Chapter 5
Stereo Vision Algorithms Suited to Constrained FPGA Cameras

Stefano Mattoccia

Abstract The advent of cheap RGBD active 3D sensors, such as those based on structured light (e.g., the Microsoft Kinect) or those based on time-of-flight technology, has significantly increased the interest in computer vision applications based on depth data that, in most cases, enables higher robustness compared to solutions based on traditional 2D images. Unfortunately, active techniques are quite noisy or even completely useless in outdoor environments (in particular under sunlight). An effective and well-known technique to infer depth suited to indoor and outdoor environments is passive stereo vision. Nevertheless, despite the frequent deployment of this technology in many research projects since the 1960s, stereo vision is often perceived, especially in consumer applications, as an expensive technology due to its high demanding computation requirements. In this paper, we will review a subset of state-of-the-art stereo vision algorithms that have the potential to fit with a basic computing architecture made of a low-cost field-programmable gate arrays (FPGAs), without additional external devices (e.g., FIFOs, DDR memories, etc.) excluding a USB or GigaEthernet communication controller. Compared to more complex designs based on expensive FPGAs coupled with additional external memory devices, clear advantages of the outlined simplified computing architecture are the reduced design and manufacturing costs as well as the reduced power consumption. Another significant advantage consists in better code portability as well as in improved robustness with respect to obsolescence of electronic devices being almost the whole design self-contained into the FPGA logic. On the other hand, mapping stereo vision algorithms into a similar low-power, low-cost architecture poses a very challenging task and only a subset of existing algorithms appropriately modified are suited to this constrained computing platform. Nevertheless, we believe that devices based on such a proposed simplified computing architecture would make RGBD sensors based on stereo vision suitable to a wider class of application scenarios not yet fully addressed by this technology.

S. Mattoccia (✉)
Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
e-mail: stefano.mattoccia@unibo.it

© Springer International Publishing Switzerland 2014

B. Kisačanin and M. Gelautz (eds.), *Advances in Embedded Computer Vision*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-09387-1_5

5.1 Introduction

In recent years, with the widespread diffusion of 3D sensors, there has been increasing interest in consumer and research applications based on dense range data. Some of these sensors provide a depth map and an RGB (or monochrome) image of the sensed scene and, for this reason, they are often referred to as RGBD (RGB plus Depth) sensors. A well-known and representative example of such devices is the Microsoft Kinect, a cheap and accurate RGBD sensor based on structured light technology. Since its presentation in 2010, it has been deployed in many scientific and consumer applications. This technology, developed by Prime Sense, relies on a standard color camera, an infrared projector, and an infrared camera. The projected pattern is sensed by the infrared camera and analyzed according to a patented technology in order to infer depth. The Kinect enables the user to obtain accurate depth maps and images at VGA resolution in indoor environments. Another interesting technology that in recent years gained popularity is *Time of Flight* (ToF). In this case, the sensor emits a modulated light and, by measuring the time required to receive the bounced light, it infers depth. In most cases, this technology also provides a monochrome image of the sensed scene and hence belongs to the class of RGBD sensors. However, compared to the Kinect technology, ToF currently provides depth maps and images at a reduced resolution compared to structured light sensors as well as to stereo vision based sensors. Nevertheless, Microsoft recently presented for its new gaming console an evolution of the original Kinect based on time-of-flight technology enabling increased resolution compared to other time-of-flight sensors currently available. Active technologies have specific strengths and limitations [25]; however, they are ill-suited to environments flooded with sunlight (the Kinect in particular becomes useless in these circumstances). On the other hand, it is worth observing that, in stereo vision technology, depth and image resolutions are only constrained by the computational requirements of the stereo matching algorithm. For these reasons, especially for the limitations concerned with ToF sensors, there have been attempts to improve resolution and effectiveness of active sensors by means of sensor fusion techniques (e.g., [6]). These approaches combine the depth maps provided by active sensors with registered images and depth maps provided by high-resolution stereo vision systems.

Stereo vision is a well-known technology for inferring depth and, excluding projection-based approaches, it is a passive technology based on standard imaging sensors. Stereo vision systems infer dense depth maps by identifying corresponding projections of the same 3D point sensed by two or more cameras in different positions. This challenging task, often referred to as the *correspondence problem*, can be tackled with many algorithms (the Middlebury stereo evaluation website [28] provides a quite updated list and evaluation of stereo vision algorithms) and consequently produces different outcomes in terms of accuracy and computational requirements. This means that, in stereo vision, the algorithm aimed at tackling the correspondence problem plays a major role in the overall technology and, in recent years, there has been a dramatic improvement in this area. Another important factor that has made stereo vision more suitable to a wider range of applications has been the recent availability of low-cost powerful computing platforms such as FPGAs, GPUs, and CPUs with DPS capabilities. Of course, stereo vision technology intrinsically provides RGB or monochrome images, and thus belongs to the class of RGBD sensors. Compared to active technologies such as structured light or ToF, stereo vision may provide unreliable results in regions where the correspondence problem becomes ambiguous (e.g., in poorly textured regions or in presence of repetitive patterns along image scanlines). However, compared to active technologies, it is well suited to both indoor and outdoor environments, as well as to close-range and long-range depth measurements by simply changing the relative position of the digital imaging sensors and/or their optics. Finally, being a passive technology, multiple stereo vision sensors sensing the same area do not interfere with each other enabling multicamera setups. Nevertheless, despite these positive aspects and a widespread deployment in many research applications in the last few decades, stereo vision is often perceived as a bulky and expensive technology not suited to mainstream or consumer applications. In this paper, we will try to address this concern by outlining a simple and cheap computing architecture mainly based on low-cost FPGAs. We will also review a subset of state-of-the-art stereo matching algorithms that have the potential to entirely fit within this constrained architecture without other external device (e.g., FIFOs, DDR memories, etc.), with the exception of a high-speed communication controller. In some cases, the constraints imposed by such simplified computing architecture require modifications to the original algorithms that will be discussed in the remainder of this chapter. The topic addressed in this paper is related to an ongoing research activity aimed at developing a cheap, accurate, self-powered RGBD sensor based on stereo vision technology, deploying as a computing platform only the reconfigurable logic available in standard low-cost FPGAs. This choice has several advantages and also some limitations that will be discussed in the remainder of this chapter.

Figure 5.1 reports preliminary experimental results, computed on a frame of the KITTI dataset [10], concerned with three algorithms discussed in this chapter and implemented on the constrained computing architecture. Additional and updated results can be found here.[1]

5.2 Related Work

Dense stereo vision has been a widely researched topic for decades [31] and, due to its highly demanding computational requirements, many different computing platforms (e.g., CPUs, GPUs, DSPs, FPGAs, ASICs, etc.) have been deployed to obtain depth maps (hopefully) in real time. However, some of these computing architectures, such as those based on standard CPUs or GPUs, are currently ill-suited to

[1] Videos and applications of the 3D camera are available at this links:
http://www.youtube.com/channel/UChkayQwiHJuf3nqMikhxAlw
http://www.vision.deis.unibo.it/smatt.

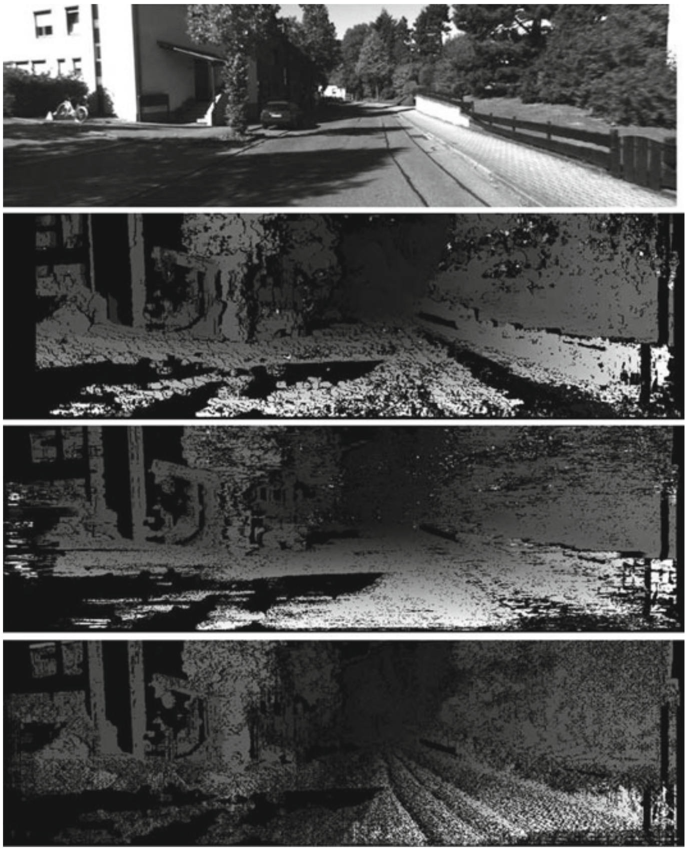

Fig. 5.1 Preliminary experimental results for three algorithms implemented in the target computing architecture. Disparity maps are concerned with frame ♯66 of the KITTI dataset [10], using a simple x-Sobel filter as prefiltering step. From *top* to *bottom* rectified reference image, disparity map computed by the FW implementation, disparity map computed by a modified version of the [5] algorithm using two paths, and disparity maps computed by a modified version of the SGM [13] algorithm using four paths. Additional experimental results are available at this link: http://www.youtube.com/channel/UChkayQwiHJuf3nqMikhxAlw

consumer/embedded applications due to their high power requirements, cost, and size. Computing architectures, such as those based on high-end FPGAs, are often too expensive as well, while solutions based on custom *application specific integrated circuits* (ASICs), despite the limitations regarding their reconfigurability and *time to market* compared to FPGAs, represent a less expensive solution in large volumes. Finally, we point out that interesting low-power, low-cost, reconfigurable architectures for real-time dense stereo vision are represented by embedded CPUs coupled with integrated DSPs, such as the OMAP platform [11], extensively used for stereo vision. A recent and detailed review of stereo vision algorithms for different computing architectures can be found in [33]. In this chapter, we will consider a simple computing architecture based entirely on low-cost FPGAs that, in our opinion, represent an optimal solution to design compact, low-cost, low-power 3D sensors based on stereo vision.

