algorithms are named *kernel* functions if they are designed to be executed in a multi-threaded environment, given an index of the current position to be processed by a single thread.

4.1.1 Local Image Processing

Local image processing comprises all filters that are being applied to an image pixel by pixel and that do not depend on other surrounding pixels in order to complete their task. This applies to many filters such as intensity adjustments, binary thresholding or the application of arithmetic and logic operators. As a simple example, intensity adjustments have been chosen for this section.

Intensity Adjustments The basic formulation of a global adjustment of the intensity of a frame f can be defined as $\forall g_i \in V \, : \, \varphi_f(g_i') = \varphi_f(g_i) + \Delta_g$. A related sequential algorithm is shown in Algorithm 4.1. The complexity of the shown algorithm concerning processing time can be considered linearly growing $\mathcal{O}(n)$ for frames comprising n pixels to process. The memory consumption can be considered constant $\mathcal{O}(1)$ due to the fact that no additional

space is needed when increasing the number of pixels.

When the same algorithm is being designed for parallel computations, the loop over all pixels is being replaced by an index parameter that specifies the position to work on for the current thread. Such a kernel is shown in Algorithm 4.2. The memory consumption of the algorithm is still constant $\mathcal{O}(1)$, but the time complexity has been decreased. Assuming that there is a number of threads $n_{threads}$ such that $n_{threads} = n_{pixels}$, the growth of the processing time can be stated as constant $\mathcal{O}(1)$.

Algorithm 4.1 Adjusting the image intensity

```
1: procedure AdjustIntensity(f, Δ_g)        ▷ f is the frame that should be altered by Δ_g
2:     for all g_i ∈ V do
3:         φ_f(g_i) = φ_f(g_i) + Δ_g
4:     end for
5: end procedure
```

Algorithm 4.2 Adjusting the image intensity (kernel)

```
1: procedure AdjustIntensityKernel(f, Δ_g, i)        ▷ i represents the current index
2:     φ_f(g_i) = φ_f(g_i) + Δ_g
3: end procedure
```

4.1.2 Neighborhood Image Processing

Neighborhood image processing comprises filters that are being applied to an image pixel by pixel but for every pixel they depend on a group of other pixels in a defined neighborhood. Hence, the access to single pixels is not as simple as with local image filters, but has to be controlled. If the filters are designed to work in-place on an image, the values of the neighborhoods have to be copied to another place in order to prevent distorting the computation of subsequent iterations by overwriting values. In terms of parallel processing, simultaneous read and write access to the same memory blocks has to be prevented as well. Neighborhood image filters comprise for example mean filtering, median filtering, adaptive histogram equalization or morphological filters such as dilation and erosion filters. For this section, mean filtering and median filtering have been chosen to represent simple instances of those filters.

Mean Filter Mean filtering computes the average value in a given neighborhood of a pixel g_i and assigns the computed average to the pixel itself. A sequential algorithm for mean filtering of a 2D frame f by using 8-connected neighborhoods $N_8(g_i)$ is shown in Algorithm 4.3.The given simple mean filter could be extended by a parameter r that defines the radius of the considered neighborhoods. However, the simple version is sufficient for stating that the processing time grows linearly with the number of pixels, which can be expressed as $\mathcal{O}(n)$. The inner loop in this case can be neglected as it has a constant processing time $\mathcal{O}(1)$ for the given neighborhood size $|N_8|$. The memory consumption in this example grows linearly with

the number of pixels $\mathcal{O}(n)$ because the original pixel values have to be stored separately from the output in order to produce correct results.

A version of the same filter, designed for massively parallel computation is given in Algorithm 4.4.The memory consumption and processing time of the kernel function can both be considered constant $\mathcal{O}(1)$. However, regarding the overall process, the growth of memory consumption is still $\mathcal{O}(n)$. Additionally, the decrease of the growth of processing time to a constant value is only valid under the assumption of $n_{threads} = n_{pixels}$.

Algorithm 4.3 Mean filtering

```
1: procedure MeanFilter(f)
2:     for all g_i ∈ V do
3:         sum = 0
4:         for all g_j ∈ N_8(g_i) do
5:             sum = sum + φ_f(g_j)
6:         end for
7:         φ_f(g_i) = sum/|N_8|
8:     end for
9: end procedure
```

Algorithm 4.4 Mean filtering (kernel)

```
1: procedure MeanFilterKernel(f, i)        ▷ i represents the current index
2:     sum = 0
3:     for all g_j ∈ N_8(g_i) do
4:         sum = sum + φ_f(g_j)
5:     end for
6:     φ_f(g_i) = sum/|N_8|
7: end procedure
```

Median Filter Median filtering computes the median value of a given neighborhood of a pixel g_i and assigns the computed value to the pixel itself. Although, the basic structure of this filter is similar to mean filtering, a major difference is the median value computation which comprises sorting the values of the neighborhood before picking the middle one as the output value. A sequential algorithm for median filtering of a 2D frame f by using 8-connected neighborhoods $N_8(g_i)$ is shown in Algorithm 4.3.Whereas the memory consumption is equal to the mean filter $\mathcal{O}(n)$, the growth of the processing time is additionally dependent on the sorting algorithm that is being used. In the case of a constant sized neighborhood, the complexity of the sorting may be considered constant as well, leading to a linear overall complexity $\mathcal{O}(n)$. However, if the size of the neighborhood is depending on a radius r, the use of a slow sorting algorithm such as bubble-sort brings in a complexity of $\mathcal{O}(r^4)$, leading to an overall growth of $\mathcal{O}(n \cdot r^4)$.

The design of a kernel version for median filtering is similar to the mean filtering kernel and stated in Algorithm 4.6.The kernel version has the same effect of removing the factor n from the complexity. It has to be stated that the given version of median filtering is a

primitive example. There has been research on temporal optimized median filters such as multivariate versions [5].

Algorithm 4.5 Median filtering

```
1: procedure MedianFilter(f)
2:     for all g_i ∈ V do
3:         X = Sort(N_8(g_i))
4:         φ_f(g_i) = φ_f(X_median)
5:     end for
6: end procedure
```

Algorithm 4.6 Median filtering (kernel)

```
1: procedure MedianFilterKernel(f, i)
2:     X = Sort(N_8(g_i))
3:     φ_f(g_i) = φ_f(X_median)
4: end procedure
```

4.1.3 Global Image Processing

Global image processing comprises filters that for every output value depend on all input values. Therefore, these algorithms cannot be split easily into separately processable parts. However, the split and merge strategy can be applied to decrease computation time.

Histogram generation When computing a m-bin histogram of a given frame f, the whole range of possible pixel values is equally split into m parts. Each of the pixels in V is then being assigned to the correct bin by computing its bin index and counting the number of occurrences for each index. A sequential version of histogram generation is presented in Algorithm 4.7.The temporal complexity of this algorithm is linearly dependent on the number of pixels $\mathcal{O}(n)$ whereas the memory consumption is constant $\mathcal{O}(1)$.

Algorithm 4.7 Histogram generation

```
1: function ComputeHistogram(f, m)
2:     for all g_i ∈ V do
3:         index = ComputeIndex(g_i, m)
4:         hist_index = hist_index + 1
5:     end for
6:     return hist
7: end function
```

A parallel processable version of the histogram generation using the split and merge structure is shown in Algorithm 4.8. The parallel executable version splits the frame f into a set of non-overlapping regions $R \subseteq V$ that can be processed in parallel, creating a set of partial histograms H. After that, the partial results are being combined to the final histogram. As can be seen, the temporal complexity of computing a partial histogram is now

only linearly dependent on the size of the partial regions $\mathcal{O}(|R|)$. The temporal complexity of the combination operation is linearly dependent on the number of splits $\mathcal{O}(n_{split})$, assuming that the number of bins for the histogram is constant. This leads to an overall growth of processing time of $\mathcal{O}(\frac{n}{n_{split}} + n_{split})$, which is in the case of no splits at all $\mathcal{O}(n)$. In case of the maximum number of splits $n_{split} = n$ it is also $\mathcal{O}(n)$, but in between these two cases the temporal complexity is lower. However, the growth of memory consumption has increased to $O(n_{split})$ due to the fact that the partial results have to be stored.

