Chapter 3
Stereo Matching and Viewpoint Synthesis FPGA Implementation

Chao-Kang Liao, Hsiu-Chi Yeh, Ke Zhang, Vanmeerbeeck Geert, Tian-Sheuan Chang and Gauthier Lafruit

Abstract With the advent of 3D-TV, the increasing interest of free viewpoint TV in MPEG, and the inevitable evolution toward high-quality and higher resolution TV (from SDTV to HDTV and even UDTV) with comfortable viewing experience, there is a need to develop low-cost solutions addressing the 3D-TV market. Moreover, it is believed that in a not too distant future 2D-UDTV display technology will support a reasonable quality 3D-TV autostereoscopic display mode (no need for 3D glasses) where up to a dozens of intermediate views are rendered between the extreme left and right stereo video input views. These intermediate views can be synthesized by using viewpoint synthesizing techniques with the left and/or right image and associated depth map. With the increasing penetration of 3D-TV broadcasting with left and right images as straightforward 3D-TV broadcasting method, extracting high-quality depth map from these stereo input images becomes mandatory to synthesize other intermediate views. This chapter describes such "Stereo-In to Multiple-Viewpoint-Out"

C.-K. Liao (✉)
IMEC Taiwan Co., Hsinchu, Taiwan
e-mail: ckliao@imec.be

H.-C. Yeh · K. Zhang · V. Geert · G. Lafruit
IMEC vzw, Leuven, Belgium
e-mail: lalalachi@gmail.com

K. Zhang
e-mail: zhangke@imec.be

V. Geert
e-mail: vanmeerb@imec.be

T.-S. Chang
National Chiao Tung University, Hsinchu, Taiwan
e-mail: tschang@twins.ee.nctu.edu.tw

G. Lafruit
e-mail: lafruit@imec.be

C. Zhu et al. (eds.), *3D-TV System with Depth-Image-Based Rendering*,
DOI: 10.1007/978-1-4419-9964-1_3,

functionality on a general FPGA-based system demonstrating a real-time high-quality depth extraction and viewpoint synthesizer, as a prototype toward a future chipset for 3D-HDTV.

Keywords 3D-TV chipest · Autostereoscopic · Census Transform · Computational complexity · Cross-check · Depth map · Disparity · Dynamic programming · Free viewpoint TV · FPGA · GPU · Hamming distance · Hardward implementation · Hole filling · Matching cost · Real-time · Stereo matching · Support region builder · View synthesis · Warping

3.1 Introduction

In recent years, people get more and more attracted by the large range of applications enabled by 3D sensing and display technology. The basic principle is that a vision device providing a pair of stereo images to the left and right eye respectively is sufficient to provide a 3D depth sensation through the brain's natural depth interpretation ability. Many applications like free viewpoint TV and 3D-TV benefit from this unique characteristic.

3.1.1 Stereoscopic Depth Scaling

Often in the early days of 3D movie theaters, 3D content was captured with exaggerated depth to give a spectacular depth sensation at the cost of sometimes visual discomfort. Nowadays, the depth perception has been greatly reduced to an average degree of comfort to all viewers. Anyway, depth perception is subjective and also depends on the viewing distance and display size. Consequently, as there exists contrast and color/luminance controls on each 2D TV set, future 3D TV sets will also add the 3D depth scale control, as shown in Fig. 3.1.

Such depth scaling can be achieved by calculating a new left/right image pair from the original left/right stereo images. In essence, pixels in the images have to be shifted/translated over a distance depending on their depth; hence, such viewpoint synthesis system is very similar to the free viewpoint TV of Fig. 3.2, with the difference that two viewpoints are rendered instead of one, for achieving 3D stereoscopic rendering.

3.1.2 3D Autostereoscopic Displays

Undoubtedly, the next step in watching 3D content is to use a glasses-free 3D display solution where many intermediate views between extreme left and right captured views are projected toward the viewer. The eyes then capture two of these dozens of viewing cones, and lateral movements of the viewer will position the

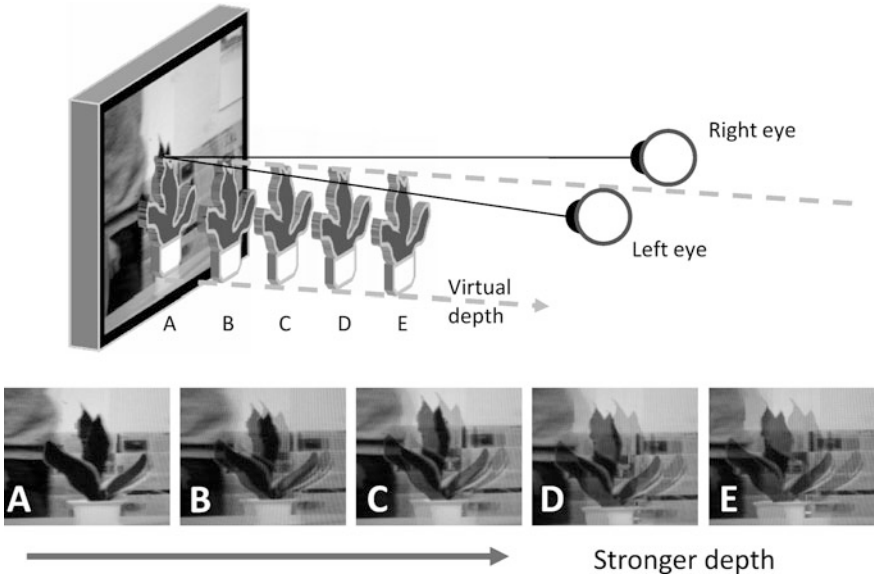

Fig. 3.1 Synthesized virtual images with different level of depth control

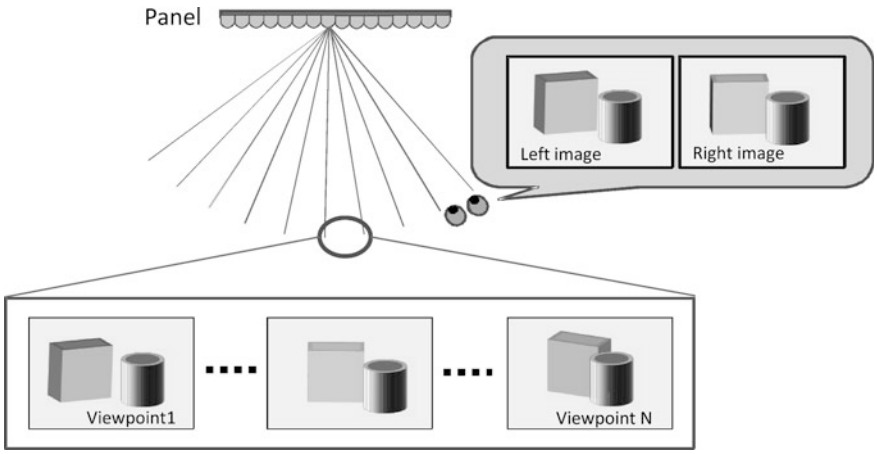

Fig. 3.2 Schematic of an autostereoscopic display with multiple synthesized views

viewer's eyes into the two corresponding adjacent viewing cones, giving the viewer the illusion to turn around the projected 3D content.