5.2.1 Field-Programmable Gate Arrays

FPGAs can be configured, and in most cases reconfigured many times, by means of hardware description languages (HDLs) such as VHDL or Verilog. The internal structure of an FPGA consists of a large amount of *logic cells*, each containing a small amount of elementary logic blocks (e.g., Flip-Flops, multiplexers, and lookup tables). Distributed into the FPGA, there are also small multiport memories, often referred to as *block RAM*, with fast access time. Moreover, modern low-cost FPGAs often integrate configurable DSPs for efficient arithmetic operations, clock managers, and high-speed transceivers. All these components can be configured by programmers/designers according to their specific requirements by means of HDLs. For instance, considering a Xilinx Spartan 6 Model 45 FPGA, we can find roughly 44,000 logic cells, 116 dual-port block RAMs (18 Kb each), 58 DSPs, 4 clock managers, and 358 configurable I/O pins. It is worth noting that the reconfigurable logic of an FPGA can be configured/programmed with HDLs at a higher level of abstraction using a *behavioral* programming methodology. However, mapping computer vision algorithms on the reconfigurable logic with HDLs is not as simple as mapping the same algorithms on CPUs with traditional high-level programming languages. Nevertheless, recent years have seen the appearance of effective high-level synthesis (HLS) tools that enable the automatic conversion of code written in a standard programming language, such as C/C++ or Matlab, into HDLs. Despite these facts, being the hardware resources of the reconfigurable logic highly constrained, a clear understanding of the overall FPGA architecture and of the available resources is crucial for writing optimized code with HDLs as well as with HLS tools. The key advantage, compared to most other computing architectures, is that FPGAs, thanks to their complete reconfigurability, can be programmed to massively exploit parallelism and tailored to specific application requirements enabling one to obtain the optimal performance/Watt.

5.2.2 Stereo Vision

Stereo vision [31] is a technique aimed at inferring dense or sparse depth maps from two or more views of the same scene observed by two or more cameras. Although increasing the number of cameras has the potential to improve accuracy and reliability, the binocular setup (i.e., deploying two imaging sensors) is frequently deployed in practice. Due to the many applications that can take advantage of dense depth data, this topic has received a constant research interest in the last decades, and significant algorithmic improvements have been proposed [15, 31]. However, most dense stereo vision algorithms are computationally demanding, and parallel computing architectures are in most cases mandatory if one is to obtain dense depth measurements in real time. Not surprisingly, for this reason, FPGAs have attracted the interest of many researchers working in this field [33].

5.3 Overview of the Constrained Computing Architecture

Our target computing architecture is aimed at minimizing cost, power consumption, and at enabling better portability with respect to future evolutions of the FPGA core that, in our case, currently relies on the Xilinx Spartan 6 family. The design strategy adopted here would easily enable the porting of algorithms to FPGA devices provided by different manufacturer to high-end devices manufactured by Xilinx, such as devices belonging to the high-end Virtex class as well as to new computing architecture made of multicore and programmable logic such as those recently proposed by Altera or Xilinx (e.g., the Zynq platform for Xilinx). In particular, these latter *hybrid* architectures, made of ARM cores tightly coupled with powerful reconfigurable logic, would perfectly match our strategy enabling the design of self-contained smart cameras with a very simplified and almost standard hardware design.

A brief overview of our current computing architecture is depicted in Fig. 5.2. It is based on a single FPGA and aims to obtain dense depth maps at more than 30+ fps processing WVGA (752×480) stereo pairs sensed by monochrome or color sensors. These specific imaging sensors, manufactured by Aptina (specifically the MT9V034 model adopted for our evaluation), provide some interesting features well suited to smart cameras for computer vision applications. In fact, these imaging sensors have global shutter capability, are available in monochrome or color (based on the Bayer pattern) format, have a maximum frame rate of 60 fps, provide an optional LVDS data communication between sensors and the device (the FPGA in our case) that manages the video streams and also enable simultaneous acquisition by means of hardware synchronization. Nevertheless, it is worth pointing out that our design is not constrained to this specific imaging sensor and other devices could be used in place with minimal modifications to the overall design. Observing Fig. 5.2, we can notice that the two synchronized color or monochrome imaging sensors are connected, through two *low-voltage differential signaling* (LVDS) channels for clocks and data, to the FPGA. This choice, plus the additional LVDS link between the two imaging sensors, enables us to put the sensors and the computing platform in arbitrary positions, even at distances of meters, in order to deal with different setups according to different application requirements. For instance, in gesture recognition applications, the baseline is typically very small (few centimeters), while for systems aimed at autonomous navigation, the baseline can be significantly larger (50 cm or more). Both cases would be easily handled by the depicted solution. Despite this important fact, other crucial design goals in our project were minimal power require-

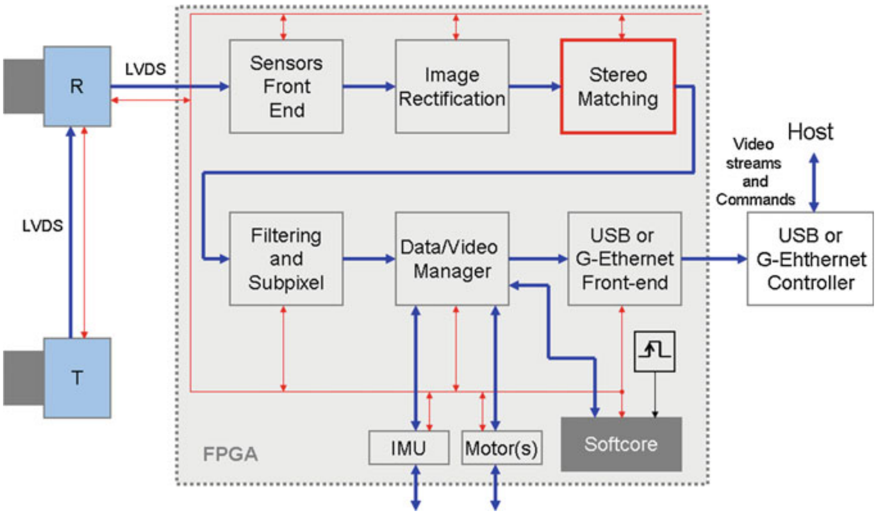

Fig. 5.2 Basic architecture of the target computing platform. The overall design contains the imaging sensors (e.g., at WVGA resolution), a low-cost FPGA (e.g., a Spartan 6 Model 45 or better), and an external high-speed communications controller (e.g., USB 2.0 or 3.0, GigaEthernet). The overall processing pipeline, including the FIFO aimed at handling transfers to/from the high-speed communications controller and a softcore that supervises the whole system, is synthesized into the reconfigurable logic of the FPGA

ment, compactness, and reduced bill of materials. Concerning power requirements, the overall design has a power consumption of about 2.0 W supplied by a standard USB 2.0 data connector. The overall size of the computing platform depicted in Fig. 5.2, excluding the imaging sensors modules, has an area smaller than a credit card. Finally, the bill of materials can be summarized by considering a small amount of inexpensive hardware devices. Namely, it results in the FPGA, the imaging sensors and the external USB or GigaEthernet controller plus some standard electronic components such as clocks and passive devices. According to this overview, it is clear that the considered computing platform is not equipped with any external memory device such as a SRAM or a DDR. This choice simplifies the overall design (enabling power, area, and costs reduction) but, on the other hand, enforces strong constraints to the processing capabilities of such a processing platform that will be thoroughly discussed in the remainder of this chapter.