Algorithm 4.8 Histogram generation (kernel)

```
1: function ComputePartialHistogram(R, m)
2:     for all g_i ∈ R do
3:         index = ComputeIndex(g_i, m)
4:         parthist_index = parthist_index + 1
5:     end for
6:     return parthist
7: end function
8: function CombinePartialHistograms(H, m)
9:     for all parthist_i ∈ H do
10:        for all m ∈ parthist_i do
11:            hist_m = hist_m + parthist_{i,m}
12:        end for
13:    end for
14:    return hist
15: end function
```

4.2 Multi-Device Pipelines

Concerning massively parallel computations, Ahmdahl's law [3] and Gustavson's law [31] have to be considered. Both formulations consider the maximum possible speedup of a program when using parallel processing with $n_{threads}$ parallel threads or processors, the proportion $p_{parallel}$ of the code that can be parallelized and accordingly $p_{serial} = 1 - p_{parallel}$ of serial computations. Ahmdahl's law as stated in Equation 4.1 thereby proposes an upper limit to the potential speedup that is defined by the serial proportion.

$$s(n_{threads}) = \frac{1}{p_{serial} + \frac{p_{parallel}}{n_{threads}}} \tag{4.1}$$

The plot in Figure 4.2 visualizes Ahmdahl's law for three different configurations of $p_{parallel}$ and p_{serial}. Thus, according to Ahmdahl, even if it would be possible to design one of the pipelines of either Section 3.1.2, Section 3.1.3 or Section 3.1.4 to be executed in a massively parallelized environment such as in GPGPU, and even if there would be a unlimited number $n_{threads} \to \infty$ of SMs available, the serial proportion of the code would limit the possible speedup to a finite factor. Concerning a practical implication of this law, it can be stated

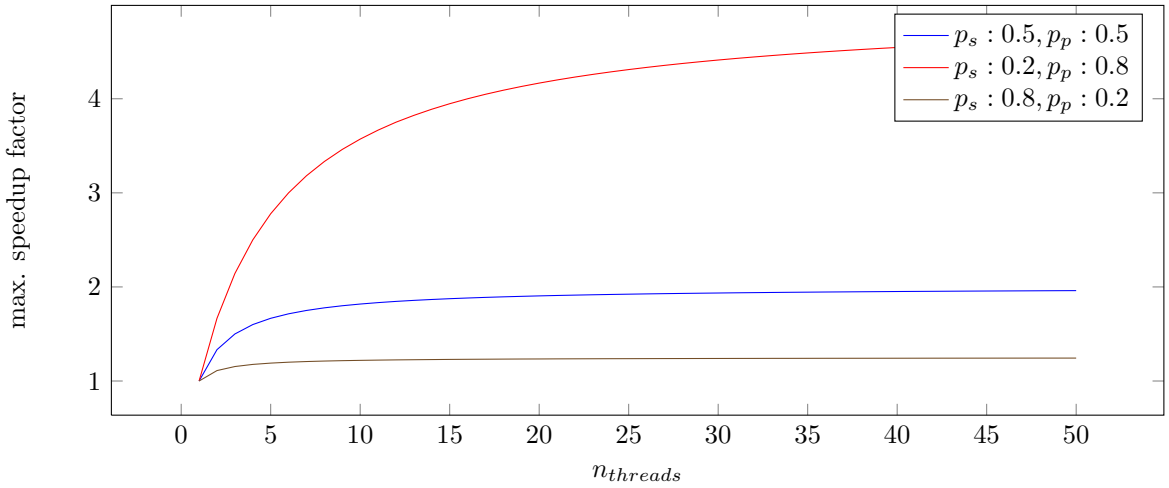

Figure 4.2: Ahmdahl's law, showing the maximum achievable speedup with distributions of serial and parallel proportions of (1): 50%,50% (2): 20%,80% and (3): 80%,20%.

that a single object tracking pipeline comprising a fixed amount of data as input can only be speeded up to a specific factor, regardless of the number $n_{threads}$ available.

Gustavson's law as presented in Equation 4.2 is a criticism of Ahmdahl's law. In contrast to the latter, it is based on the assumption of the parallel proportion of the work $p_{parallel}$ varying linearly with $n_{threads}$.

$$s(n_{threads}) = n_{threads} - p_{serial} \cdot (n_{threads} - 1) \tag{4.2}$$

The plot in Figure 4.3 visualizes Gustavson's law for the same three configurations of $p_{parallel}$ and p_{serial}. Thus, according to Gustavson, increasing the number of available threads is rais-

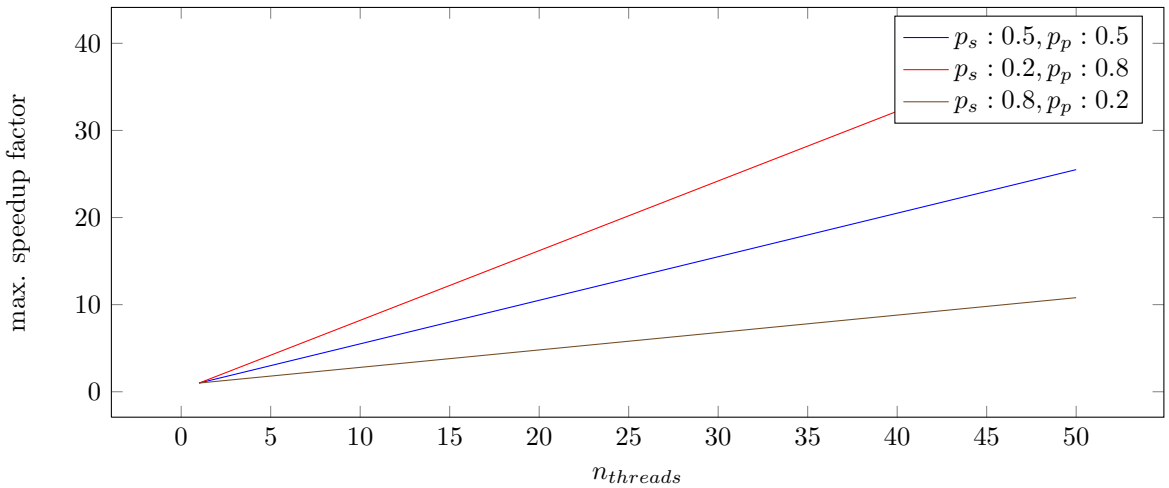

Figure 4.3: Gustavson's law, showing the maximum achievable speedup with distributions of serial and parallel proportions of (1): 50%,50% (2): 20%,80% and (3): 80%,20%.

ing the amount of data that can be processed in the same time. Regarding the presented object tracking pipelines, this would imply that if it is possible to design one of those pipelines such that it can be executed in a massively parallel environment and if the amount of concurrently processed frames rises linearly with the number of available threads, the theoretical speedup factor for $n_{threads} \to \infty$ is not limited. Of course, regarding practical GPGPU environments, the amount of SMs is always a finite number and regarding a practical video stream source such as a US modality, the framerate τ is also limited. Nevertheless, considering Gustavson's law, a practical implication of a massively parallel environment would be that a increase of τ can simply be compensated by increasing $n_{threads}$ [3][31].

From a practical point of view, Ahmdahl's law can be utilized to estimate the maximum possible speedup of an partially parallel executable application. Using an estimation of the serial and parallel proportions of the code, the finite theoretical speedup factor can be approximated by calculating

$$\lim_{n_{threads} \to \infty} s(n_{threads}) = p_{serial}^{-1}$$

However, the computed value can only serve as an indication of the possible speedup as p_{serial} can only be estimated approximately and as the computation does not include technological factors. Practical implications of Gustavson's law on the other hand may comprise design instructions for parallel processable pipelines. In order to prepare a partially parallel executable pipeline to be efficiently scalable with the underlying hardware, the amount of input data has to be varied with the amount of available threads. In case of bottom-up object tracking in video streams, the size of frames is typically fixed but the amount of parallel processed frames can be adjusted. In order to allow for parallel computed frames, a design guideline could comprise removing internal dependencies of the pipeline such as a required order of frames.

Bottlenecks There are many types of parallel processing, e.g. multi-threading on a CPU, cluster computing, cloud computing or GPGPU. Each of those technologies comprises specific bottlenecks that have to be considered when designing algorithms or computation pipelines. As this thesis focuses on GPGPU which has been presented in Section 2.2, the specific bottlenecks of this technology have to be considered for an optimized object tracking pipeline.