Though it is conceivable that the accompanying viewpoint interpolation process requiring scene depth information might be based on sender-precalculated depth maps broadcasted through the network, it is more likely that today's left/right stereo video format used in 3D digital cinema and 3D bluray with polaroid/shutter-glasses display technology will become legacy in tomorrow's 3D-TV broadcasting

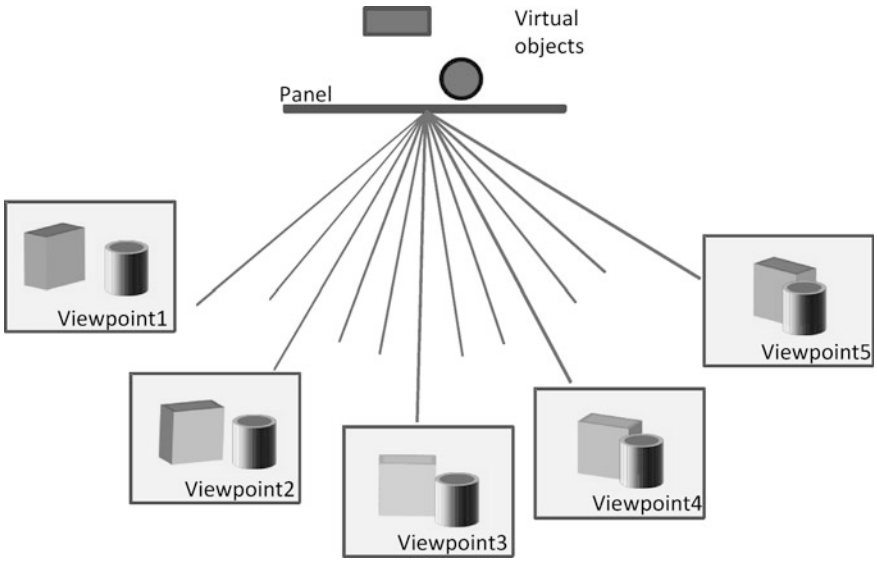

Fig. 3.3 Free viewpoint TV: the images of a user-defined viewing direction are rendered

market. Each autostereoscopic 3D-TV (or settop-box) receiver will then have to include a chipset for depth estimation and viewpoint synthesis at reasonable cost.

3.1.3 Free Viewpoint TV

Free viewpoint TV projects any viewer selected viewpoint onto the screen, in a similar way as one chooses the viewpoint in a 3D game with the mouse and/or joystick (or any gesture recognition system), as shown in Fig. 3.3. The difference in free viewpoint TV however is that the original content is not presented in a 3D polygonal format; the original content consists only of the images captured by a stereo camera system, and any intermediate viewpoint between the two extreme viewpoints (viewpoint 1 and 5 in Fig. 3.3) has to be synthesized by first detecting depth and occlusions in the scene, followed by a depth-dependent warping and occlusion hole filling (inpainting) process. In Fig. 3.3, five viewpoints can be rendered and only one of the viewpoints 2, 3, and 4 is actually synthesized as a new free viewpoint TV viewpoint.

3.1.4 3D Gesture Recognition

In recent years, gesture detection for interfacing with gaming consoles, such as Nintendo Wii and Microsoft Kinect for Xbox360, has gained a high popularity. Games are now controlled with simple gestures instead of using the old-fashioned joystick. For instance, the Kinect console is equipped with an infrared camera that

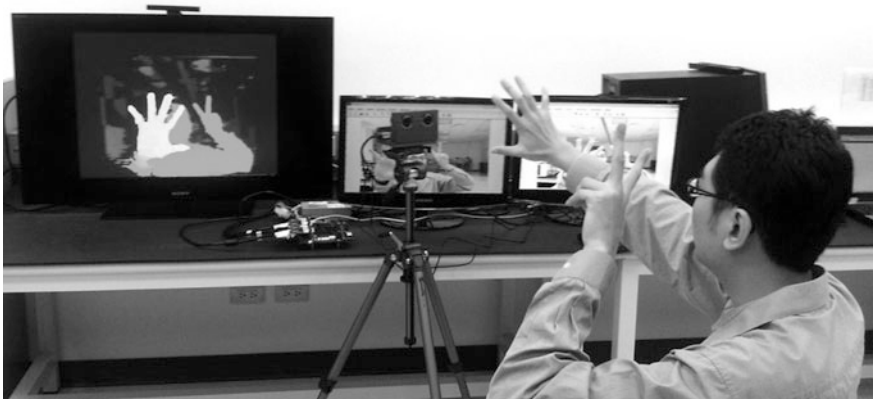

Fig. 3.4 Depth measurement for 3D gesture recognition. The monitor shows the depth-image captured and calculated by a FullHD stereo-camera with the proposed system

projects a pattern onto the scene, out of which the player's 3D movements can be extracted.

The same function can be achieved using a stereo camera where depth is extracted by stereo matching (Fig. 3.4). There is much benefit from using a stereo camera system: the device allows higher depth precision (active depth sensing devices always exhibit a low output image resolution), and the system can be used outdoors, even under intense sunlight conditions, which is out of reach for any IR active sensing device. Finally, compared with the active IR lighting systems, the stereo approach exhibits lower energy consumption. Therefore, using stereo cameras with a stereo matching technique has a high potential in replacing Kinect-alike devices.

Although stereo matching provides depth information providing added-value to a gesture recognition application, it remains a challenge to have a good image quality at a reasonable computational complexity. Since today even GPUs with their incredible processing power are hardly capable of performing such tasks in real time, we propose the development of a stereo matching and viewpoint interpolation processing chain, implemented onto FPGA, paving the way to future 3D-TV chipset solutions. In particular, we address the challenge of proper quality/complexity tradeoffs satisfying user requirements at minimal implementation cost (memory footprint and gate count). The following describes a real-time FPGA implementation of such stereo matching and viewpoint synthesis functionality. It is also a milestone to eventually pave the way to full-HD 3D on autostereoscopic 3D displays.

3.2 Stereo Matching Algorithm Principles

Depth map extraction from the stereo input images is mostly based on measuring the lateral position difference of an object shown on two horizontally separated stereo images. The lateral position difference, called "*disparity*", is a consequence

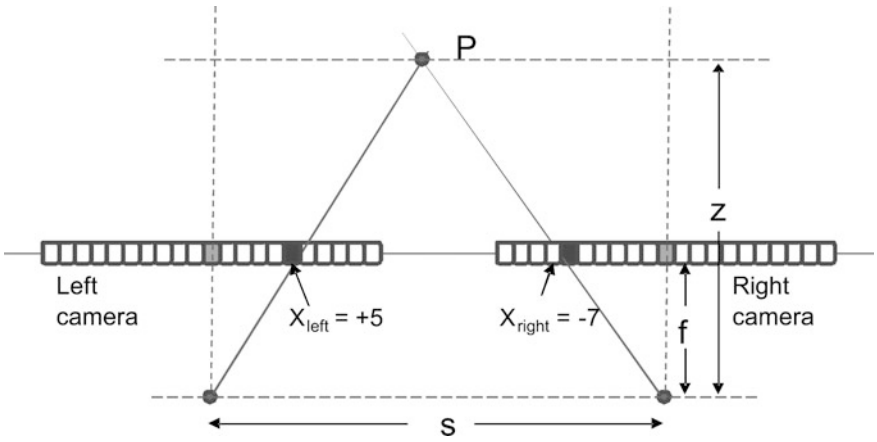

Fig. 3.5 The schematic model of a stereo camera capturing an object *P*. The cameras with focal length *f* are assumed to be identical and are located on an epipolar line with a spacing *S*

of different light paths from the object to respectively the left and right cameras with their corresponding location and viewing angle [1], mimicking the human visual system with its depth impression and 3D sensation.