Observing Fig. 5.2, we can also notice that our design contains a *softcore* synthesized into the FPGA logic; this is a small RISC processor mainly aimed at handling communications with the host computer in order to change camera parameters, such as the frame rate or other features of the imaging sensors, by means of standard serial communication protocols (e.g., I2C). The softcore, by means of software commands issued by the host, also allows us to select, among those available within the processing pipeline, the video streams actually required by the user. For video stream configuration, the softcore is tightly coupled with the Data/Video manager unit as depicted in the figure. In fact, this module manages the video streams processed by the pipeline and other data available inside the FPGA according to the configuration commands issued by the host. Optionally, the hardware design could be equipped with an Inertial Measurement Unit (IMU) made of a gyroscope, an accelerometer and other additional digital devices such as a GPS receiver or a digital compass. The IMU can be useful for robotic applications in order to integrate the measurements provided by a visual odometry module based on SLAM (Simultaneous Localization And Mapping) techniques. Another optional component of the camera, managed by the softcore, is the Motor controller unit. This module enables control of multiple stepper motors according to software commands issued by the host and can be useful, for instance, for handling pan and tilt.

The upper side of Fig. 5.2 summarizes the main steps executed by the vision processing pipeline. Once the raw images provided by the image sensors are sent to the FPGA, they are rectified in order to compensate for lens distortions. Moreover, the raw stereo pair is put in *standard form* (i.e., *epipolar lines* are aligned to image *scanlines*). Both steps require a *warping* for each image of the stereo pair, which can be accomplished by knowing the *intrinsic* and *extrinsic* parameters of the stereo system [31]. Both parameters can be inferred by means of an offline calibration procedure. Once the rectified images are in standard form, potential corresponding points are identified by the stereo matching module as will be discussed in the next sections. Unfortunately, since not all of the correspondences found by the previous module are reliable, a robust outlier detection strategy is crucial. This step typically consists of multiple tests aimed at enforcing constraints to the inferred disparity maps (e.g., left–right consistency check, uniqueness constraint, analysis of the cost curve, etc.), the input images, or the matching costs computed by the previous matching engine according to specific algorithmic strategy. The filtered disparity map is then sent to the Data/Video manager. This unit contains a small FIFO synthesized in the FPGA logic, and it is mainly focused on packaging selected video streams and other relevant data. This data is then sent to the communication front end implemented in the FPGA logic and directly connected to the external communication controller that in the current prototype is an USB 2.0 controller manufactured by Cypress.

The host computer, once it has received the disparity map, will compute depth by triangulation according to the parameters inferred by the calibration procedure. Although in this paper, we will focus our attention on the stereo matching module, the overall goal consists in mapping all the blocks depicted in Fig. 5.2 into a low-cost FPGA. As previously pointed out, a similar design would allow for small cost, size, weight, power requirements, and reconfigurability. Moreover, the upgrade of the whole project to newer FPGAs (typically cheaper and with better performance in terms of speed and power consumption compared to previous generation) is almost straightforward. Finally, we point out that, with the availability of integrated solutions based on reconfigurable logic, plus embedded processors such as the Xilinx Zynq [44], a self-contained FPGA module would make feasible the design of a fully embedded 3D camera with complete onboard processing without any additional external host computer.

5.4 Stereo Vision: Analysis of Memory Footprint and Bandwidth

Stereo vision algorithms are well-known for their demanding computational requirements that sometimes even do not enable their deployment in practical applications with real-time constraints. This limitation in standard computing architecture such as CPUs or GPUs is often concerned with *number crunching* capabilities. However, when it comes to consider highly constrained computing architectures such as that previously outlined, major limitations typically consist in the massive memory footprint and/or bandwidth requirements within the memory and the processing unit. Let us consider these facts by analyzing the simplest stereo matching algorithm that evaluates, within a prefixed disparity range D with disparity $d \in [d_{\min}, d_{\max}]$, the matching costs $C(x, y, d)$ computed, on a point basis, between each point in the reference image at coordinate $R(x, y)$ and each potential corresponding pixel $T(x - d, y)$, $d \in [d_{\min}, d_{\max}]$ in the target image. Many effective cost functions have been proposed in the literature and among these, the absolute difference of pixel intensity (AD) or its truncated version, often referred to as truncated absolute difference (TAD), that saturates the cost to an upper threshold T, Census transform coupled with Hamming distance [47] and its variants such as the mini-Census [4] or the more robust ternary based approach proposed [30] are widely adopted by algorithms implemented into FPGAs. In fact, AD- and Census-based approaches, compared to other cost functions such as squared differences (SD), normalized cross-correlation (NCC) or zero-mean normalized cross-correlation (ZNCC), robust cost functions computed on rectangular patches, or mutual information (MI) [41], are certainly less demanding in terms of reconfigurable logic required for their hardware implementation. In terms of robustness, the nonparametric local transform [47] makes this approach robust to strong photometric variations, although in its original formulation, it is quite noisy in uniformly textured region. Concerning AD, in order to increase its robustness to photometric distortions that frequently occur in practical application scenarios, a transformation that reduces the low-frequency components (e.g., LoG (Laplacian of Gaussian) or Sobel filter) is often applied to the stereo pair before AD computation. For the reasons outlined so far, AD- and Census-based approaches are frequently deployed by stereo vision algorithms implemented into FPGAs. Sometimes, such as in [22], different cost functions (in [22], AD and Census) are combined to increase robustness. Finally, there are approaches [37] that rely on direct edge detection mechanism to improve computational efficiency. An exhaustive review and evaluation of cost functions suited to practical stereo vision systems, not restricted to FPGA implementation, can be found in [14].

Considering the previous example, from the memory point of view, stacking each $C(x, y, d)$ for each point and for each disparity within the disparity range would result in the 3D memory structure depicted in Fig. 5.3 and often referred to DSI (Disparity Space Image). However, in most effective algorithms adopted in practical applications, the matching cost evaluated to determine the best disparity value consists in aggregated pointwise matching costs $C(x, y, d)$, accumulated costs along

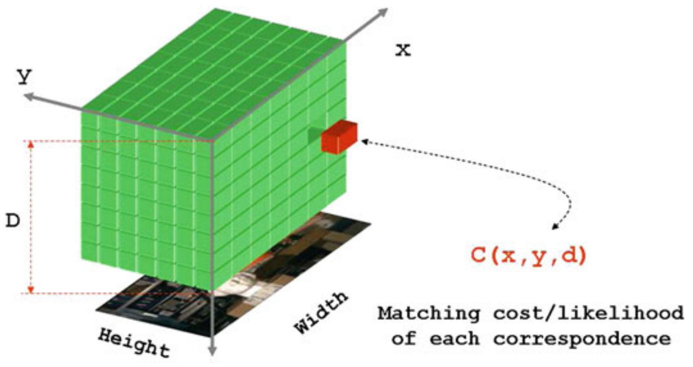

Fig. 5.3 Disparity Space Image, the 3D structure containing the pointwise matching cost $C(x, y, d)$ for each point and for each disparity value within the disparity range D

scanlines in order to enforce smoothing constraints on the disparity maps or by means of other strategies.

Some algorithms, as will be discussed in the remainder, require to store in a memory structure the whole DSI depicted in Fig. 5.3. Unfortunately, even with standard image resolution and disparity range, this is a significant amount of data that typically exceeds the memory available in most current FPGAs and for this reason, an external memory would be mandatory in these cases. For instance, by considering images at 752 $\times$ 480, a disparity range of 64 and 16 bit for each matching cost $C(x, y, d)$, the DSI consists of 44 MB. Although a similar amount of data seems not critical deploying external memory devices such as DDR memory or SRAM memory, there is a more critical constraint concerned with memory bandwidth. In fact, FPGAs, despite their reduced clock frequency, compared to other parallel computing devices such as GPUs, can be effective with respect to such devices by exploiting their potential massive parallel capabilities by means of tailored internal logic reconfigurations. Nevertheless, to this aim and in order to provide a throughput of (at least) one disparity per clock cycle to keep pace with the pixels provided by the imaging sensors, there is a strong memory pressure when intermediate results (for instance, as typically occurs, the D values concerned with the point under examination or D values for intermediate results (sometimes even $k \times D$ values) must be read within a single clock cycle. This case is summarized in Fig. 5.4.

For instance, by considering our previous configuration, D = 64 and size of each matching cost $C(x, y, d)$ 2 bytes, with a pixel clock frequency of 30 MHz (appropriate for imaging sensors similar to those deployed in our camera), the memory bandwidth required turns out to be higher than 3.5 GB/s. In most cases, for each clock, this amount of data must be read, processed/updated, and then written back to memory, thus doubling the overall required memory bandwidth highlighted. Of course, with higher resolution imaging sensors, typically clocked at higher frequency, moving data back and forth between FPGA and memory further emphasizes the memory bandwidth bottleneck.