Kirk and Hwu [38] state in their book that on the one hand, the potential performance of many-core GPUs is continuously rising from around 100 gigaflops in 2005 to approximately 1 teraflops in 2009, but on the other hand these numbers represent not necessarily achievable application speeds. They also state that as of 2009 the floating point calculation throughput of GPUs was at a ratio of 10 to 1 compared to CPUs . The G80 architecture as has been shown in Section 2.2 comprises around 128 SPs and modern GPUs have multiplied this number. Thus, one important issue is memory bandwidth. The G80 architecture comprised a 86.4 GB/s bandwidth to its *global memory* which represents the largest but also slowest accessible internal memory. Data can be sent from the CPU to the GPU only by using

this global memory. However, the communication bandwidth for such a transfer is only around 8 GB/s for the G80. There is also a hardware managed L1 register memory and a user managed shared memory per SM with significantly more bandwidth up to 1 TB/s for modern GPUs. However, these memory parts are comparably small in size and hence have to be used with purpose. Typically, a huge block of data is being transferred from the host memory to the device global memory first. Next, the data is split into smaller portions, e.g neighborhoods when concerning image processing, which are being transferred to the individual shared memory. After that, the data portions are being processed by the SPs before writing them back to the global memory. As a last step, the entire chunk of data is being transferred back to the host memory. The transfer process is visualized in Figure 4.4. In this process, the transfer between the host and the device can clearly be considered a

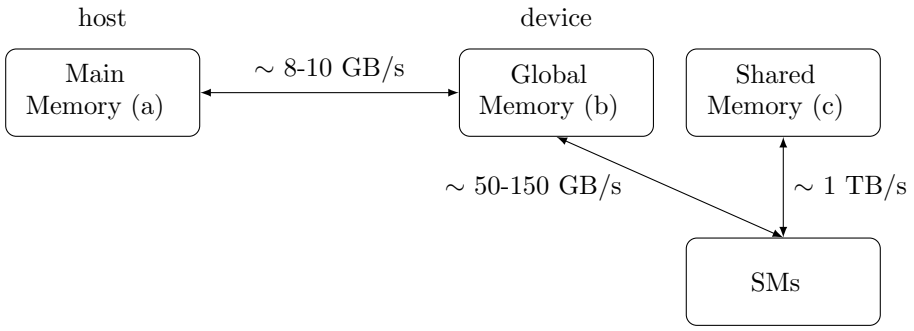

Figure 4.4: CPU to GPU data transfer pipeline between host and device memory with descriptions of available bandwidth.

bottleneck that limits the maximum achievable speedup [38].

Bottlenecks and limitations of the GPU acceleration approach have been researched in the literature in the past. Vuduc et al. [71] for example examined the boundary between CPU and GPU performance. They stated that in addition to the computational costs of an algorithm, there are costs of moving data to the GPU and costs of *data reorganization* if the optimal data structure for parallel computation differs from the data structure for serial computation. Stratton et al. [67] have analyzed the continuously changing architecture of modern GPUs, including programming concepts such as *tiling* or *binning* that have been introduced with newer generations of GPUs. They state that improvements of up to additional factor of 5 have been made by introducing these concepts. Conversely, this means that bottlenecks can also emerge from the coding point of view, when using a former generation of GPUs and APIs accordingly. Ryoo et al. [59] have been working on automated optimizations that avoid performance loss through misguided resource allocation [71][67][59].

For the case of object tracking, this raises the question of how to distribute computations between host and device such that memory bandwidth based bottlenecks have the lowest possible impact on the performance.

4.3 Pipeline Approaches Revised

This section comprises a summary of the observations that have been made in the previous sections. Furthermore, real-time versions of the top-down object tracking pipelines presented in Section 3.1 are being analyzed in terms of the observations that have been made.

Summarizing the previous sections, the following statements can be made about parallel processing concerning an object tracking pipeline:

- Massively parallel computations are a common way of speeding up image processing tasks.
- Not all image processing tasks can efficiently be performed in parallel. However, local and neighborhood-local operations can always be designed for parallel execution. Global image operations may sometimes be executed in parallel when using strategies such as split and merge.
- When designing algorithms for parallel processing, the maximum theoretically achievable speedup is limited by the remaining serial proportion of the code, as stated by Ahmdahl's law. In designing the optimal distribution of parallel and serial proportions of a pipeline in a practical setting, technology bottlenecks such as memory access bandwidth and time consumption of bulk data transfers between associated processors have to be taken into account. These variables may decrease a significant theoretical parallel processing speedup to an insignificant value or even to increase time consumption.
- The theoretical speedup factor can be further increased with the number of available processors when there is a way of processing multiple frames concurrently, as proposed in Gustavson's law. In a practical setting, parallel processing efficiency gains can be increased by minimizing internal dependencies in object tracking pipelines, e.g. through a frame order independent redesign of all algorithms, allowing for a parallel computation of multiple frames.

Regarding the top-down object tracking approaches that have been presented in Section 3.1, there has been notably work on real-time solutions so far. Considering for example the mean-shift approach, in 2010 Santner et al. [61] have presented an optical flow based mean-shift object tracker. However, they have not changed the mean-shift pipeline itself, but instead they have used a dense optical flow field as an input for the mean-shift procedure. As has been stated by Santner et al., the optical flow vectors can be computed massively parallel on the GPU whereas the tracking itself is being carried out on the CPU. For the purpose of robustness, the have added two components which supervise the mean-shift tracking approach and prevent it from drifting off. An online random forest classifier thereby overrules the mean-shift tracker if both results are not overlapping. A normalized cross-correlation procedure prevents the online random forest approach from making too many wrong updates. Both of the additional parts allow according to Santner et al. for GPU implementations due to their parallel structure of computation [61].

Also in 2010, Exner et al. [25] have presented a GPU based robust version of the CAMSHIFT extension to the mean-shift procedure. In their approach, they utilize accumulated image histograms in order to generate back projected probabilities and subsequent image moments. All of these computations have been implemented as massively parallel executable algorithms and are being executed on the GPU in their approach. The CPU is then only responsible for initializing the tracking procedure, uploading the current frames to the GPU and recomputing the ROI that is used in the CAMSHIFT approach. Thus, Exner et al. have limited the required transfers of data between the host and the device. The current frame data and the computed ROI are being uploaded to the GPU only once per frame, and there is only the image moments and the matching probability being downloaded to the CPU. A simplified version of the pipeline of Exner et al. is visualized in Figure 4.5 [25].

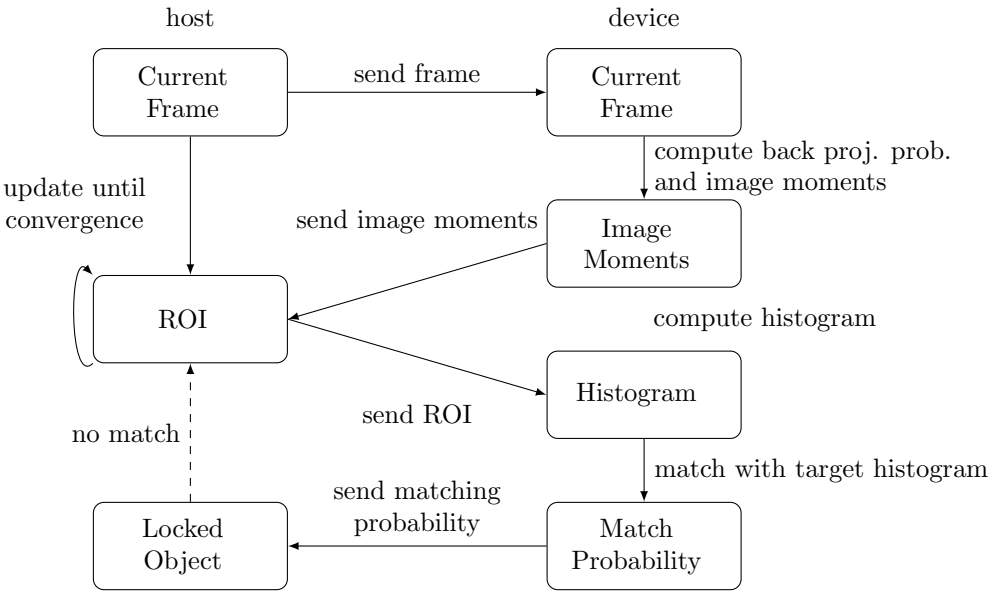

Figure 4.5: Data transfers of a GPU based CAMSHIFT approach, according to [25].

In 2008 Lozano and Otuska [49] have proposed an approach of particle filter based tracking that utilizes the massively parallel computation power of a GPU. However, instead of moving the majority of the computations to the GPU as has been done by Exner for the CAMSHIFT approach, they have decided to develop only the weight calculation of the Monte Carlo simulations for the GPU. According to them, this part is the most suitable for massively parallel computations. Hence, each new frame requires only one transfer of its data to the GPU and a transfer of the computed weights back to the CPU. A selection of particles has to be sent to the GPU only for the initialization of the procedure. Thus, the approach of Lozano and Otuska is focused on developing only the computationally most costly part of the pipeline for parallel execution and also on using the least amount of transfers possible [49].

Although, there has been research on real-time tracking before, it seems that most of it is focused on top-down approaches rather than on bottom-up tracking. Since top-down

approaches are based on prediction of a future state that is related to a preceding state, they comprise per definition an internal dependency. The analyzed papers have not included research on the effect of this dependency on the application of Gustavson's law. Hence, an open question is if it is possible to process multiple frames at different stages of the tracking pipeline at the same time, leading to a maximum workload for the hardware.

5 Chapter 5

Information Reduction

In order to speed up the processing time of an object tracking pipeline, there are alternatives available to the approach of a multithreaded execution. One of these alternatives is the technique of reducing the amount of input data that has to be processed by the algorithms. Thereby, the available information is being reduced to a form where only the minimal required information for tracking is remaining. However, dangers arise from removing too much information which may lead to wrong tracking results. This chapter therefore contains an introduction of various approaches for information reduction and an analysis of their impact on the computational complexity of object tracking pipelines.