Figure 3.5 shows an illustration of disparity and the corresponding geometrical relation between an object and a stereo camera rig. Two images from the object are captured by two cameras which are assumed to be perfectly parallel and aligned to each other on a lateral line, corresponding to the so-called epipolar line [2]. A setup with converging cameras can be transformed back and forth to this scenario through image rectification. To simplify the discussion, the camera parameters (except their location) are identical.

In this chapter, we assume that the rectification has been done in advance. The origins of the cameras are shown to be on their rear side, corresponding to their respective focal point. When an object image is projected to the camera, each pixel senses the light along the direction from the object to the camera origin. Consequently, any camera orientation has its own corresponding object image and therefore the object is at a different pixel location in the respective camera views. For example, the object on point "P" is detected in pixel "$X_{\text{left}} = +5$" and "$X_{\text{right}} = -7$" respectively in the left and right cameras (see Fig. 3.5). The disparity is calculated as the difference of these two lateral distances (i.e., $5 + 7 = 12$). The disparity is a function of the depth z, the spacing between these two cameras s, and focal length f of the camera as shown in Eq. (3.1).

$$z = sf/d \tag{3.1}$$

According to this equation the object depth can be calculated by measuring the disparity from the stereo images. In general, textures of an object on the respective stereo images exhibit a high image correlation, both in color and shape, since they originate from the same object with the same illumination environment and a

similar background and are assumed to be captured by identical cameras. Therefore, it is suitable to measure the disparity by detecting the image texture displacement with a texture similarity matching metric.

According to the taxonomy summarized by [3] and [4], algorithms for stereo matching have been investigated for mostly 40 years, and can be categorized into two classes, the local-approach and the global-approach algorithms. The stereo matching computational flow can be summarized in four steps:

1. Matching cost computation.
2. Cost aggregation.
3. Disparity computation.
4. Disparity map refinement.

where the local-approach algorithms generally perform steps 1, 2, 3, and 4 but without global optimization; whereas global-approach algorithms commonly perform steps 1, 3, and 4.

In the local-approach algorithms, the disparity is calculated from the matching cost information of matching candidates that are only included in a preset local disparity range. In order to increase the matching accuracy, local approaches rely on cost aggregation to include more matching cost information from neighborhood pixels [3]. The global-approach algorithms, on the other hand, concentrate more on disparity computation and global optimization by utilizing energy functions and smoothness assumptions. The energy function includes the matching cost information from the entire image. It reduces the matching failure in some areas due to the noise, occlusion, texture less, or repetitive pattern, etc. Although global-approach methods render a good quality disparity map by solving these issues, they always put higher computational complexity demands and memory resources, compared to the local-approach methods, making a real-time system implementation challenging [3]. All in all, there exists a tradeoff between matching quality and computational complexity.

In this section, we introduce the aforementioned four computation steps with regard to their applicability in hardware system setups without impeding on the resulting visual quality.

3.2.1 Matching Cost Calculation

The fundamental matching cost calculation is based on pixel-to-pixel matching within one horizontal line within a certain disparity range. The image intensity of a pixel on the reference image is compared with that of the target stereo image with different disparities as shown in Fig. 3.6.

The cost is a function of disparity that shows how much the similarity is in between the chosen stereo regions. Several methods, including summation of absolute intensity difference (SAD), summation of squared intensity difference (SD), normalized cross-correlation (NCC), and census transform (CT), etc. [3],

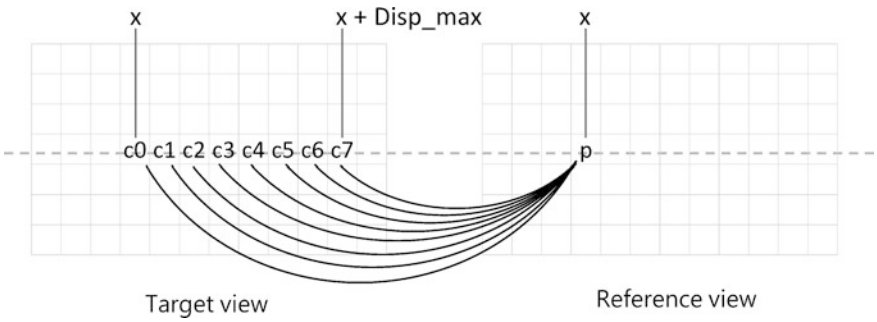

Fig. 3.6 Fundamental pixel-to-pixel matching method

Table 3.1 General matching cost calculation methods

Match metrics	Definition	Point operation
NCC	$\frac{\sum_{x,y}\left(I_r(x,y)-\tilde{I}_t\right)\left(I_t(x+d,y)-\tilde{I}_r\right)}{\sqrt{\sum_{x,y}\left(I_r(x,y)-\tilde{I}_r\right)^2\left(I_t(x+d,y)-\tilde{I}_t\right)^2}}$	Multiplication
SAD	$\sum_{x,y}\lvert I_r(x,y)-I_t(x+d,y)\rvert$	Subtraction
SSD	$\sum_{x,y}(I_r(x,y)-I_t(x+d,y))^2$	Subtraction and squarer
CT	$I_k(x,y)=\text{Bitstring}_{m,n}(I_k(m,n)<I_k(x,y))$ $\sum_{x,y}\text{Hamming}(I_r(x,y),I_t(x+d,y))$	XOR

have been proposed to calculate the matching cost. The definitions of these methods with their point operation are shown in Table 3.1 [5].

Cross-correlation is a commonly used operation to calculate the similarity in the standard statistics methods [6]. Furthermore, NCC is also reported in order to prevent the influence of radiometric differences between the stereo images to the cost estimation result [7, 8]. Cross-correlation is however computationally complex and thus only few researches use this method to calculate the matching cost in a real-time system [9]. Similar to SSD, SAD mainly measures the intensity difference between the reference region and the target region. Comparing to the SSD, SAD is often used for system implementation since it is more computationally efficient. Different from these approaches, the CT performs outstanding similarity measurements at acceptable cost [10]. The CT method firstly translates the luminance comparison result between the processed anchor pixel (the central pixel of the moving window) and the neighboring pixels into a bit string. Then it generates the matching cost by computing the hamming distance between these bit strings on the reference and target images. As such, the CT matching cost includes the luminance relative information, not the luminance absolute values, making it more tolerant against luminance error/bias between the two regions to match, while increasing the discrimination level by recording a statistical distribution rather than individual pixel values.