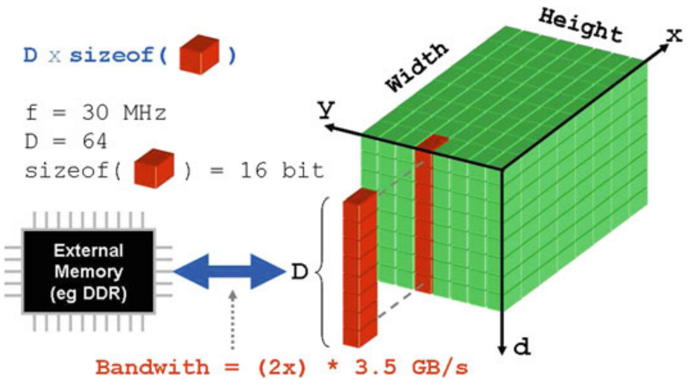

Fig. 5.4 Moving data back and forth between FPGA and external memory can easily lead to exceed the available bandwidth

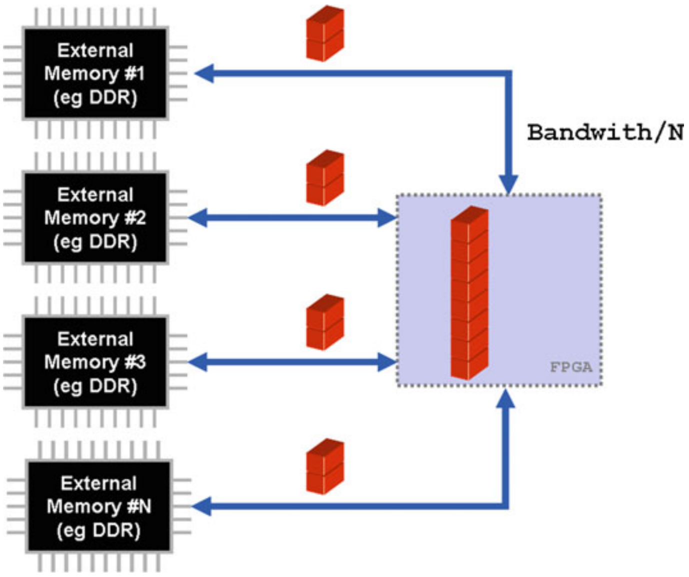
External Memory #1 (eg DDR)
External Memory #2 (eg DDR)
External Memory #3 (eg DDR)
External Memory #N (eg DDR)
Bandwith/N
FPGA

Fig. 5.5 The overall memory bandwidth can be increased by adding to the design multiple memory devices

According to the previous analysis, since the memory bandwidth required can exceed that available in current memory devices, a straightforward solution to overcome this problem would consist in using multiple memory devices as outlined in Fig. 5.5. However, although the strategy depicted in the figure could solve memory bandwidth issues, this strategy would have on the other hand some disadvantages. In particular, using multiple memory devices to increase the overall memory bandwidth would lead to increased costs, power consumption, overall complexity, and area. Moreover, such a solution would also increase the overhead, in terms of reconfigurable logic and memory controllers, required by the FPGA for handling multiple external memory devices.

For the reasons outlined so far, in our design, we decided to avoid external memory devices at all. Although this choice enables to overcome some of the problems previously outlined, it also means that we can rely only on the fast, yet small, memory available inside the FPGA. As should be clear, this choice imposes very strong constraints on the algorithms that can be implemented on such a computing architecture. Nevertheless, as we will show in the next sections, following specific algorithmic methodologies, very effective results can be obtained adopting this simplified design strategy.

5.5 Stereo Vision Algorithms Suited to the Constrained Computing Architecture

A computing architecture similar to that outlined in the previous section poses significant constraints to the computational structure of the algorithms that can be implemented. In fact, considering a representative case study of the Xilinx Spartan 6 FPGA family [44], we can see that the overall block memory available is about 261 KB for the Model 45 (and about 600 KB for the most powerful device of this family, the Model 150). In between models 45 and 150, there are two devices, 75 and 100, with an amount of logic cells close to, respectively, 75,000 and 100,000 and with an amount of block memory, respectively, of about 400 KB and 600 KB. This means that, ignoring other requirements, we would not even be able to store a stereo pair at WVGA resolution (about 720 KB) in the most complex 150 device. This observation, plus the limited overall reconfigurable logic available (about 43,000 and 147,000 logic cells for the Model 45 and 150, respectively), dictates that *stream processing* [2] is mandatory for our purposes. This technique consists of processing pixels as soon as they are available from the imaging sensors, with minimal buffering requirements. Of course, for the same reason, the resulting output cannot be entirely stored into the FPGA and must be sent to the communications controller as soon as it is made available by the processing pipeline. We also point out that other relevant constraints are concerned with the overall reconfigurable logic available for processing (e.g., about 55,000 flip-flops for the Model 45 and 185,000 flip-flops for the Model 150) and the maximum distributed RAM available (e.g., about 50 KB for Model 45 and about 17 KB for Model 150). More details concerned with these devices are available in [44].

In the next sections, we will consider some relevant stereo vision algorithms potentially suited to this constrained target architecture. For this purpose, we will adopt the classification proposed in [31], where algorithms are classified into two major categories, local approaches and global approaches, making a further distinction when dealing with approaches not completely described by these two broad categories.

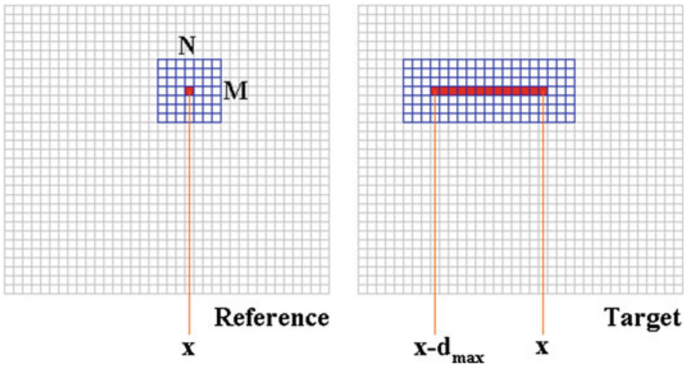

Fig. 5.6 Support windows, of size $M \times N$, for cost aggregation in local algorithms at disparity d. In the reference image, the support window is centered on point (x), while in the target image, the support windows are centered on points $[x, x - d_{\max}]$

5.5.1 Local Algorithms

Local algorithms process each point independently ignoring relationship between neighboring disparity values. For this reason, they do not enforce an explicit smoothness constraint on the target disparity map, and typically for each disparity candidate within the disparity range D, compute the matching cost by aggregating neighboring pixels (often on rectangular patches referred to as *support* windows, as depicted in Fig. 5.6). Cost aggregation is often explicitly obtained by summing up, according to different strategies, each pointwise matching cost within the support window, as depicted in Fig. 5.6. However, it is worth noting that some recent approaches implicitly aggregate costs in constant time, independently of the size of the support [7, 27].

More generally, cost aggregation performed by most local algorithms can be figured out, as depicted in Fig. 5.7, as a filtering [15] of the DSI data structure according to different strategies. Examples of filtering operations applied to the DSI are averaging/sum, bilateral filtering [46], approximated bilateral filtering [19], guided filter [12], etc. Since local algorithms completely ignore relationships between neighboring points and different disparity values within the disparity range, from the computational point of view, this means that D filtering operations can be applied in parallel to the DSI in order to aggregate matching cost for each disparity candidate. Adopting for the processing pipeline the same clock of imaging sensors, the computation of the D filtering operations for each point should hopefully finish within a single (pixel) clock cycle. This fact potentially enables a high degree of parallelism (multiplied by a factor D compared to the sequential case) but at the same time it imposes that all the data required in the DSI (highlighted in the DSI depicted in Fig. 5.7), or a subset of this data centered in the point under examination, must be accessed in parallel. Therefore, at least the portion of data highlighted in Fig. 5.7 should be carefully managed, by means of appropriate data structures and buffering

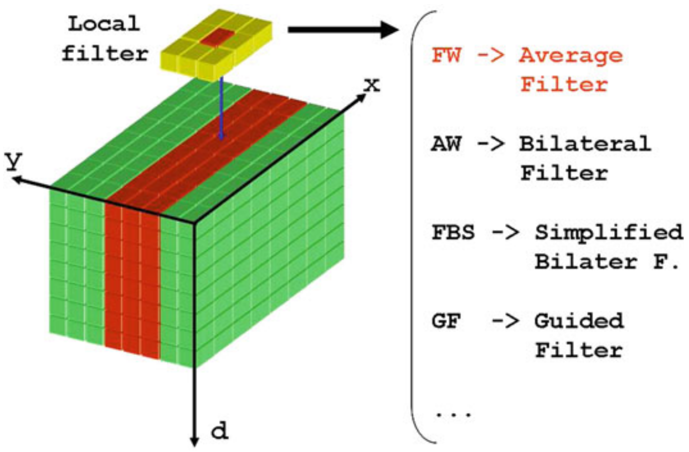

Fig. 5.7 In most local algorithms, cost aggregation is carried out by filtering the DSI according to different strategies

techniques implemented in the internal logic of the FPGA, in order to enable parallel access required to sustain parallelism. Of course, in the outlined constrained computing architecture, the overall portion of the DSI highlighted in the figure must be stored in the internal memory of the FPGA, typically in the block RAM. Nevertheless, with WVGA imaging sensors (even with imaging sensors at higher resolution) and typical disparity range deployed in practical applications, this amount of data becomes compatible with the internal memory made available by FPGAs similar to those previously examined.