Considering a typical bottom-up object tracking pipeline, an ideal solution would comprise a reduction of data in every processing step of the pipeline such that every intermediate result contains only the least amount of information that is required for the next step. In Figure 5.1 such an ideal pipeline is shown.

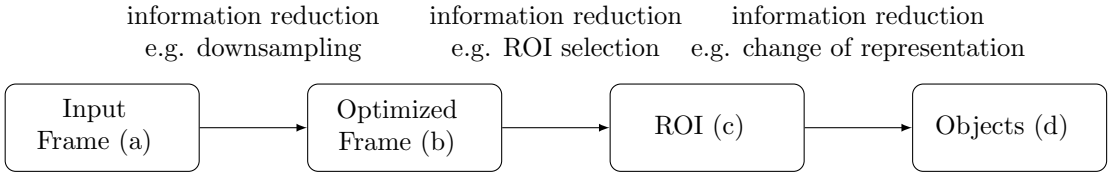

Figure 5.1: Ideal information reduction for an object tracking pipeline which recurs in every processing step, even though by using varying techniques.

There are multiple types of how information reduction can be performed when having an infinite stream of video data for object tracking. This chapter covers a selection of approaches of spatial information reduction such as downsampling, regions of interest and sparse image processing. Thereby, downsampling refers to a modification of the pixel grid V of a frame f such that the pixel size is being increased but the physical image dimensions remain constant. Hence, the number of pixels n_{pixels} to process is being decreased. Downsampling has been described for example by Bovik [11] or Jähne [35]. A specification of a region of interest on the other hand refers to a modification of V such that only a subset $X \subseteq V$ is used for processing. ROI based tracking has been implemented for example by Deguchi [20] and a robust general ROI determination by visual rhythm analysis has been presented by Chi et al. [15]. Sparse processing refers to a more advanced approach of limiting the amount of data to process. Therein, only a few pixels of V are being considered relevant, e.g. by Monte Carlo sampling or particle injection, and further processing is based on this small sample

only. Sparse processing based object tracking has been implemented for example by Lozano and Otuska [49]. Alternative approaches to a reduction of spatial information may include a change of the representation, e.g. from pixels to object representations such as have been stated in Section 3.1, relatively early in the pipeline.

5.1 Downsampling

Downsampling provides an effective way of decimating the number of pixels in V. Thereby, the grid of the original image V_o is being replaced by a new grid V_d and intensities for the positions in V_d have to be estimated by using the known intensities in V_o. In this process, the locations of the pixels may not only be reduced but also their positions may be altered. Also, when reducing the amount of pixels in the grid, their size is being increased in order to retain the original dimensions of the frame. A typical downsampling process is shown in Figure 5.2.

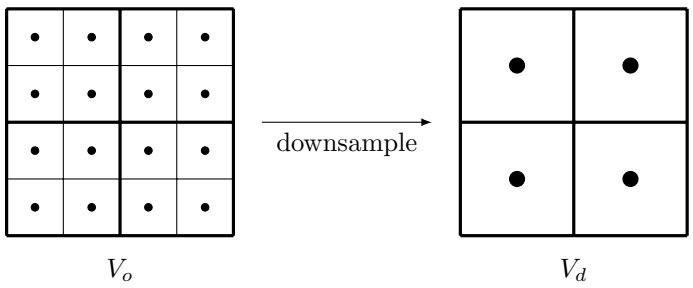

Figure 5.2: Downsampling of a frame, showing the original grid (left) V_o and the desired grid (right) V_d. Pixel center positions are being shown by black dots whereas the surrounding squares represent the size of the pixels. The pixels are not only being decimated but also their positions are being changed and their size is being increased.

Bovik [11] describes downsampling or *decimation* as a process consisting of low-pass filtering and subsampling. Thereby, the original image is being convolved with the impulse response of a low-pass filter before discarding unused pixels of the image. Downsampling involves a loss of information due to aliasing effects which is being reduced by the low-pass filter. The low-pass filter restricts the bandwidth of the signal in order to satisfy the Nyquist-Shannon sampling theorem. In theory, an ideal digital anti-aliasing filter with infinite length could be used for that. However, in real image processing applications short finite impulse response (FIR) filters are being utilized. Bovik also states that the opposite operation, named upsampling or *interpolation,* consists typically of an upsampling operation which inserts zeros between the pixels of the image and a subsequent low-pass filtering [11].

Let V_o be the original grid of a frame f and let V_d be a desired downsampled grid. Let ϕ be a filter on V_o which intersects it with a square neighborhood

$$\phi : V_o \cap S(g, r)$$

The function φ'_f which maps elements $g' \in V_d$ into a field can be denoted as convolution

$$\varphi'_f = \varphi_f \star \phi$$

A subsequent subsampling onto V_d leads to the desired downsampled frame.

Jähne [35] states that interpolation is required when transformed grid points are not coinciding with the original grid points. Thus, this interpretation of interpolation is true for both, upsampling and downsampling. Additionally, interpolation can be used for arbitrary transforms of a grid such as rotation or shearing. Goshtasby [28] describes the same procedure as image resampling, wherein interpolation is used to determine image intensities at new grid positions. Jähne states that the sampling theorem provides the basis for interpolation. A general framework for interpolation therefore has to reconstruct the continuous image from the discrete version and perform a subsequent sampling of the new grid points. Since the reconstruction of the continuous image in practice can be performed only approximately, practical implementations utilize constraints for optimization. Jähne [35] describes in his book linear interpolation, polynomial interpolation, spline-based interpolation and a least-squares approach. Goshtasby [28] lists the following interpolation approaches: nearest-neighbor resampling, bilinear interpolation, cubic convolution, cubic spline interpolation and compactly supported radial functions. However, this set of interpolation functions can only be considered a small subset of available approaches since many specialized interpolation functions such as convolution interpolation [57] have been introduced in publications. In this thesis, an analysis regarding the computational complexity of nearest-neighbor resampling, linear interpolation and b-spline interpolation is being performed, since these interpolators are commonly being used.

5.1.1 Nearest-Neighbor Resampling

One of the simplest versions of interpolation is nearest-neighbor resampling. According to Goshtasby [28] it preserves image intensities, thus leading to similar histograms before and after resampling. Given a original grid V_o and a new downsampled grid V_d, the algorithm computes the value of the nearest-neighbor pixel in $g \in V_o$ for each of the pixels $p \in V_d$. Therefore, the resampled image will not contain any new intensity values that are not existing in the original image. According to Goshtasby, nearest-neighbor resampling does not blur the image but may produce aliasing effects.

Let V_o be the original grid of an image with pixels $g \in V_o$ and V_d be the desired downsampled grid with pixels $p \in V_d$. Moreover, let (g_x, g_y) be coordinates of the pixels g, let (p_x, p_y) be coordinates of the pixels p and let $(p_u, p_v), u \in \mathbb{Z}, v \in \mathbb{Z}$ be coordinates which can represent only integer values. A function for the determination of the nearest-neighbor pixel (g_x, g_y) of a desired pixel (p_x, p_y) is being denoted as

$$SelectNearestNeighbor(p) = \arg\min_{g \in X} d(p, g) \tag{5.1}$$

where $d(p, g)$ represents a distance metric between two pixels, e.g. Euclidean distance, and X is a neighborhood of (p_u, p_v) such that

$$X = \{(g_u, g_v), (g_{u+1}, g_v), (g_u, g_{v+1}), (g_{u+1}, g_{v+1})\}$$

An algorithm for nearest-neighbor resampling of a frame f from V_o to V_d is subsequently being defined as shown in Algorithm 5.1. If there are n_o pixels in the grid V_o and n_d pixels

Algorithm 5.1 Nearest-neighbor resampling

```
1: procedure ResampleNearestNeighbor(f)
2:     for all p ∈ V_d do
3:         p_u = ⌊p_x⌋, p_v = ⌊p_y⌋
4:         g = SelectNearestNeighbor(p)          ▷ according to Equation 5.1
5:         φ_f(p) = φ_f(g)
6:     end for
7: end procedure
```

in the grid V_d, the complexity of this algorithm can be considered linear $\mathcal{O}(n_d)$. Also, the following algorithms of the pipeline will have to process n_d pixels rather than n_o.

A kernel version of the same interpolation, designed for massively parallel execution is shown in Algorithm 5.2. As can be seen, the complexity has been reduced from $\mathcal{O}(n_d)$ to a constant complexity of $\mathcal{O}(1)$.

Algorithm 5.2 Nearest-neighbor resampling (kernel)

```
1: procedure ResampleNearestNeighborKernel(f, i)
2:     p_{i,u} = ⌊p_{i,x}⌋, p_{i,v} = ⌊p_{i,y}⌋
3:     g_i = SelectNearestNeighbor(p_i)          ▷ according to Equation 5.1
4:     φ_f(p_i) = φ_f(g_i)
5: end procedure
```

In comparison, a subsample of the frame f from V_o to V_d would only allow downsampling by an unsigned integer factor, due to the fact that no interpolation of inter-grid values is being performed. The resulting algorithm requires even less code but has still complexity $\mathcal{O}(n_d)$. An example is given in Algorithm 5.3.