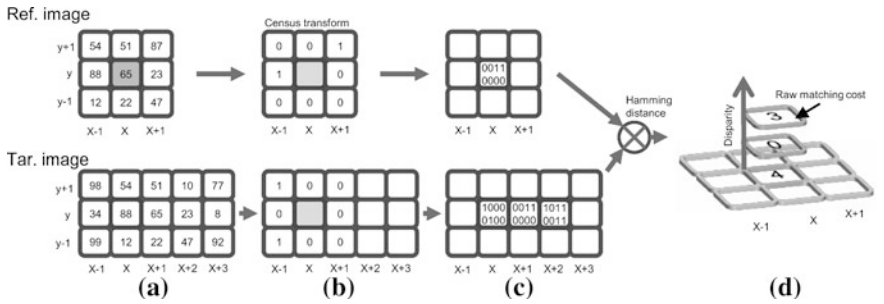

Fig. 3.7 The CT and hamming distance calculation resulting in a raw matching cost in terms of disparities. **a** The image intensity map. **b** The CT. **c** The census vector aggregated from neighbor pixels, and **d** the calculated raw matching cost. Note that the processed anchor pixel ("65") is at coordinates (x, y)

With a predetermined area of pixels the CT results in a census vector (a binary array with the same number of bits as the number of pixels in the window region around the pixel under study) that can be used as a representation of the processed anchor pixel, which is the central pixel of the window (i.e., pixel at (x, y) in Fig. 3.7). These bits are set to '1' if the luminance of the processed anchor pixel is greater than the neighboring pixels, otherwise they are set to '0' (see Fig. 3.7). Although this transform cannot exactly represent the uniqueness of the window center pixel, the involved calculations like comparators and vector forming are hardware friendly. After the CT, both left/right images become left/right census arrays, as shown in Fig. 3.7. Consequently, the pixel characteristic is no longer represented by its luminance, but rather by an aggregated vector that also includes the relative luminance difference from the neighborhood of the processed anchor pixel.

The cost aggregation step accumulates the neighboring matching costs within a support region in order to increase the matching accuracy. For the hardware implementation, the chosen cost aggregation method depends on the hardware cost and the quality requirement of the disparity map. In general, there are four types of cost aggregation strategies from coarse to fine: fixed window, multiple windows, adaptive shape, and adaptive weight as shown in Fig. 3.8. The fixed window approach sums up the matching cost within a simple window region. It yields the lowest computational complexity but performs weakly in discontinuous, textureless, and repetitive regions. To overcome the problems of the fixed window method, multiple windows is an advanced approach, combining several subwindows. A number of subwindows are predefined to make up the support region, which is not limited to be of rectangular shape. Another approach is the adaptive shape method, where the matching cost aggregation is performed within a region of neighboring pixels exhibiting similar color intensities. This method is able to preserve object boundaries. Zhang et al. propose the cross-based method [11] which is an example of the adaptive shape approach with acceptable computational complexity. Finally, the most accurate method is the adaptive weight approach [12–14], where the nearest pixels with similar intensity around the central pixel have higher probability to share

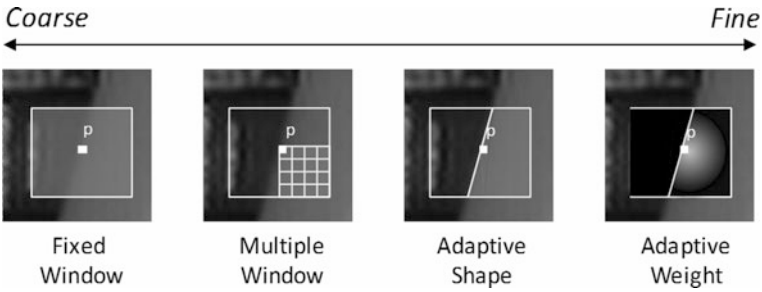

Fig. 3.8 Categories of local stereo matching algorithms based on the level of detail in the support region of neighboring pixels around the processed anchor pixel. Four categories are listed from course to fine, i.e., fixed window methods, multiple windows methods, adaptive shape methods, and adaptive weight methods. One possible support region to the pixel p is presented for each category, respectively (brightness denotes the weight value for the adaptive weight methods)

the same disparity value. Typically, the Gaussian filter is used to decide the weight of aggregation matching cost based on the distance to the processed anchor pixel. Although the adaptive weight method is able to achieve very accurate disparity maps, the computational complexity is relatively higher than other approaches.

Conventional cost aggregation method within the support region requires a computational complexity in $O(n^2)$, where n is the size of support region window. Wang et al. [15] propose a two-pass cost aggregation method, which efficiently reduces the computational complexity to $O(2n)$. Other techniques for hardware complexity reduction use data-reuse techniques. Chang et al. [16] propose the mini-CT and partial column reuse techniques to reduce the memory bandwidth and computational complexity. It caches each column data over the overlapped windows. In our previous hardware implementation work which was contributed from Lu et al. [17–19], the two-pass cost aggregation approach was chosen and associated with the cross-based support region information. Data-reuse techniques were also applied to both horizontal and vertical cost aggregation computations to construct a hardware efficient architecture.

A more recent implementation has modified the disparity computations as explained in the following sections.

3.2.2 Disparity Computation

Disparity computation approaches can be categorized in two types: local stereo matching and global optimization. Local stereo matching algorithms utilize winner take all (WTA) strategies to select the disparity which possesses the minimum matching cost value. The chosen displacement is regarded as the disparity value [20]. Global stereo matching optimization algorithms compute the disparity value with the help of an energy function, which introduces a smoothness assumption. Additional constrains are inserted to support smoothness such as penalizing sudden

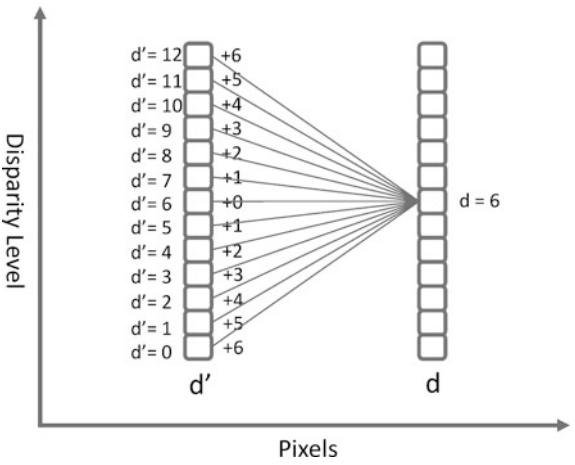

Fig. 3.9 Example of smoothness cost penalties in Linear model

changes in disparity. The energy function can be simply represented as in Eq. (3.2).