In local algorithms, the best disparity for each point is often identified according to a simple *winner-takes-all* (WTA) strategy by finding the candidate with the best aggregated cost. For the reasons previously outlined, local algorithms are inherently parallel, and hence are ideal candidates for FPGA implementation. Detailed reviews and evaluations of local stereo algorithms can be found in [15, 31, 33, 36, 42].

5.5.1.1 Fixed Window Algorithm

In spite of their simplicity and intrinsic parallel nature of local algorithms, even the mapping of the simplest approach to the constrained target architecture outlined in previous sections should be carefully planned. The simplest local algorithm is often referred to as fixed window (FW) or *block matching* and simply sums up/averages all the matching costs within the support window. Although this algorithm has some well-known limitations, such as inaccurate depth reconstruction near depth discontinuities and, as most local algorithms, problems in uniformly textured regions of the sensed scene, it is often deployed in several practical applications, thanks to its overall robustness in determining the rough 3D structure of the scene and to its simple algorithmic structure.

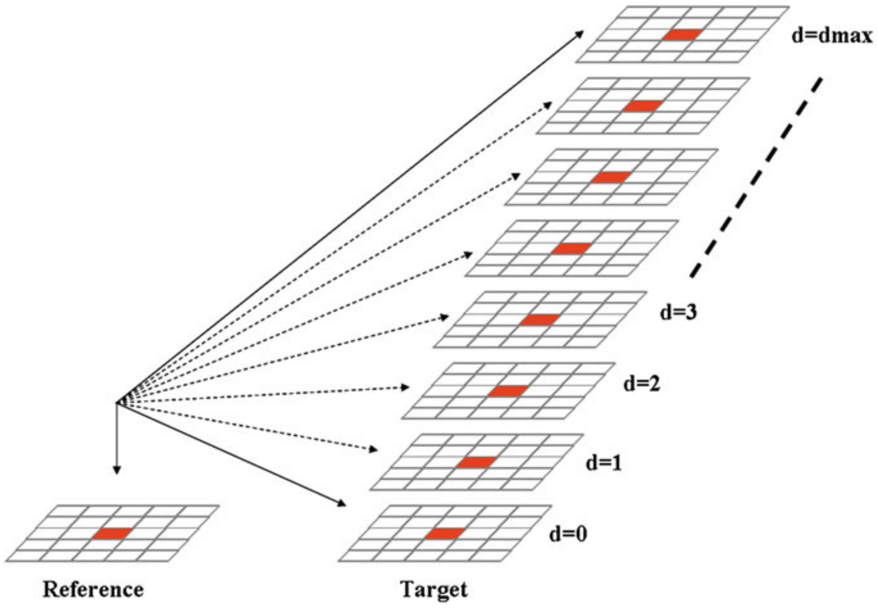

Fig. 5.8 Multiple filters applied to the reference and target images for cost aggregation. The number of filter is equal to the disparity range ($d_{\max} + 1$ in the figure)

According to Fig. 5.7, in this case filtering consists in summing/averaging within the support, for each disparity value, the matching costs in the DSI. From a different point of view, this operation consists in applying multiple instances of the same filter (sum/average filter for FW) between reference and target image as illustrated in Fig. 5.8.

With a support of size $M \times N$ and a disparity range of $[0, d_{\max}]$, the number of arithmetic operations for the brute force approach is proportional to $M \times N \times (d_{\max} + 1)$. Considering that plausible values for these parameters could be $M = 15$, $N = 15$, and $d_{\max} = 63$, the number of arithmetic operations required might exceed the hardware resources available in the target FPGA. Nevertheless, this number of operations can be significantly reduced by adopting well-known incremental calculation schemes such as *box-filtering* [21] or *integral images* [40]. The former in particular, as outlined in Fig. 5.9, is particularly suited for FPGA implementation of the FW algorithm.

The figure shows that the overall cost aggregation for the supports depicted in the upper side of the figure can be obtained more efficiently by deploying the 1D optimization depicted in the middle of the same figure. In fact, the aggregate costs required by the operations in the upper side of Fig. 5.9 can be reduced by observing that the overall cost for the central point can be obtained by updating the overall cost computed in the previous position along the scanline, adding the aggregated costs of the rightmost columns, and subtracting the aggregated costs of the leftmost columns.

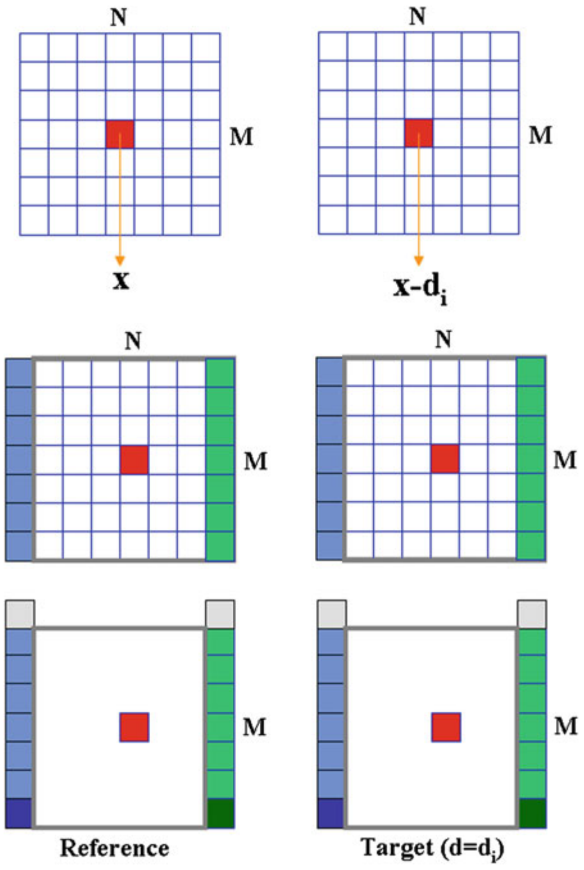

Fig. 5.9 Incremental techniques, referred to as box-filtering, aimed at reducing to a constant value the number of basic operations required for cost aggregation in fixed window. *Top*, full cost computation. *Middle*, incremental optimization along horizontal direction (number of elementary operations reduced by a factor N). *Bottom*, vertical optimization required to compute the sum of the vertical stripes shown in the middle of the figure. In this latter case, the full cost for each disparity value, is computed with a fixed number of operations involving the four points highlighted at the *bottom* of the figure in reference and target images

In deploying this 1D optimization strategy, the number of operations is reduced by a factor N, manageable with the logic included in most FPGAs. Nevertheless, a further reduction of operations can be obtained by deploying a 2D incremental scheme that stores intermediate results for each column as depicted in the bottom of the figure. In this case, the number of operations per window is constant and independent of the size of the support, though compared to the brute force approach depicted at the top of the figure, at the expense of a higher memory footprint for buffering intermediate results required to sustain the additional 2D incremental calculation.

Several implementations of the FW algorithm, and its variants, suited for FPGA were proposed in the literature with different degrees of algorithmic complexity and hence with different resources used. Some recent representative approaches in this field are [17, 39, 50]. In Sect. 5.6, we report experimental results concerned with our implementation of the FW algorithm on the constrained hardware architecture previously outlined.

5.5.1.2 Local Algorithms Based on Adapting Weights and Constrained Supports

Although the FW approach is widely used in many practical applications, it is clearly outperformed by more recent local approaches based on cost aggregation techniques that aggregate costs according to weights assigned by examining the image content [15, 19, 23, 36, 46]. In these approaches, differently by the simple average score computed by FW, the overall score is given by a weighted sum/average of the costs computed within each support window [18]. The key idea behind this strategy consists of weighting each cost according to its *relevance* with respect to the point under examination (i.e., the central points of the supports).

Many methodologies to assign weights have been proposed in the literature and an effective rationale is that inspired by bilateral filtering [26, 34] applied to the stereo matching by the AW (Adaptive Weights) algorithm [46]. That is, points with *similar* intensity with respect to the central point should be more influential in the weighted sum. Moreover, points *closer* to the central point should also be more relevant according to [46]. This strategy is similar to the weight computation strategy used by bilateral filtering and, in stereo, weights are often computed within the support window of reference and target images (this strategy is often referred to as *joint* or *symmetric*). In the strategy based on segmentation [35], the original bilateral filtering weight computation was relaxed by removing the proximity constraint.