Algorithm 5.3 Subsampling

```
1: procedure Subsample(f)
2:     for all p ∈ V_d do
3:         g_x = p_x, g_y = p_y
4:         φ_f(p) = φ_f(g)
5:     end for
6: end procedure
```

5.1.2 Linear Interpolation

Linear interpolation of the intensity value of a point p involves a determination of the weighted sum of intensities of its neighboring pixels. There are multiple versions of linear interpolation available such as linear interpolation itself, bilinear interpolation or trilinear interpolation. For a 2D image, the most commonly used interpolation method is bilinear interpolation, thus covered in this section. According to Goshtasby [28], bilinear interpolation reduces the effect of aliasing at the cost of a blur effect on the image. Thus, repeatedly resampling an image with linear interpolation will destroy details. As image intensities are being adjusted for the new grid positions, the histogram of the image will most likely change as well.

A bilinear interpolation of an image can be denoted by using the same notation as in Section 5.1.1, comprising $V_o, V_d, g \in V_o, p \in V_d, (p_x, p_y), (p_u, p_v), (g_x, g_y)$ and X. Additionally, weights at grid positions of V_o are being defined using the weight function $\omega(g, p)$. According to the definition of Goshtasby, the bilinear interpolation of a point p can be denoted as

$$\varphi_f(p) = \sum_{g \in X} \omega(g, p) \varphi_f(g) \tag{5.2}$$

where the weight function is defined as the distance metric $d(p, g)$, e.g. Euclidean distance, between the point p and a pixel g_j that presents the opposite pixel of g_i in the neighborhood X, denoted as

$$\omega(g_i, p) = d(p, g_j) \ : \ g_j \in X, g_{i,x} \neq g_{j,x}, g_{i,y} \neq g_{j,y}$$

The pseudo-code for resampling a frame f with a bilinear interpolation is shown in Algorithm 5.4. As can be seen, the computational complexity of bilinear interpolation is also

Algorithm 5.4 Resampling using bilinear interpolation

```
1: procedure ResampleBilinear(f)
2:     for all p ∈ V_d do
3:         p_u = ⌊p_x⌋, p_v = ⌊p_y⌋
4:         X = (p_u, p_v), (p_{u+1}, p_v), (p_u, p_{v+1}), (p_{u+1}, p_{v+1})
5:         sum = 0
6:         for all g ∈ X do
7:             sum = sum + (ω(g, p) * φ_f(g))
8:         end for
9:         φ_f(p) = sum
10:    end for
11: end procedure
```

$\mathcal{O}(n_d)$, although bilinear interpolation may require slightly more computation time than nearest-neighbor interpolation. A parallel executable version of bilinear interpolation can be constructed by using the same pattern as has been used for the kernel version. Thereby, the computational complexity is being reduced to a constant value $\mathcal{O}(1)$. However, the kernel

version is not being explicitly shown in this section.

5.1.3 Cubic Spline Interpolation

Unser [69] described splines as piecewise interpolation polynomials that are being connected smoothly at joining points. For cubic spline interpolation, basic splines or B-splines provide an effective way of approximating intensity values. As has been stated by Unser, B-splines of degree n are symmetrical, bell-shaped functions constructed from the $n+1$ fold convolution of a rectangular pulse. Thus, a B-spline of order 0 that is being used for interpolation can be seen as nearest-neighbor interpolation. A B-spline of order 1 represents linear interpolation. A cubic spline has degree 3 and order 4 as has been stated by Goshtasby [28]. For interpolation of discrete images, a discrete B-spline kernel has to be sampled from the continuous B-spline function. Given intensity samples of an image, the coefficients $\beta(p)$ of the B-spline model at a point p in the image can be determined. According to Goshtasby [28], for a 1D image the intensity $\varphi_f(p)$ at a point $0 \leq p \leq 1$ can be estimated by taking into account a neighborhood of pixels $X = \{g_i : i = -1, 0, 1, 2\}$. The estimation is shown for reasons of simplicity only for 1D images and is denoted as follows:

$$\varphi_f(p) = \sum_{i=-1}^{2} \varphi_f(g_i)\beta_i(p) \tag{5.3}$$

where

$$\begin{aligned}
\beta_{-1}(p) &= (-p^3 + 3p^2 - 3p + 1)/6 \\
\beta_0(p) &= (3p^3 - 6p^2 + 4)/6 \\
\beta_1(p) &= (-3p^3 + 3p^2 + 3p + 1)/6 \\
\beta_2(p) &= (p^3)/6
\end{aligned}$$

Thus, the intensity at p is a weighted sum of the intensities in X similar to the approach of linear interpolation. Therefore, in 2D images the estimation is based on the squared neighborhood [69, 28].

Goshtasby [28] additionally states that the estimation of the intensity of p is not the final interpolation. Instead, new intensities have to be found such that when these intensities are being used as control points, the original sample points can be evaluated by the resulting B-spline curve. According to Goshtasby, thereby the entire set of control points has to be computed collectively, leading to an overall complexity of the algorithm of $\mathcal{O}(n^2)$. However, he also mentions that there are faster local implementations of cubic spline interpolation existing and also parallel computable versions are available. For example computing the interpolation in Fourier space will reduce the complexity to $\mathcal{O}(n \log n)$. Cubic spline interpolation has been shown to be computationally more complex than linear interpolation but also produces more accurate results which might be of interest for applications where small details have to be preserved [28].

5.1.4 Information Measurement

As the proposed methods for image downsampling reduce the amount of pixels in the grid, also the information contained in the images may be reduced to a certain extent. In order to determine how much information is lost by a downsampling operation, a measure for the amount of information contained in an image is required. As has been stated by Bankman [4], a common measure for information in image processing is the Shannon-Weiner entropy. Let b represent a single bin of the image histogram B and let p_b be the probability of such a bin. For image processing applications, the entropy measure H can be denoted as

$$H = -\sum_{b \in B} p_b \log_2 p_b \tag{5.4}$$

The probability can be computed as

$$p_b = \frac{n_b}{n_{pixels}}$$

where n_b is the number of pixels in the bin and n_{pixels} is the overall number of pixels in the image.

In order to compare the entropy measures of an original frame and a downsampled frame, the Kullback-Leibler (KL) divergence as has been proposed by Kullback and Leibler [42] can be utilized. It is a non-symmetric measure of the information loss between two probability distributions where one of them is an approximation of the other. Let the probability distributions be represented by a histogram of the original frame B_o and a histogram of a downsampled frame B_d comprising an equal number of bins. The KL divergence can be computed similar to the Shannon-Weiner entropy and is denoted as

$$D_{KL}(B_o \parallel B_d) = \sum_{b \in B_o, q \in B_d} p_b \log_2 \frac{p_b}{p_q} \tag{5.5}$$

The output is a non-negative value which is 0 in the best case if and only if $B_o = B_d$. The higher the output value is, the more information is lost through the process of downsampling. The original intended purpose of the KL divergence is the measurement of the expected number of extra bits that are required to code samples from the original probability distribution using the approximated probability distribution. However, as it is a measure of difference between the two distributions it can also be used as measurement of information loss.

5.2 Regions of Interest

According to Handels [32], a region of interest (ROI) describes a selected region inside a given image that is of particular interest for medical staff. Bovik [11] states that most diagnosis algorithms begin with a specification of a ROI. In the case of kidney stone tracking

in US videos for example, a region of interest in a single frame could comprise the kidney region. ROIs are commonly being used for limiting the amount of data that has to be processed in subsequent stages of a processing pipeline. Another reason for the creation of ROIs is the exclusion of information that is irrelevant for subsequent processing, thus limiting the risk of processing misleading information such as US artifacts. Handels [32] states that the shape of ROIs depends on the application and the structure of the image. Commonly used shapes in 2D image processing include squares, rectangles, circles, elliptical shapes or polygonal structures. Squares, rectangles and circles can be described in a discrete image for example by a specification of a center pixel and a radius such as $S(g, r)$ or $R(g, r_x, r_y)$. As has been stated by Handels, a square can alternatively be described by two points which represent two opposite corners, e.g. $S(g_1, g_2)$. A polygon can be described by a set of points which form a closed region. However, the determination of pixels inside and outside the polygonal region is not as simple as when using rectangular shapes. The usage of a square ROI is depicted in Figure 5.3 [32].

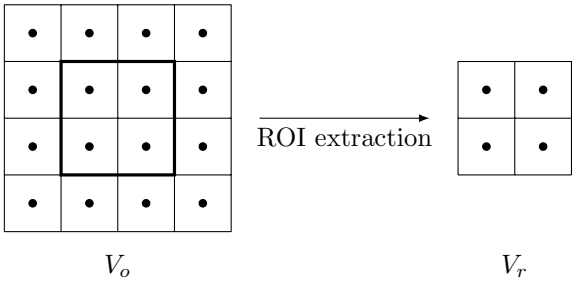

Figure 5.3: Region of interest (ROI) extraction, showing the original grid (left) V_o including the ROI (thick line) and the extracted grid (right) V_r.