$$E(d)=E_{\text{data}}(d)+E_{\text{smooth}}(d) \tag{3.2}$$

The first term represents the sum of matching costs for the image pixels p at their respective disparities d_p. Equation (3.3) shows the function of first term of Eq. (3.2):

$$E_{\text{data}}(d) = \sum\nolimits_{p\in N} C(p, d_p) \tag{3.3}$$

where N represents the total number of pixels, and d_p ranges from 0 to a chosen maximum disparity value. The second term of Eq. (3.2) represents the smoothness function as in Eq. (3.4). We compare two models, Linear and Potts models, to implement the smoothness function:

$$E_{\text{smooth}}(d) = \lambda \sum\nolimits_{p,q\in N} S(d_p, d_q) \tag{3.4}$$

where λ is a scaling coefficient, which adapts to the local intensity changes in order to preserve disparity discontinuous regions. d_p and d_q are disparity arrays in adjacent pixels p and q. Equation (3.5) shows the Linear model for the smoothness function. The smoothness cost of the linear model depends on the difference between the disparity from the current pixel and all candidate disparities from the previous pixel. A higher disparity difference introduces a higher smoothness cost penalty. Figure 3.9 is an example demonstrating the smoothness cost penalties in the Linear model. This mechanism helps to preserve the smoothness of the disparity map when performing global stereo matching.

$$S(d_p, d_q) = \left|d_p - d_q\right| \tag{3.5}$$

where d_p and $d_q \in$ [0, disparity range−1].

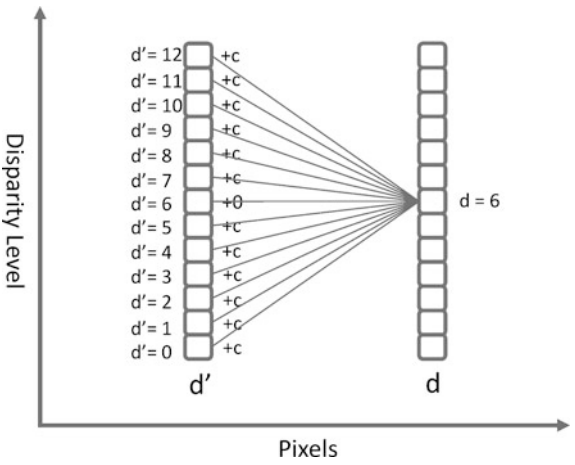

Fig. 3.10 Example of smoothness cost penalties in Potts model

Unfortunately, the linear model smoothness function requires high computational resources. Assuming that the maximum disparity range is D, the computational complexity of the smoothness cost is $O(D^2)$.

Equation (3.6) shows the Potts model smoothness function which has lower complexity than the linear model. The Potts model introduces the same smoothness cost penalty to all candidate disparities. Figure 3.10 is an example demonstrating the smoothness cost penalties in the Potts model. If the disparity value is different, the smoothness cost penalty is introduced into the energy function to reduce the probability of different disparities to win in the winner-take-all computation. In the Potts model smoothness function, the computational complexity is of $O(D)$.

$$S(d_p, d_q) = \begin{cases} 0, & d_p = d_q \\ c, & \text{others} \end{cases} \tag{3.6}$$

where c is a constant to introduce smoothness penalty.

Figure 3.11 shows an example of disparity maps that are generated from Linear and Potts model individually. Obviously, the linear model performs better at slant surfaces and reveals more detail. However, it also blurs the disparity map on discontinuous regions (such as object edges).

The state-of-the-art global stereo matching approaches include dynamic programming (DP) [21], graph cut (GC) [22], and belief propagation (BP) [23]. The principle of those approaches is based on the way one minimizes the energy function. In the following, we choose the DP approach for the example of the hardware implementation that we will study later.

DP is performed after the calculation of the raw matching cost. To simplify the computational complexity, DP generally refers to executing the global stereo matching within an image scanline (an epipolar line after image rectification). It consists of a forward updating of the raw matching cost and a backward minimum tracking. The DP function helps to choose the accurate minimum disparity value

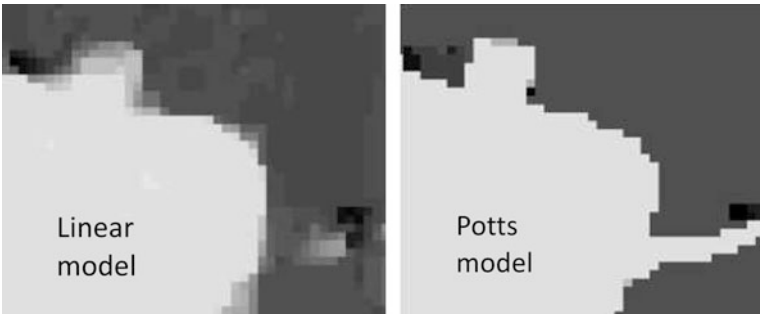

Fig. 3.11 Disparity map results by using different smoothness function models

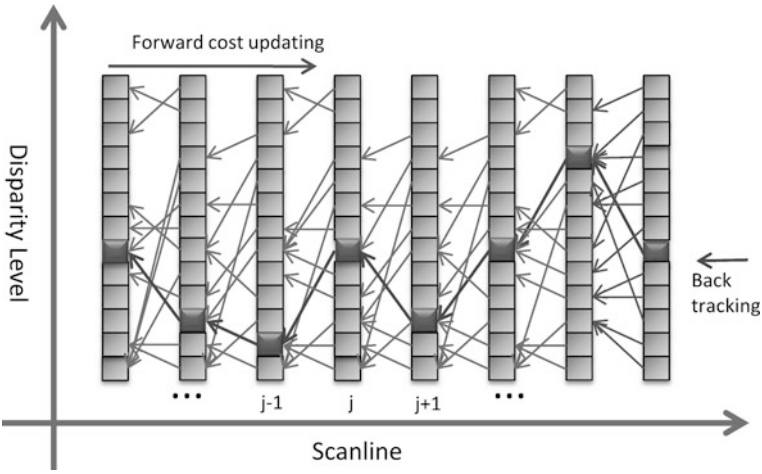

Fig. 3.12 DP consists of a forward cost update and a backward tracking. In the forward process, the costs are updated based on the cost distribution of the previous pixel, and therefore, the final disparity result is not only determined by its original cost map, but also affected by neighboring pixels

among the somewhat noise-sensitive row matching cost disparity candidates by considering the cost of previous pixels. The forward pass of DP can be expressed as:

$$C_w(j, d_q) = c(j, d_q) + \min_{d_p \in w} \{c_w(j-1, d_p) + s(d_p, d_q)\} \tag{3.7}$$

where the function $C()$ presents the original raw matching cost at disparity d_q for pixel j on scanline W, and $C_w()$ function is the accumulated cost updated from the original cost. $S()$ is the smoothness function that introduces a penalty to the disparity difference between the current and previous scanline pixels. (Fig. 3.12).

After updating the cost of every pixel on the scanline, the optimal disparity result of a scanline can be harvested by backward selecting the disparity with minimum cost, according to:

$$d(w) = \arg \min_{d' \in [0,D]} [c(w, d')] \tag{3.8}$$

3.2.3 Disparity Map Refinement

This is the final step that pushes the quality of the disparity map to an even higher accuracy. Many articles have reported their technology on fine-tuning the resulting disparity map based on reasonable constraints. In this subsection, we introduce cross-checks, median filters, and disparity voting methods. Those techniques contribute to a significant improvement of the quality of the disparity map [24].