A first optimization [15, 23] consists of asymmetrically computing weights, examining only the image points belonging to the reference image. Although this strategy significantly reduces weight computation by a factor of $d_{\max}$, the number of operations required for cost aggregation is always proportional to $M \times N \times (d_{\max} + 1)$ and may exceed the resources available in the target FPGA. However, simplified yet effective strategies based on the computation of weights and/or costs and/or overall weighted costs only in sampled points may help to further reduce the number of elementary operations per point, maintaining high accuracy. These approaches also exploit massively incremental calculation schemes for cost computation, similar to those outlined for FW. An approach that efficiently computes weights, on a sparse regular grid, and aggregated costs on a block basis, by means of [21], is the Fast Bilateral Stereo (FBS) algorithm [19]. This approach represents a link between the traditional fixed window approach and AW. Figure 5.10 shows for the Tsukuba stereo pair, the results obtained by FW, AW, and FBS. Compared to AW, FBS obtains equivalent results with a fraction (about 10 %) of operations making it suitable for hardware

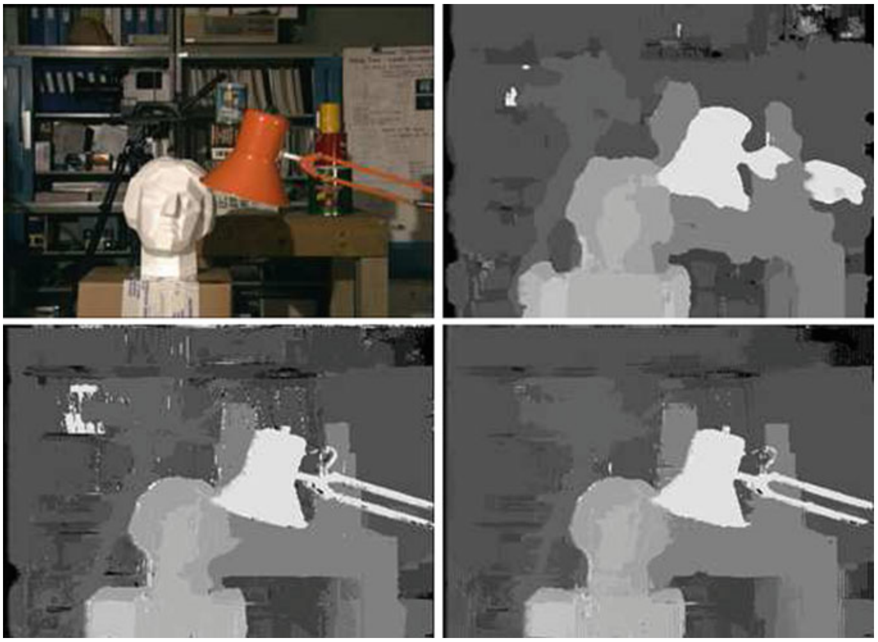

Fig. 5.10 *Top left*, original reference image of the Tsukuba stereo pair [28, 29]. *Top right*, disparity map obtained by the FW approach. *Bottom left*, disparity map obtained by the AW algorithm [46]. *Bottom right*, disparity map obtained by the Fast Bilateral Stereo algorithm [19]

implementation. Observing the figure, we can also notice that algorithms based on the adapting weights strategy are much more accurate near depth discontinuities.

In [23], further optimizations compared to FBS have been devised, including a *preselection* of potential candidate disparities and asymmetric weight computations making also this algorithm a candidate for hardware implementation.

Zhang et al. [48] described a different and effective strategy for cost aggregation based on two orthogonal cost aggregation phases. As for many previous methods, this approach heavily relies on incremental calculation schemes for fast cost aggregation.

A different two-phase strategy to reduce the computational burden of the original AW approach consists in the two-pass aggregation described in [43]. In this method, originally deployed within a Dynamic Programming framework, the nonseparable weighted cost computation of AW is approximated with a vertical cost aggregation, followed by an horizontal aggregation of the costs computed during the first phase (i.e., vertical cost aggregation). Compared to AW, this simplified strategy enables to obtain similar results reducing significantly the number of operations from O(n^2) to O(n), with n the cardinality of the support window.

Finally, it is worth noting that some recent local algorithms [7, 27] filter the DSI according to the guided filtering technique [12], thus enabling weighted cost aggregation in constant time. Such approaches massively exploit incremental calculation

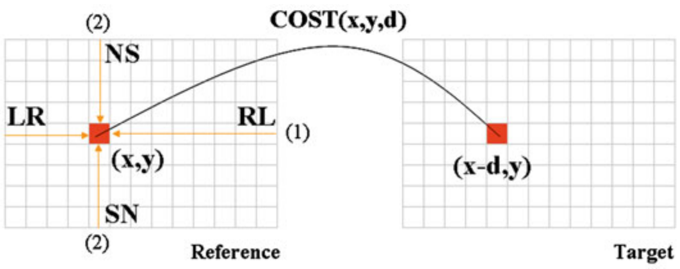

Fig. 5.11 The four paths used to compute permeability terms and aggregated cost for algorithm [5]

techniques [21] potentially suited to the constrained FPGA architecture, thanks to the reduced (and constant) number of operations required with respect to explicit cost aggregation approaches inspired by bilateral filtering. Despite these positive facts, the results provided by these constant time algorithms are comparable to those based on explicit cost aggregation, and hence these algorithms are potentially suited for implementation in the outlined target platform.

Concerning FPGA implementations based on adaptive weights strategy, [8] reported a hardware friendly implementation of the original AW approach [46]. In [24], a further simplification of the AW approach based on simpler binary weights was devised. A method based on the mini-census transform for cost computation and the two-pass approach [43] for cost aggregation is [4] while [49] used, with the same cost function, the two pass orthogonal cost aggregation strategy proposed in [48]. An FPGA implementation of a cost aggregation strategy based on segmentation is reported in [38]. Finally, an interesting method based on adaptive weight cost aggregation, identification of reliable points and disparity refinement, by means of an effective method aimed at enforcing local consistency [20] of the disparity field, was proposed in [16].

5.5.1.3 Algorithms Based on Unconstrained Supports

According to the taxonomy provided in [29], the algorithms reviewed in the previous sections clearly belong to the class of local algorithms. However, there are some local algorithms that significantly diverge from traditional approaches, in particular for what concerns the support regions used for cost aggregation.

An interesting local approach, referred to as Permeability, was proposed in [5]. This technique performs multiple 1D cost aggregations constrained by an *information permeability* term (see [5] for a detailed explanation) computed along horizontal or vertical scanlines, as shown in Fig. 5.11, without enforcing any explicit smoothness term. The permeability term, computed along the horizontal scanline from left to right is defined as follows:

$$W^{\mathrm{LR}}(x, y) = e^{-\frac{|I(x,y)-I(x-1,y)|}{\sigma}} \tag{5.1}$$

with $I(x, y)$ and $I(x-1, y)$ the pixel intensity at coordinate (x, y) and $(x-1, y)$, respectively, in the reference image and σ an appropriate empirically determined constant value. The aggregated cost computed along the same horizontal scanline and direction, from left to right in the considered example, is then computed according to:

$$C^{\mathrm{LR}}(x, y, d) = C(x, y, d) + W^{\mathrm{LR}}(x-1, y) \cdot C^{\mathrm{LR}}(x-1, y, d) \tag{5.2}$$

This strategy, applied to both horizontal directions depicted in Fig. 5.11 and along both vertical directions on horizontally aggregated costs, efficiently enables to adaptively perform cost aggregation on unconstrained 2D support windows. More precisely, cost aggregation is initially independently performed along horizontal scanlines (from left to right (LR) and right to left (RL)). Then, a similar approach is applied along vertical directions (from top to bottom (NS) and bottom to top (SN)) to the summed aggregated matching cost computed along horizontal paths. In practice, in this method, the support window implicitly consists of the entire image. Although this strategy requires us to store the entire image and matching costs, a simplification of the original approach restricted to a subset of scanlines (e.g., from left to right, from right to left, and from top to bottom) is certainly feasible for the constrained target computing architecture outlined. We report in Sect. 1.6, results concerned with our hardware friendly implementation of the Permeability algorithm based on two single directions. Another implementation suited to FPGAs was proposed in [1].

Finally, a different and effective algorithm that, similarly to the previous method, does not explicitly define a fixed support window was proposed in [45]. In this approach, the matching costs are aggregated using as weights the minimum intensity distance between any two points in the reference image. These weights are stored in a tree structure and, to this aim, a MST (Minimum Spanning Tree) containing a number of nodes equal to the number of image points is created. This enables to very efficiently and in constant time obtain for each point the aggregated weighted cost computed on the whole image. Nevertheless, although this method is very fast and effective on traditional CPU or GPU architectures, in its original formulation, it, due to the memory footprint required to store the MST, seems inappropriate to a target computing architecture without being provided with external memory devices, such as DDR memory.