Let $S(g, r)$ be a defined ROI in a frame f with $g \in V_o$. A subsequent filter to the ROI definition with a complexity of $\mathcal{O}(n_o)$ would have to check for all g if they are inside the ROI which can be denoted by $g \in V_o \cap S$. Thus, the complexity of the filter would not be decreased. However, if the intersection of the original grid with the ROI is stored in a new grid $V_r = V_o \cap S$, the subsequent filter has a reduced complexity of $\mathcal{O}(n_r)$. Since the extraction algorithm appends additional complexity to the pipeline, depending on its implementation, it has to be decided for a specific case if the reduction process will reduce the amount of processing time or if it will increase time consumption.

Considering a ROI approach to object tracking, an additional computational complexity arises not only from the representation of the ROI but also from the process of ROI selection. In most cases, the selection of a ROI for a frame is dependent on the application. In 2009, Chi et al. [15] have presented a robust ROI determination for multimedia content by analysis of visual rhythms. In that approach, the detected visual rhythms are being matched with user attention models in order to locate attention attracting ROIs. For the application of kidney stone tracking in US videos, the ROI will most likely be determined as the kidney region. However, there are also other possibilities for a ROI including a reduction to a specific renal calix which is a part of the whole kidney region. In order to locate such a ROI, the frame

may have to be analyzed and segmented which introduces additional time complexity to the pipeline. However, the ROI approach can be combined with a downsampling approach in order to reduce the number of pixels which are being considered for ROI detection.

5.3 Sparse Image Processing

Sparse image processing has been researched continuously over the last years. The technique of sparse image processing comprises a sparse representation of images and subsequent processing steps that are capable of using the sparse representation instead of the full image grid. By using a sparse representation, the number of values to process is being decreased, thus $n_{sparse} < n_{pixels}$. However, in contrast to downsampling approaches, comprised essential information is being preserved. As this procedure is dependent on the application, information measures such as the entropy metric may not be reasonable in every case. Starck et al. [66] introduce the terms of *strictly sparse images* and *compressible images*. A strictly sparse image can be modeled as a linear combination of elementary waveforms or *atoms* and can be reconstructed exactly. A compressible image can only be approximated by a linear combination of atoms and therefore can be reconstructed with a certain, ideally minimized, loss of information. Starck et al. list representations such as wavelets, bandlets, curvelets or local discrete cosine transforms which present a backbone of popular applications of sparse image processing such as compression standards, e.g. JPEG, JPEG-2000, MP3, AAC or MPEG. Bandlets have for example been researched by Le Pennec and Mallat [43]. Contourlets for multiresolution image representation have been introduced by Do and Vetterli [21].

According to Starck et al. [66], sparsity terminology consists of four basic terms. *Atoms* represent elementary waveforms such as sinusoids, wavelets or gaussians. From these, more complex waveforms can be constructed by linear superposition. A *dictionary* D is an indexed collection of atoms such as the Fourier dictionary or the wavelet dictionary. Thus, it can be seen as a $N \times K$ matrix where K is the number of included atoms and N is the number of elements inside the atoms. *Analysis* is an operation that decomposes an image f into a coefficient vector $\vec{\alpha}$ wherein each coefficient is linked to an atom in D. *Synthesis* on the other hand is the operation of reconstructing f from α and D. This relationship can be denoted as

$$f \approx \alpha D$$

Thus, an image can be represented sparse by finding its coefficients in α, given a dictionary D such that the number of nonzero elements in the new representation α is minimal and therefore, subsequent algorithms have to process only a fraction of the amount of data that was in the original image. The stated procedure is visualized in Figure 5.4. However, this procedure results in an remaining error ε due to compression [66].

According to Elad [23] a remaining problem is to find α such that ε is minimal and also the number of non-zero elements in α is minimal. One way of measuring ε could be

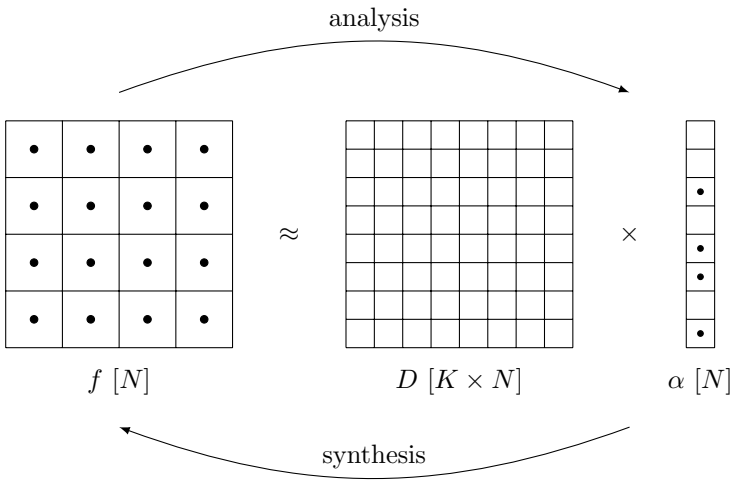

Figure 5.4: Sparse representation analysis and synthesis of $f \approx \alpha D$. The size of the depicted elements is stated in square brackets.

to determine the squared difference of the approximated image αD and the original image f. Another option is to compute the information loss between the two images. This can be done by utilizing for example the KL divergence which has been presented in Equation 5.5 of Section 5.1.4. In order to measure and penalize the number of non-zero elements in α which is also referred to as *sparsity*, Elad utilizes the ℓ_0-pseudo-norm which can be denoted as $\|\alpha\|_0 = \lim_{p \to 0} \|\alpha\|_p^p$, thus counting the number of non-zero elements. In order to solve the problem, either ε can be specified as a maximum error that is allowed in order to find the minimum sparsity, or the maximum sparsity L which is also referred to as *support* can be specified in order to find the minimum ε. The former is shown in Algorithm 5.5. Therein,

Algorithm 5.5 Sparse representation analysis that is limited by a maximum error ε_{max} of the approximation.

```
1: procedure FindSparseRepresentation(f, D, ε_max)
2:     L = 0                                   ▷ set support to minimum
3:     while ε ≥ ε_max do
4:         L = L + 1                           ▷ increase support
5:         α̂ = MinimizeAlpha(D, L)
6:     end while
7:     return α̂
8: end procedure
```

the support is being set to the minimum at the beginning and being increased during the iterations of the algorithm until the desired error has been reached. The latter can be denoted as in Equation 5.6.

$$\hat{\alpha} = \arg\min_{\alpha} \left(\frac{1}{2} \|\alpha D - f\|_2^2 \right) \; : \; \|\alpha\|_0 \leq L \tag{5.6}$$

Regarding to Elad [23], this represents a combinatorial problem of gathering $\binom{K}{L}$ support values which leads to a complexity class of NP-hard. This means that it cannot be solved in

polynomial time and therefore would not be suitable for real-time processing. However, there has been considerable work on this problem and real-time capable practical implementations of sparse processing are available which utilize for example *greedy* approaches. Elad also mentions patch-processing of an image for retrieving a sparse representation. Thereby, patch-processing stands for neighborhood-local processing of the image which might be suitable for parallelization. From the point of view of memory consumption, the given algorithms need additional space for storing D but every frame f can be represented by its compressed version α. In theory, α has the same size as f but in practical implementations it can be stored efficiently by using specific data structures such as kd-trees [23].

Another problem that has been pointed out by Stack et al. [66] is the selection of a dictionary. They state that for example the Fourier dictionary has fast operators and therefore is a suitable candidate for analyzing. However, other dictionaries may lead to sparser representations, depending on the images. Another option is to learn a dictionary instead of using a predefined version. In 2006, Elad and Aharon [24] have shown an approach for sparse representations of grayscale images with learned dictionaries. They have used the K-SVD algorithm to obtain a dictionary that describes a given image efficiently. The algorithm develops a dictionary iteratively by updating it after a new image has been processed. However, the update stage proposes additional time consumption to the overall process. In 2008, Mairal et al. [48] have extended the algorithm for color images to use it for an image restoration application.

Elad [24] listed in his book applications for sparse image processing comprising for example image denoising, image compression, deblurring or restoration tasks. These procedures may be interesting for object tracking in terms of preprocessing of the incoming frames. Especially when tracking objects in US images, denoising may be a critical process in order to increase the signal to noise ratio (SNR). However, some of the implementations of sparse image denoising are only capable of removing additive noise. Therefore, the usefulness of sparse processing for tracking has to be decided for each particular case. Lucas et al. [47] have used sparse processing for image segmentation. They have presented a multi-object level set method that is computationally efficient and is based on a geodesic active contour representation. According to them, the time complexity of the algorithm is linearly growing with the number of pixels in the images but independent of the number of segmented objects. In 1998, Whitaker [74] has presented a level set approach for reconstructing shapes of 3D objects. The proposed level set approach is based on a sparse-field algorithm which tracks active pixels in a set of linked lists. According to Whitaker the time complexity of this algorithm is less than the complexity of the original level set approach.