3.2.3.1 Cross-Check

Cross-check techniques check the consistency between the left and right disparity maps. It is based upon the assumption that the disparity should be identical from the left to the right image and vice versa. If the disparity values are not consistent, they either represent mismatching pixels or occlusion pixels. The cross-check formulas can be expressed as:

$$\begin{cases} |D(x,y) - D'(x - D(x,y), y)| \leq \lambda, & \text{left\_disparity} \\ |D'(x,y) - D(x + D(x,y), y)| \leq \lambda, & \text{right\_disparity} \end{cases} \tag{3.9}$$

where D and D' represent the left and right disparity maps from the disparity computation step, respectively. The constant λ is the cross-check threshold for tolerating trivial mismatching. After knowing the occlusion regions, the simplest solution to solve the inconsistent disparity pixels is to replace them by the nearest good disparity value [25]. Another solution is to replace the inconsistent disparity pixels by the most popular good disparity value within the support region (similar texture) by disparity voting [25].

3.2.3.2 Disparity Voting

Following the cross-check, a voting process is performed by selecting the most popular disparity value within the support region. Disparity voting is based on the assumption that the pixels within a similar texture-region share the same disparity value. This method helps to improve the quality of the disparity map. In this refinement process, a support region is used as the voting area, and only the pixels in this irregularly shaped region (as shown in Fig. 3.13) are taken into account. In the following, we introduce the cross-based support region generation method that is proposed by Zhang et al. [12].

The arm stretches to left/right and top/bottom directions respectively and stops when two conditions are satisfied. First the arm is limited to a maximum length. Second, the arm stops when two adjacent pixels are not consistent in color, according to:

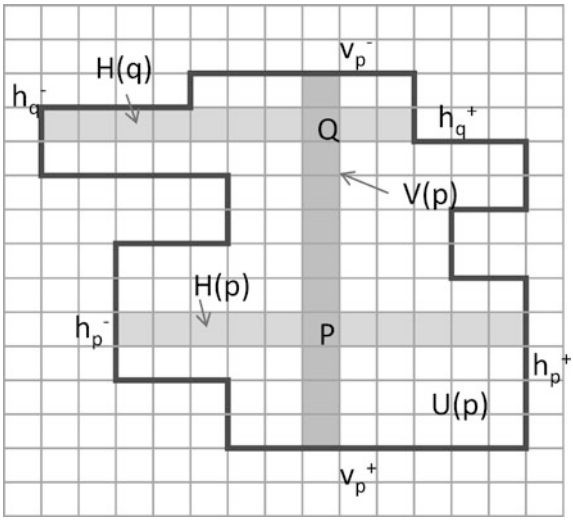

Fig. 3.13 Support region $U(p)$ of the anchor processed pixel p is used to take every pixel inside into account. It is a region defined by four stretched arms

$$r^* = \max_{r\in[1,L]}\left(r \prod_{i\in[1,r]} \delta(p, p_i)\right) \tag{3.10}$$

where r^* indicates the largest span for the four direction arms. In the equation $p_i = (x_p - i, y_p)$ and L corresponds to the predetermined maximum arm length.

$$\delta(p_1, p_2) = \begin{cases} 1, & \max\limits_{c\in[R,G,B]}(|I_c(p_1) - I_c(p_2)|) \le \tau \\ 0, & \text{otherwise.} \end{cases} \tag{3.11}$$

After the arm lengths are calculated, the four arms (h_p^-, h_p^+, v_p^-, v_p^+) are used to define the horizontal segment $H(p)$ and vertical segment $V(p)$. Note that the full support region $U(p)$ is the integration of all the horizontal segments of those pixels residing on the vertical segment.

Once the support region (indicating the texture-less region) has been determined, the disparity voting method can be performed to provide a better image quality. It accumulates the disparity values within the support region of the central pixel into histogram; and then chooses the winner as the final disparity value. Figure 3.14 demonstrates an example of such 2D voting.

3.3 Co-Designing the Algorithm and Hardware Implementation

Most stereo matching implementation studies in the literature are realized on four common-used platforms: CPU, GPU, DSP, and FPGA/ASIC. To decide about the best implementation platform, different aspects such as matching accuracy,

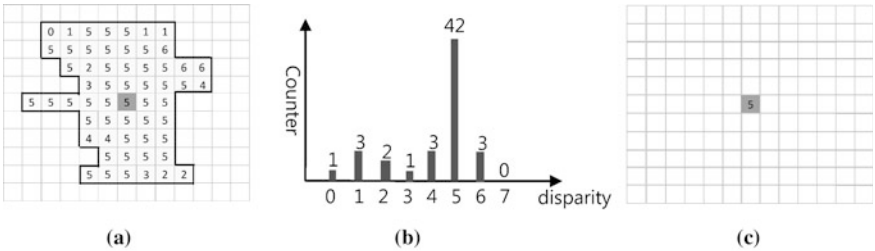

Fig. 3.14 Example of 2D disparity voting. All the disparities in the support region are taken into account in the histogram to choose the most voted disparity. **a** Disparities in support region. **b** disparities in support region **c** voted disparity

robustness, real-time performance, computational complexity, and scalability should be taken into consideration when implementing the algorithm. Implementations with CPU, GPU, and DSP are software-based approaches, which require less development time. Zhang et al. [11] propose a real-time design for accurate stereo matching on CUDA, which achieves an acceptable tradeoff between matching accuracy and computational performance by introducing various sampling rate into the design. Banz et al. [26] apply the semiglobal algorithm that was proposed by Hirschmuller [20] on GPU. However, the computational logic, instructions and data paths are fixed. Besides, high clock frequency and memory bandwidth are required. Therefore, dedicated hardware design is an option to overcome the above-mentioned problems. FPGA and ASIC platforms allow parallelism exploration and a pipeline architecture to achieve a higher throughput at moderate clock speeds. Chang et al. [16] implement a high performance stereo matching algorithm with mini-CT and adaptive support weight on UMC 90 nm ASIC, achieving 352 × 288@42FPS with 64 disparity ranges. Jin et al. [27] design a pipelines hardware architecture with the CT and sum of hamming distances, achieving 640 × 480@30FPS at a 64 disparity range under 20 MHz. Banz et al. realize the semiglobal matching algorithm on a hybrid FPGA/RISC architecture, achieving 640 × 480@30FPS at a 64 disparity range under 12 M∼208 MHz [26]. Stefan et al. [28] also utilize semiglobal matching algorithm to implement a stereo matching engine, which achieves 340 × 200@27FPS with a disparity range of 64 levels. Zhang et al. [12] implement a local algorithm with a cross-based support region in the cost aggregation and refinement stages, achieving over a range of 64 disparity levels with a resolution of 1024 × 768@60FPS under 65 MHz. In this section, we will briefly introduce our stereo matching algorithm and hardware implementation approach to reach an optimal trade-off between efficiency and image quality.