5.5.2 Global and Semiglobal Approaches

Although local algorithms described so far yield excellent results, they are often outperformed by approaches that explicitly enforce a smoothness term on the resulting disparity map. These methods solve the correspondence problem in terms of a pixel-labeling assignment of disparities, determining the disparity field D that minimizes the energy term (5.3):

$$E(D) = E_{\text{data}}(D) + E_{\text{smooth}}(D) \tag{5.3}$$

The *data term* E_{data} in (5.3) encodes how well the disparity assignment fits with the stereo pair, and often it is the sum of per-pixel data costs $C(D(p))$ between one point in the reference image R and the supposed homologous point in the target image T:

$$E_{\text{data}}(D(p)) = \sum_{p \in R} C(D(p)) \tag{5.4}$$

In (5.3), the *smoothness term* $E_{\text{smooth}}(D)$ enables to enforce a piecewise disparity field D by modeling the interaction between each point p and its neighboring points $q \in \mathcal{N}(p)$. In fully global approaches, $\mathcal{N}(p)$ includes points in vertical and horizontal directions (typically, the four nearest neighbors of p on the pixel grid) while in 1D approaches, based on Scanline Optimization (SO) or Dynamic Programming, the smoothness term is enforced only in one direction (typically $\mathcal{N}(p)$ includes only one point along a scanline). The former disparity optimization methods are typically referred to as 2D. In general, 2D methods perform better as they enable the enforcement of *inter* and *intra* scanline smoothness assumptions.

Unfortunately, when deploying a 2D approach, minimization of (5.3) turns out to be an $\mathcal{NP}$-hard problem. Therefore, global approaches typically rely, under particular hypotheses [32] on (5.3), on efficient energy minimization strategies typically based on Graph Cuts (GC) or Belief Propagation (BP). However, the iterative nature of these energy minimization strategies and their high memory footprint typically render these approaches inappropriate for devices with limited resources such as for our target architecture.

Nevertheless, a subclass of these algorithms that enforces disparity constraints on 1D domains by means of dynamic programming such as [43] or multiple scanline optimization [13] represents, for the outlined target computing architecture, a viable and effective alternative to local approaches. In particular, the semiglobal matching algorithm [13] computes multiple energy terms by means of the SO technique [29], independently enforcing 1D smoothness constraints along different paths (typically 8 or 16 from all directions as depicted in Fig. 5.12 for 8 paths).

The 1D energy terms independently minimized by means of the scanline optimization approach are then summed up and the best disparity is determined by means of the same WTA strategy adopted by local algorithms. Figure 5.13 shows, at top and middle, the disparity maps concerned with two 1D minimizations, independently computed along paths 0 and 5, and, at the bottom of the figure, the result of the multiple 1D optimization computed along eight paths. Observing the figure, we can notice that, although single scanline optimizations are not very accurate, their combination by means of the method proposed in [13] turns out to be very effective as can be seen, using eight paths, at the bottom of Fig. 5.12.

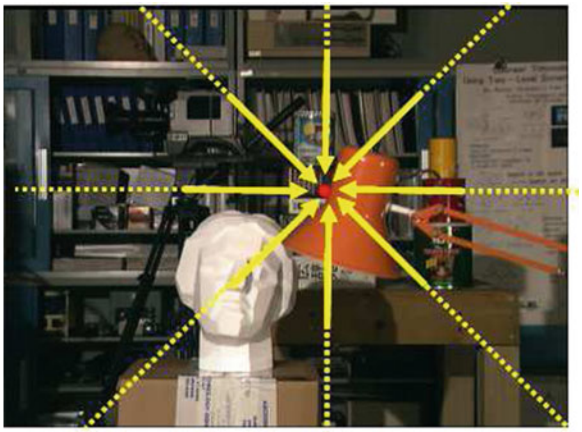

Fig. 5.12 The semiglobal algorithm [13], on each path, performs independent 1D disparity optimizations. The figure considers only eight paths

The strategy adopted by SGM enables fast implementations on CPUs and GPUs and it is very effective. For these reasons, SGM is frequently deployed in many practical applications. However, in its original formulation, due to its high memory footprint (it requires the entire DSI), it is not well suited to a computing architecture without a large amount of fast external memory. Moreover, in its original formulation, the SGM algorithm scans reference and target images two times (from top to bottom and then from bottom to top) making unfeasible the stream processing methodology required by our target architecture.

Nevertheless, by deploying a subset of the original paths (e.g., only four paths), the SGM algorithm becomes suitable with acceptable performance degradation for our target platform. We report in Sect. 5.6 experimental results concerned with our implementation of the SGM algorithms adopting this strategy.

Concerning FPGA implementations of the SGM algorithm, Gehrig et al. [9] implemented the original algorithm proposed in [13] by means of a two-pass approach on downscaled half resolution input images (originally at 680) using 8 paths. Differently, Banz et al. [3] proposed a simplified version of the SGM algorithm for hardware implementation aimed at reducing memory constraints using 4 paths (0, 4, 2 and 7 in Fig. 5.13).

5.6 Experimental Results

In this section, we report preliminary experimental results concerned with the implementation of three stereo vision algorithms—belonging to the three classes defined in the previous section—in the outlined computing architecture made of a single FPGA without additional external devices, with the exception of a high-speed

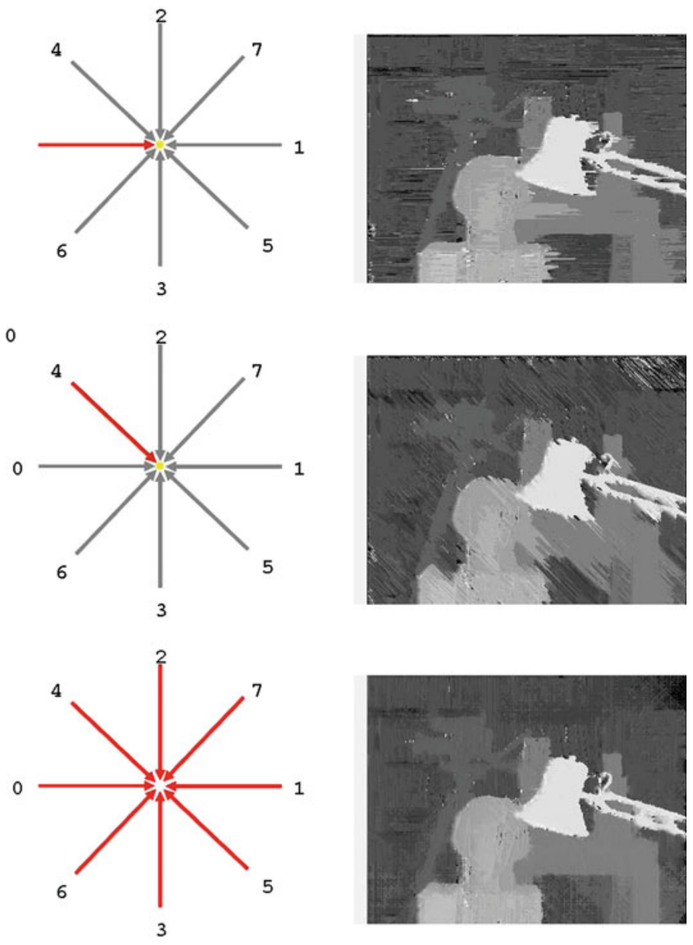

Fig. 5.13 *Top*, 1D scanline optimization along scanline 1—*Middle*, 1D scanline optimization along scanline 4—*Bottom*, result of the SGM algorithm performing multiple 1D scanline optimizations along the eight paths shown at the *left*

communications controller as depicted in Fig. 5.2. Each of these algorithms, as well as all of the other blocks depicted in the figure, was mapped on a Spartan 6 FPGA 45 and delivers depth maps at 30+ fps when processing stereo pairs at WVGA resolution as those deployed by our camera.

Specifically, the algorithms currently implemented in our target architecture are: FW using the optimization strategies previously outlined, a modified version of the Permeability algorithm [5] using two paths and a modified version of SGM [13] using four paths. Each implementation of these algorithms also includes image rectification, a prefiltering step based on the x-Sobel filter, and a postprocessing step aimed at filtering outliers by detecting uniformly textured regions as well as by detecting unreliable disparity candidates analyzing their local distribution. The design also mapped into the FPGA also includes the internal FIFO and all the other modules depicted in Fig. 5.2.

For evaluation purposes, we provide in Fig. 5.1 experimental results concerned with the three implemented algorithms processing frame ♯66 of the KITTI dataset [10]. Observing the figure, where in the disparity maps brighter greyscale levels encode closer points and darker levels farther points, we can see that all three algorithms enable us to obtain dense and fairly accurate disparity maps of this challenging stereo pair. Observing the trees in the right side of the reference frame, we can notice that the SGM algorithms seems less noisy compared to the two local algorithms. In the same figure, we can also notice the most unreliable (e.g., occlusions) and uniform regions (e.g., shadows) are correctly detected by the postfiltering modules.

At this link http://www.youtube.com/watch?v=KXFWIvrcAYo is available a video[2] concerned with an outdoor sequence processed by our modified version of the SGM algorithm implemented on the outlined constrained target architecture. In this case, the stereo camera, based on a Spartan 6 Model 75, was configured with a very short baseline of about 4 cm. Observing this video, we can notice that the camera provides, at high frame rate, very accurate and dense depth maps processing stereo pairs at 640×480 resolution. In the video, we can also notice that the outlier detection module implemented into the FPGA correctly detects most unreliable disparity measurements.