The presented sparse processing approach is based on an information preserving internal representation of the given images. Particle injection based approaches, such as discussed in Section 3.1.3 on the other hand use an extremely sparse representation of only the most important parts of the frame. These representations reject a major part of the information comprised in a frame. Nevertheless, this often allows for fast and robust tracking and avoids occlusion effects when tracking multiple objects as has been shown in the past in [49] and [75].

Particle injection is frequently used for top-down object tracking approaches but can also be utilized for bottom-up trackers. Chowdhury et al. [16] have proposed a cell tracker based on bipartite graph matching techniques. They assume that the cell objects in each frame have been segmented already and utilize cell centers as vertices of a bipartite graph. The graph is is being utilized for matching cells in subsequent frames which runs in polynomial time. The preceding segmentation could possibly be replaced by Monte Carlo particle injection, resulting in a tracker that is completely based on sparse processing. However, only a small amount of literature seems to exist about a combination of these techniques for object tracking.

Sparse processing has also been used for other object tracking applications in the past. In 2011, Mei and Ling [51] have presented a robust visual tracker that uses a sparse approximation in a particle filter framework similar to the work of Lozano and Otuskta [49] that has been presented in Section 3.1.3. Neighborhood-local sparse patches have been utilized by Wang et al. [72] for online discriminative object tracking. They claim that patch processing associated with a learned dictionary resulted in increased robustness for the approach. Object tracking by using a sparse level set approach has been implemented by Mosaliganti et al. [52]. They have utilized the ITK framework for the implementation which has also been used by Altinger et al. [2] for US nephrolith tracking. The presented tracker has been shown to be capable of real-time cell tracking.

5.4 Information Reduction Revised

This section summarizes the observations that have been made in the preceding sections about information reduction approaches to object tracking. Moreover, the presented tracking pipelines of Section 3.1 are being analyzed in terms of information reduction.

Considering the information that has been presented in the preceding sections, the following statements about information reduction approaches for object tracking can be given:

- Information reduction is a well researched area comprising various fields of applications including object tracking.
- All of the introduced approaches imply an additional complexity that contributes to the overall pipeline complexity. The added complexity also leads to an increased processing time.
- However, the processing time of succeeding steps in the pipeline may be reduced by a considerable factor even when adding the extra complexity of information reduction.
- Some of the demonstrated downsampling operations as well as the sparse processing approach produce an approximated version of the image comprising altered histogram information. Thus, the result of an object tracker might be compromised by removing information.
- Sparse processing in comparison with downsampling or ROIs involves considerably more additional complexity but may reduce the amount of information further than the other

approaches.

Regarding the top-down object tracking approaches of Section 3.1, some of the work reflects information reduction approaches. The CAMSHIFT extension for the mean-shift procedure such as has been presented by Bradski [12] for example uses a ROI for tracking. The internal mean-shift method therefore only works on a reduced region of the frame, thus speeding up the internal part of the pipeline. However, costs are arising from calculating and updating the ROI for each frame. Regarding to Bradski the main intention of using a ROI was to increase the robustness of the approach. Additionally, CAMSHIFT does not solve the problem of finding an initial ROI for tracking. Nevertheless, given an initial ROI the algorithm uses the centroid of the detected object as the center of the next ROI and image moments for deciding on the size of the following ROI.

Particle filter based object tracking by definition utilizes a sparse image representation for tracking. The particle filter such as described by Kroese [41] thereby utilizes a set of particles for filtering. The set of particles can be seen as the coefficient vector α that, when combined with a dictionary such as a PDF, approximately describes the image and allows for filtering. The implementation of Lozano and Otuska [49] indicates that not only the massively parallel computation but also the limited number of particles led to a real-time tracking solution. Parallel computations have been used as countermeasures to the additional computation time that is required for picking the particles.

Although, there has been considerable research on information reduction strategies for object tracking, only a small amount of the publications involves downsizing the input data by downsampling. Hence, an open question is if downsampling in a preprocessing stage of a bottom-up tracking approach affects the tracker such that the accuracy of the result is being decreased massively.

6 Chapter 6

Summary & Outlook

This thesis presents real-time approaches for image processing of US video streams, especially focusing on the problem of object tracking.

The thesis comprises in Chapter 3 an evaluation of existing top-down object tracking pipelines. They have been classified and a sample of approaches has been analyzed in terms of real-time techniques and pipeline structure. The resulting knowledge about existing approaches has been subsequently utilized throughout the thesis. Chapter 4 focuses on the approach of parallel processing for a pipeline speedup. Based on literature, various image processing algorithms have been evaluated in terms of computational complexity and their applicability to the given problem. Multi-device pipelines for massively parallel computations have been analyzed considering fundamental computing science laws and arising technological bottlenecks. As an interim result, the distribution between serial an parallel executed code as well as bottlenecks such as memory bandwidth have been stated to be critical factors to the achievable speedup. Chapter 5 focuses on the approach of information reduction for speedup. Based on literature, the techniques of downsampling, ROI definition and sparse image processing are being evaluated in terms of their suitability for object tracking. As an interim result, it has been stated that although the given approaches reduce the complexity of the pipeline, they produce approximated versions of the frames which may lead to false tracking results.

The practical part of the thesis including an implementation and analysis of the proposed methods is available through the masters thesis *Real-Time Processing of Ultrasonic Video Streams* [37] at the Salzburg University of Applied Sciences.

Bibliography

[1] J. E. Aldrich, "Basic physics of ultrasound imaging," *Critical care medicine*, vol. 35, no. 5 Suppl, pp. S131–7, 2007.

[2] B. Altinger, P. F. Keuschnigg, W. Zweimüller, S. Wegenkittl, and R. Posch-Zimmermann, "Ultrasonic Nephrolith Detection: Documentation of the Research and Development Project," Salzburg, 2012.

[3] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Solid State Circuits Newsletter, IEEE*, vol. 12, no. 3, pp. 19–20, 2007.

[4] I. Bankman, Ed., *Handbook of Medical Image Processing and Analysis*. Academic Press, 2008.

[5] M. Barni and V. Cappellini, "On the computational complexity of multivariate median filters," *Signal Processing*, vol. 71, no. 1, pp. 45–54, 1998.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[7] A. K. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society*, vol. 35, pp. 99–109, 1943.

[8] W. Birkfellner, M. Figl, and J. Hummel, *Applied medical image processing: A basic course*. Boca Raton: CRC Press, Taylor & Francis, 2011.

[9] T. Blaschke, S. Lang, and G. Hay, *Object-based image analysis: Spatial concepts for knowledge-driven remote sensing applications*. Berlin and London: Springer, 2008.

[10] A. Bleiweiss and M. Werman, "Fusing Time-of-Flight Depth and Color for Real-Time Segmentation and Tracking," in *Dynamic 3D imaging*, A. Kolb and R. Koch, Eds. Springer, 2009, vol. 5742, pp. 58–69.

[11] A. C. Bovik, *Handbook of image and video processing*, 2nd ed. Amsterdam and Boston and MA: Elsevier Academic Press, 2005.

[12] G. R. Bradski, "Real time face and object tracking as a component of a perceptual user interface," in *Applications of Computer Vision, 1998. WACV '98. Proceedings., Fourth IEEE Workshop on*, 1998, pp. 214–219.

[13] G. Carneiro, B. Georgescu, S. Good, and D. Comaniciu, "Detection and Measurement of Fetal Anatomies from Ultrasound Images using a Constrained Probabilistic Boosting Tree," *Medical Imaging, IEEE Transactions on*, vol. 27, no. 9, pp. 1342–1355, 2008.

[14] S. Challa, *Fundamentals of object tracking.* Cambridge and UK and New York: Cambridge University Press, 2011.

[15] M. C. Chi, C. H. H. Yeh, and M. J. J. Chen, "Robust Region-of-Interest Determination Based on User Attention Model Through Visual Rhythm Analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 7, pp. 1025–1038, 2009.

[16] A. S. Chowdhury, R. Chatterjee, M. Ghosh, and N. Ray, "Cell Tracking in Video Microscopy Using Bipartite Graph Matching," in *2010 20th International Conference on Pattern Recognition (ICPR)*, 2010, pp. 2456–2459.

[17] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, 2003.

[18] D. Comaniciu and P. Meer, "Mean shift analysis and applications," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1197–1203.

[19] M. Couprie and G. Bertrand, "Discrete Topological Transformations for Image Processing," in *Digital geometry algorithms*, V. E. Brimkov and R. P. Barneva, Eds. Springer, 2012, vol. 2, pp. 73–107.

[20] K. Deguchi, O. Kawanaka, and T. Okatani, "Object tracking by the mean-shift of regional color distribution combined with the particle-filter algorithms," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3, 2004, pp. 506–509.