Targeting low-cost solutions, we have reduced the memory size and computational complexity by extracting the necessary depth information from within a small texture size that is sufficient to represent the particular object and its location. There are mainly two reasons to support the depth extraction in such line-based structure. First, video signals are typically fed in a horizontal line-by-line

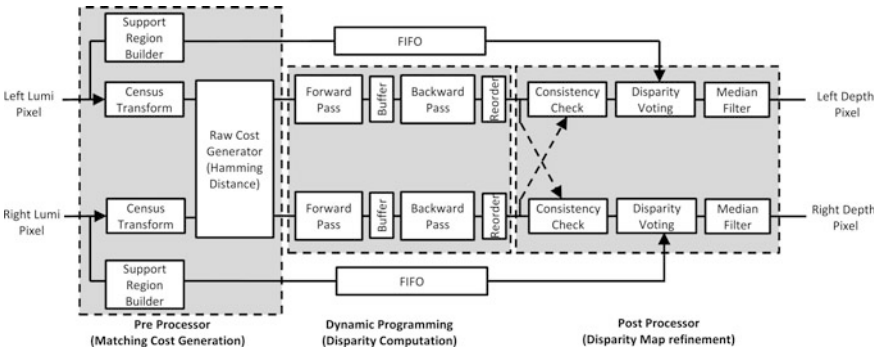

Fig. 3.15 The schematic of stereo matching process flowchart

format. For instance, the video signal onto the television screen are scanned and transferred by either the progressive or the interlaced raster scan direction format, and these scan formats both follow a horizontal direction. All information is extracted from a horizontal strip of successive lines, gradually moving from top to bottom of the image. There is hence no need to store the entire image frame in the processing memory and only a very limited amount of scan lines needed for the processing are stored. Second, the disparity is naturally expressed in the horizontal direction, since it finds its origin in the position difference between the viewer's left and right eyes which are naturally in a horizontal spacing. Due to these reasons, it is more practical to build up a stereo matching system achieving real-time performances in a line-based structure. As long as the depth extraction is simplified as harvesting information from the local data rather than from the frame data, the implementation of the stereo matching onto the FPGA prototyping system becomes more efficient.

Figure 3.15 shows the proposed example in a more detailed flow graph of the stereo matching implementation on FPGA. It performs the following steps:

- Support region builder, raw cost generation with CT, and Hamming distance.
- DP computation.
- Disparity map refinement with consistency check, disparity voting, and median filter.

The hardware architecture of these different steps will be discussed in further details in the following sections.

3.3.1 Preprocessor Hardware Architecture

The preprocessor design contains three components: the CT, the Hamming distance calculation, and support region builder.

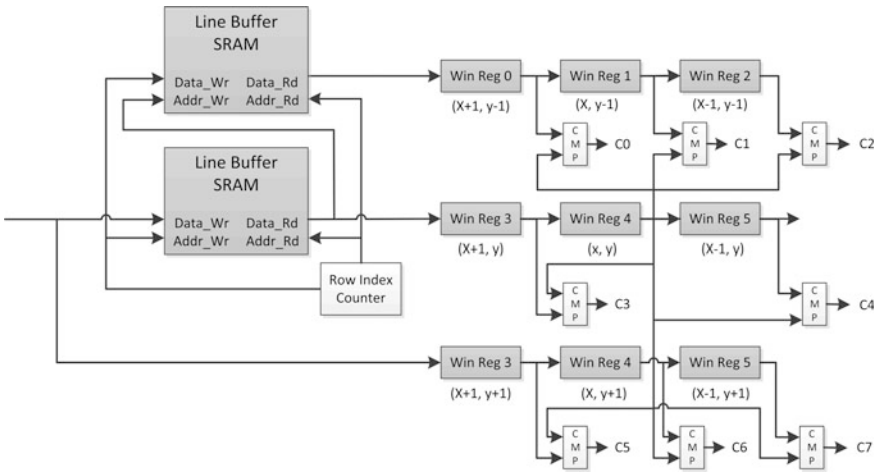

Fig. 3.16 Example of 3 × 3 CT hardware architecture for stream processing

3.3.1.1 CT

In this section, an example design of the CT is shown in Fig. 3.16. The memory architecture uses two line buffers to keep the luminance data of the vertical scanline pixels locally in order to provide two-dimensional pixel information for the CT computation. This memory architecture writes and reads the luminance information of the input pixel streams simultaneously in a circular manner, which keeps the scanline pixel data locally by a data reuse technique. Successive horizontal scanlines are grouped as vertical pixels through additional shift registers. This example illustrates a 3 × 3 CT computational array. The results from the different comparators CMP are concatenated to a census bitstream *ci* for each pixel *i*.

3.3.1.2 Hamming Distance Computation

Once the census vector is obtained, a large amount of comparison between these respective vectors in the left/right images will be performed. For example, when calculating the left depth map, the left census array is set as a reference and the right array as the target. This process is reversed for the right depth map to be calculated. The census vectors on the reference array and their locally corresponding vectors on the target array are compared and the differences (known as the raw matching cost) are quantitatively measured by checking every bit in these two arrays. A large raw matching cost corresponds to a mismatch between the reference pixel and the chosen target pixel from the right census array. The raw matching cost is measured several times depending on the maximum disparity range of the system (e.g., a 64-level disparity system requires 64 times a raw

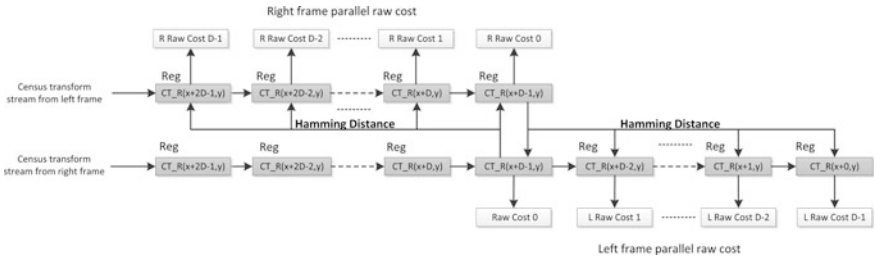

Fig. 3.17 Example of parallel hamming distance hardware architecture

matching cost measurement for each pixel). As shown in Fig. 3.17 the raw matching cost result is stored in an additional, independent dimension with a size equal to the maximum number of disparity levels.

The raw matching cost indicates the similarity of the reference pixel to the target pixels at various disparity positions. Therefore, it should be easy to get the most similar pixel by taking the disparity with the minimum raw matching cost. However, in most practical situations, there is a need to improve the extracted disparity candidates, by using global matching algorithms such as DP, as explained in next section.

3.3.1.3 Support Region Builder

The cross-based support region algorithm [12] is chosen to extract the support region information. Figure 3.18 shows an example design of such cross-based support region builder. It contains multiple line buffers, a shift register array, and support arm region encoders. Since the cross-based support function requires 2D pixel luminance information, we use multiple line buffers to keep the input pixel luminance stream on-chip. Fortunately, this memory architecture for support region builder is overlapped with the one for the CT function. Hence, it is designed to share data with the preprocessor. The encoded cross-arm information is stored in a buffer and will also be used in the postprocessor for disparity voting.