5.7 Conclusions

In this chapter, we have reviewed stereo vision algorithms that, with appropriate modifications, are suited for implementation on a basic computing architecture made of a single low-cost FPGA without additional external devices. Algorithms mapped on such architecture provide accurate and dense depth maps in real time enabling to obtain a small, low-power, and low-cost RGBD stereo vision sensor self-contained into an FPGA.

Acknowledgments I'd like to thank Ilario Marchio, Marco Casadio, Stefano Bruciati, Michael Cavina, Simone Calisesi and Marco Destro for the experimental results reported in this paper.

References

1. Aysu A, Sayinta M, Cigla C (2013) Low cost FPGA design and implementation of a stereo matching system for 3D-TV applications. In: VLSI-SoC, pp 204–209
2. Bailey DG (2011) Design for embedded image processing on FPGAs. Wiley, Asia

[2] Other videos available at:
http://www.youtube.com/channel/UChkayQwiHJuf3nqMikhxAlw.

3. Banz C, Hesselbarth S, Flatt H, Blume H, Pirsch P (2010) Real-time stereo vision system using semi-global matching disparity estimation: architecture and FPGA-implementation. In: ICSAMOS, pp 93–101
4. Chang NY-C, Tsai T-H, Hsu P-H, Chen Y-C, Chang T-S (2010) Algorithm and architecture of disparity estimation with mini-census adaptive support weight. IEEE Trans Circuits Syst Video Techn 20(6):792–805
5. Cigla C, Alatan AA (2011) Efficient edge-preserving stereo matching. In: ICCV 2001 workshops, pp 696–699
6. Mutto CD, Zanuttigh P, Mattoccia S, Cortelazzo G (2012) Locally consistent ToF and stereo data fusion. In: Proceedings of 2nd workshop on consumer depth cameras for computer vision, ECCV'12, pp 598–607
7. De-Maeztu L, Mattoccia S, Villanueva A, Cabeza R (2011) Linear stereo matching. In: ICCV: 2011, pp 1708–1715
8. Ding J, Liu J, Zhou W, Yu H, Wang Y, Gong X (2011) Real-time stereo vision system using adaptive weight cost aggregation approach EURASIP. J Image Video Process 1:1–19
9. Gehrig KS, Eberli F, Meyer T (2009) A real-time low-power stereo vision engine using semi-global matching. In: ICVS, pp 134–143
10. Geiger A, Lenz P, Urtasun R (2012) Are we ready for autonomous driving? the KITTI vision benchmark suite. In: CVPR 2012, Providence
11. Goldberg SB, Matthies LH (2011) Stereo and IMU assisted visual odometry on an omap3530 for small robots. In: ECVW 2011, pp 169–176
12. He K, Sun J, Tang X (2010) Guided image filtering. In: ECCV 2010, pp 1–14
13. Hirschmüller H (2008) Stereo processing by semiglobal matching and mutual information. IEEE Trans Pattern Anal Mach Intell 30(2):328–341
14. Hirschmüller H, Scharstein D (2009) Evaluation of stereo matching costs on images with radiometric differences. IEEE Trans Pattern Anal Mach Intell 31(9):1582–1599
15. Hosni A, Bleyer M, Gelautz M (2013) Secrets of adaptive support weight techniques for local stereo matching. Comput Vis Image Underst 117(6):620–632
16. Jin M, Maruyama T (2014) Fast and accurate stereo vision system on FPGA. ACM Trans Reconfigurable Technol Syst 7(1):3:1–3:24
17. Jin S, Cho J, Pham XD, Lee KM, Park S-K, Kim M, Jeon JW (2010) FPGA design and implementation of a real-time stereo vision system. IEEE Trans Circuits Syst Video Technol 20(1):15–26
18. Lan Z-D, Mohr R, Remagnino P (1995) Robust matching by partial correlation. In: BMVC, pp 1–10
19. Mattoccia S, Giardino S, Gambini A (2009) Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering. In: ACCV 2009, pp 23–27
20. Mattoccia S (2010) Fast locally consistent dense stereo on multicore. In: Sixth IEEE embedded computer vision workshop, San Francisco
21. Mc Donnel M (1981) Box-filtering techniques. Comput Gr Image Process 17:65–70
22. Mei X, Sun X, Zhou M, Jiao S, Wang H, Zhang X (2011) On building an accurate stereo matching system on graphics hardware. In: ICCV workshops, pp 467–474
23. Min D, Lu J, Do MN (2011) A revisit to cost aggregation in stereo matching: How far can we reduce its computational redundancy? In: ICCV 2011, pp 1567–1574
24. Motten A, Claesen L, (2010) A binary adaptable window SoC architecture for a stereo vision based depth field processor. In: VLSI-SoC, pp 25–30
25. Mutto CD, Zanuttigh P, Cortelazzo GM (2012) Time-of-flight cameras and microsoft kinect (TM). Springer Publishing Company, Incorporated
26. Paris S, Kornprobst P, Tumblin J (2009) Bilateral filtering. Now Publishers Inc., Hanover
27. Rhemann C, Hosni A, Bleyer MC, Gelautz M (2011) Fast cost-volume filtering for visual correspondence and beyond. In: CVPR 2011, pp 3017–3024
28. Scharstein D, Szeliski R, http://vision.middlebury.edu/stereo/

29. Scharstein D, Szeliski R (2002) A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Int J Comput Vis 47(1/2/3):7–42
30. Stein F (2004) Efficient computation of optical flow using the census transform. In: Rasmussen CE, Bülthoff HH, Schölkopf B, Giese MA (eds) Proceedings of the 26th DAGM symposium. Lecture Notes in Computer Science, vol 3175. Springer, pp 79–86
31. Szeliski R (2010) Computer vision: algorithms and applications. Springer New York Inc, New York
32. Szeliski R, Zabih R, Scharstein D, Veksler O, Kolmogorov V, Agarwala A, Tappen M, Rother C (2008) A comparative study of energy minimization methods for Markov random fields with smoothness-based priors. IEEE Trans Pattern Anal Mach Intell 30(6):1068–1080
33. Tippetts B, Lee DJ, Lillywhite K, Archibald J (2013) Review of stereo vision algorithms and their suitability for resource-limited systems. J Real-Time Image Process
34. Tomasi C, Manduchi R (1998) Bilateral filtering for gray and color images. In: ICCV98, pp 839–846
35. Tombari F, Mattoccia S, Di Stefano L (2007) Segmentation-based adaptive support for accurate stereo correspondence. In: Proceedings of PSIVT 2007
36. Tombari F, Mattoccia S, Di Stefano L, Addimanda E (2008) Classification and evaluation of cost aggregation methods for stereo correspondence. In: CVPR08, pp 1–8
37. Ttofis C, Hadjitheophanous S, Georghiades AS, Theocharides T (2013) Edge-directed hardware architecture for real-time disparity map computation. IEEE Trans Comput 62(4):690–704
38. Ttofis C, Theocharides T (2012) Towards accurate hardware stereo correspondence: a real-time fpga implementation of a segmentation-based adaptive support weight algorithm. In: DATE, pp 703–708
39. Villalpando CY, Morfopolous A, Matthies L, Goldberg S (2011) FPGA implementation of stereo disparity with high throughput for mobility applications. In: Proceedings of the 2011 IEEE aerospace conference, AERO'11DC, Washington, pp 1–10
40. Viola P, Jones MJ (2004) Robust real-time face detection. Int J Comput Vis 57(2):137–154
41. Paul V, Wells WM (1997) III. Alignment by maximization of mutual information. Int J Comput Vis 24(2):137–154
42. Wang L, Gong MW, Gong ML, and Yang RG (2006) How far can we go with local optimization in real-time stereo matching. In: Proceedings of the third international symposium on 3D data processing, visualization, and transmission (3DPVT 2006), pp 129–136
43. Wang L, Liao M, Gong M, Yang R, Nister D (2006) High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In: Proceedings of 3DPVT 06, pp 798–805
44. Xilinx. www.xilinx.com
45. Yang Q (2012) A non-local cost aggregation method for stereo matching. In: CVPR 2012, pp 1402–1409
46. Yoon KJ, Kweon IS (2006) Adaptive support-weight approach for correspondence search. IEEE Trans PAMI 28(4):650–656
47. Zabih R, Woodfill J (1994) Non-parametric local transforms for computing visual correspondence. In: ECCV 1994. Springer, pp 151–158
48. Zhang K, Lu J, Lafruit G (2009) Cross-based local stereo matching using orthogonal integral images. IEEE Trans Circuits Syst Video Technol 19(7):1073–1079
49. Zhang L, Zhang K, Chang TS, Lafruit G, Kuzmanov GK, Verkest D (2011) Real-time high-definition stereo matching on FPGA. In: Proceedings of the 19th ACM/SIGDA international symposium on field programmable gate arrays, FPGA'11, pp 55–64
50. Zicari P, Perri S, Corsonello P, Cocorullo G (2012) Low-cost FPGA stereo vision system for real time disparity maps calculation. Microprocess Microsyst 36(4):281–288