[21] M. N. Do and M. Vetterli, "The contourlet transform: an efficient directional multiresolution image representation," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 14, no. 12, pp. 2091–2106, 2005.

[22] S. K. Edelman, *Understanding ultrasound physics*, 3rd ed. Woodlands and Tex: ESP, 2005.

[23] M. Elad, *Sparse and redundant representations: From theory to applications in signal and image processing.* New York: Springer, 2010.

[24] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 15, no. 12, pp. 3736–3745, 2006.

[25] D. Exner, E. Bruns, D. Kurz, Grundh&#x00F6, A. fer, and O. Bimber, "Fast and robust CAMShift tracking," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, 2010, pp. 9–16.

[26] K. Fukunaga and L. D. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *Information Theory, IEEE Transactions on*, vol. 21, no. 1, pp. 32–40, 1975.

[27] J. Gai and R. L. Stevenson, "Robust contour tracking based on a coupling between geodesic active contours and conditional random fields," *Journal of Visual Communication and Image Representation*, vol. 22, no. 1, pp. 33–47, 2011.

[28] A. Goshtasby, *Image registration: Principles, tools and methods*, ser. Advances in computer vision and pattern recognition. London and New York: Springer, 2012.

[29] G. Gualdi, A. Prati, and R. Cucchiara, "Multistage Particle Windows for Fast and Accurate Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 8, pp. 1589–1604, 2012.

[30] S. Gunn and M. Nixon, "A robust snake implementation; a dual active contour," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 1, pp. 63–68, 1997.

[31] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[32] H. Handels, *Medizinische Bildverarbeitung: Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie*, 2nd ed., ser. Studienbücher medizinische Informatik. Wiesbaden: Vieweg + Teubner, 2009.

[33] P. R. Hoskins, K. Martin, and A. Thrush, *Diagnostic Ultrasound: Physics and Equipment*. Leiden: Cambridge University Press, 2010.

[34] K.-H. Hsia, S.-F. Lien, and J.-P. Su, "Moving Target Tracking Based on CamShift Approach and Kalman Filter," *Appl. Math*, vol. 7, no. 1S, pp. 193S–200S, 2013.

[35] B. Jähne, *Digital Image Processing*, 7th ed. Heidelberg and Neckar: Springer Berlin, 2010.

[36] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[37] P. F. Keuschnigg, "Real-Time Processing of Ultrasonic Video Streams," Salzburg, 2013.

[38] D. Kirk and W.-m. Hwu, *Programming massively parallel processors: A hands-on approach*. Burlington and MA: Morgan Kaufmann Publishers, 2010.

[39] A. Kolarow, M. Brauckmann, M. Eisenbach, K. Schenk, E. Einhorn, K. Debes, and H. M. Gross, "Vision-based hyper-real-time object tracker for robotic applications," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 2108–2115.

[40] J. Kowalik and T. Puzniakowski, *Using OpenCL*. Amsterdam: IOS Press, 2012.

[41] D. P. Kroese, T. Taimre, and Z. I. Botev, *Handbook of Monte Carlo methods*. Hoboken and N.J: Wiley, 2011.

[42] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[43] E. Le Pennec and S. Mallat, "Sparse geometric image representations with bandelets," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 14, no. 4, pp. 423–438, 2005.

[44] F. Leymarie, *Tracking and describing deformable objects using active contour models*. Ottawa: National Library of Canada, 1991.

[45] X. Li, "Object tracking using an adaptive Kalman filter combined with mean shift," *Optical Engineering*, vol. 49, no. 2, p. 020503, 2010.

[46] Y. Lin, Q. Yu, and G. Medioni, "Efficient detection and tracking of moving objects in geo-coordinates," *Machine Vision and Applications*, 2010.

[47] B. C. Lucas, M. Kazhdan, and R. H. Taylor, "Multi-Object Geodesic Active Contours (MOGAC)," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2012*, ser. SpringerLink : Bücher, N. Ayache and H. Delingette, Eds. Springer Berlin Heidelberg and Imprint: Springer, 2012, vol. 7511, pp. 404–412.

[48] J. Mairal, M. Elad, and G. Sapiro, "Sparse representation for color image restoration," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 17, no. 1, pp. 53–69, 2008.

[49] O. Mateo Lozano and K. Otsuka, "Real-time Visual Tracker by Stream Processing," *Journal of Signal Processing Systems*, vol. 57, no. 2, pp. 285–295, 2009.

[50] M. McCool, J. Reinders, and A. D. Robison, *Structured parallel programming: Patterns for efficient computation*. Amsterdam: Elsevier, 2012.

[51] X. Mei and H. Ling, "Robust Visual Tracking and Vehicle Classification via Sparse Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2259–2272, 2011.

[52] K. Mosaliganti, B. Smith, A. Gelas, A. Gouaillard, and S. Megason, "Cell Tracking using Coupled Active Surfaces for Nuclei and Membranes," *The Insight Journal*, vol. 2009 January - June, 2009.

[53] J. C. Nascimento and J. S. Marques, "Robust shape tracking with multiple models in ultrasound images," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 17, no. 3, pp. 392–406, 2008.

[54] J. A. Noble, N. Navab, and H. Becher, "Ultrasonic image analysis and image-guided interventions," *Interface Focus*, vol. 1, no. 4, pp. 673–685, 2011.

[55] K. Nummiaro, E. Koller-Meier, and L. van Gool, "An adaptive color-based particle filter," *Image and Vision Computing*, vol. 21, no. 1, pp. 99–110, 2003.

[56] D. D. Patil and S. G. Deore, "Medical Image Segmentation: A Review," *IJCSMC*, vol. 2, no. 1, pp. 22–27, 2013.

[57] V. Rasche, R. Proksa, R. Sinkus, P. Börnert, and H. Eggers, "Resampling of data between arbitrary grids using convolution interpolation," *IEEE transactions on medical imaging*, vol. 18, no. 5, pp. 385–392, 1999.

[58] B. Ristic and S. Arulampalam, *Beyond the Kalman filter: Particle filters for tracking applications*. Boston and MA: Artech House, 2004.

[59] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, 2008, p. 73.

[60] J. Sanders and E. Kandrot, *CUDA by example: An introduction to general-purpose GPU programming*. Upper Saddle River and NJ: Addison-Wesley, 2011.

[61] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof, "PROST: Parallel robust online simple tracking," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 723–730.

[62] M. E. Sargin, A. Altinok, B. S. Manjunath, and K. M. Rose, "Variable Length Open Contour Tracking Using a Deformable Trellis," *Image Processing, IEEE Transactions on*, vol. 20, no. 4, pp. 1023–1035, 2011.

[63] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore GPUs," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–10.

[64] Y. Shi and W. C. Karl, "Real-time tracking using level sets," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, 2005, pp. 34–41.

[65] M. C. Shih and K. M. Rose, "Hidden Markov models for tracking neuronal structure contours in electron micrograph stacks," in *Biomedical Imaging (ISBI), 2012 9th IEEE International Symposium on*, 2012, pp. 1377–1380.

[66] J.-L. Starck, F. Murtagh, and J. M. Fadili, *Sparse image and signal processing: Wavelets, curvelets, morphological diversity.* Cambridge and New York: Cambridge University Press, 2010.

[67] J. A. Stratton, N. Anssari, C. I. Rodrigues, I. J. Sung, N. Obeid, L. Chang, G. D. Liu, and W. m. Hwu, "Optimization and architecture effects on GPU computing workload performance," in *Innovative Parallel Computing (InPar), 2012*, 2012, pp. 1–10.

[68] G. Takacs, V. Chandrasekhar, S. S. Tsai, D. M. Chen, R. Grzeszczuk, and B. Girod, "Unified Real-Time Tracking and Recognition with Rotation-Invariant Fast Features," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 934–941.

[69] M. Unser, "Splines: a perfect fit for signal and image processing," *Signal Processing Magazine, IEEE*, vol. 16, no. 6, pp. 22–38, 1999.

[70] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[71] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the Limits of GPU Acceleration," in *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, 2010.

[72] Q. Wang, Feng Chen, Wenli Xu, and M.-H. Yang, "Online discriminative object tracking with local sparse representation," in *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, 2012, pp. 425–432.

[73] J. A. Webb, *Adapt: Global image processing with the split and merge model.* Pittsburgh and Pa: School of Computer Science, Carnegie Mellon University, 1991.

[74] R. T. Whitaker, "A Level-Set Approach to 3D Reconstruction from Range Data," *International Journal of Computer Vision*, vol. 29, no. 3, pp. 203–231, 1998.

[75] C. Yang, R. Duraiswami, and L. S. Davis, "Fast multiple object tracking via a hierarchical particle filter," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 1, 2005, pp. 212–219.

[76] A. Yilmaz, O. Javed, and M. Shah, "Object tracking," *ACM Computing Surveys*, vol. 38, no. 4, pp. 13–es, 2006.