3.3.2 DP Hardware Architecture

To reduce the hardware resource consumption of DP processor, we propose a solution by taking the advantage of the Potts model smoothness function. The Potts model approach takes an $O(D \cdot W)$ computational complexity in the energy function, whereas linear approach is of order $O(D^2 \cdot W)$,where D is the maximum disparity range and W is the number of pixels in the image scanline. Furthermore, the mentioned forward pass Eq. (3.7) can be rewritten in Eq. (3.12), which requires less adder components when applying maximum parallelism in the VLSI design.

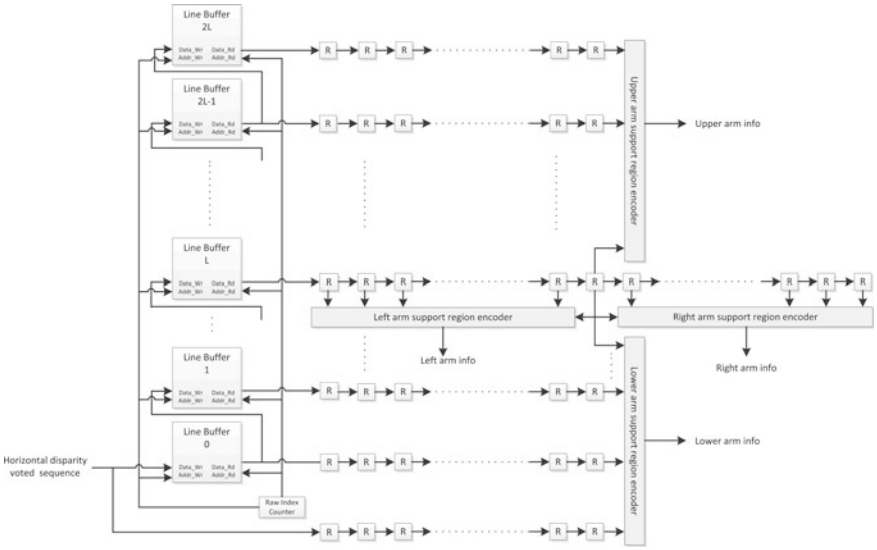

Fig. 3.18 Example hardware design of the cross-based support region buildersupport region builder

$$C_{\min\ \mathrm{Assum}} = \min_{d_p \in w}\left\{c_w(j-1,d_p) + c\right\}$$

$$C_w(j,d_q) = c(j,d_q) + \min\left\{c_{\min\ \mathrm{Assum}}, c(j-1,d_p)\right\} \tag{3.12}$$

One problem of DP is that it requires tremendous memory space to store the aggregation cost information $C_w(j,d_q)$ for the backward pass process. Therefore, we take the advantage of the Potts model again by only storing the decision of the backward pass. Indeed, the Potts model has only two possible backward path decisions: either jump to the disparity value with minimum cost or remain on the same disparity value. Therefore, the backward path information can be represented in Eq. (3.13) instead of storing a complete aggregation cost array, requiring less memory utilization.

$$\mathrm{Backward\_path}(j,d) = \begin{cases} 1, & C_{\min\ \mathrm{Assum}} \le C(j-1,d_p) \\ 0, & C_{\min\ \mathrm{Assum}} > C(j-1,d_p) \end{cases}$$

$$\mathrm{Backward\_Min}C\mathrm{\_path}(j) = \arg\min\left\{C_{\min\ \mathrm{Assum}}, C(j-1,d_p)\right\} \tag{3.13}$$

where $j \in W$, W is the number of image scanline pixels, and $d_p \in$max disparity range. In the backward pass function, Eq. (3.8) can be rewritten into Eq. (3.14). When the path decision is 0, the backward entry remains the same disparity. When the path decision is 1, the backward entry points to the path which possesses the minimum aggregation cost assumption.

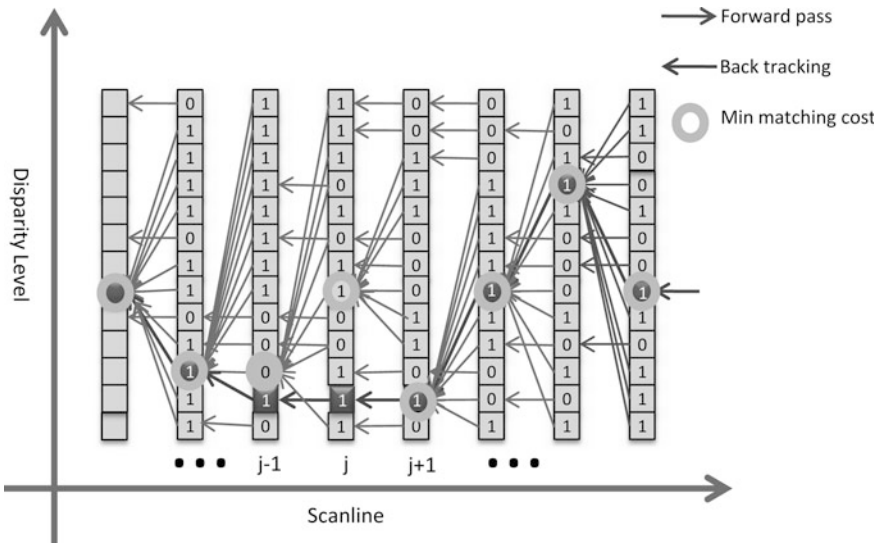

Fig. 3.19 Example of forward DP operations

$$d(j-1) = \begin{cases} \text{Backward}_{\min C_{\text{path}}(j)}, & \text{if Backward}_{\text{Path}(j,d)} = 1 \\ d(j), & \text{if Backward}_{\text{Path}(j,d)} = 0 \end{cases} \tag{3.14}$$

where $j \in W$, W represents the number of image scanline pixels.

Figure 3.19 is an example of DP that shows how the backward path information is first explored in the forward pass step, and the path decision is then stored in 1 bit. Then the backward pass step traverses back through the path information to obtain the optimal disparity result in the image scanline.

Based on Eqs. (3.13) and (3.14), Fig. 3.20 illustrates an example hardware architecture design of the forward pass that is proposed for the stream processor on VLSI. This architecture computes the input matching cost array in maximum parallelism in order to achieve 1:1 throughput, i.e., for each new pair of input pixels, create one output pixel for the depth map. Then the calculated backward path information is stored in buffer for the backward pass.

When the forward pass loop is processing new incoming matching costs for the current image scanline, the backward pass loop reads out the backward path array information from the buffer and traces back the backward path along the image scanline concurrently. Since the disparity output result from the backward pass function shows the inverse sequence order from the conventional input sequence order, a sequence reorder circuit is needed. The scanline reorder circuit can be simply achieved by a ping-pong buffer and some address counters.

Figure 3.21 shows a pair of disparity maps that are calculated from the image pair Tsukuba [3]. The disparity maps are captured from the output of the DP function. Comparing to the original stereo image sources, the disparity maps suffer

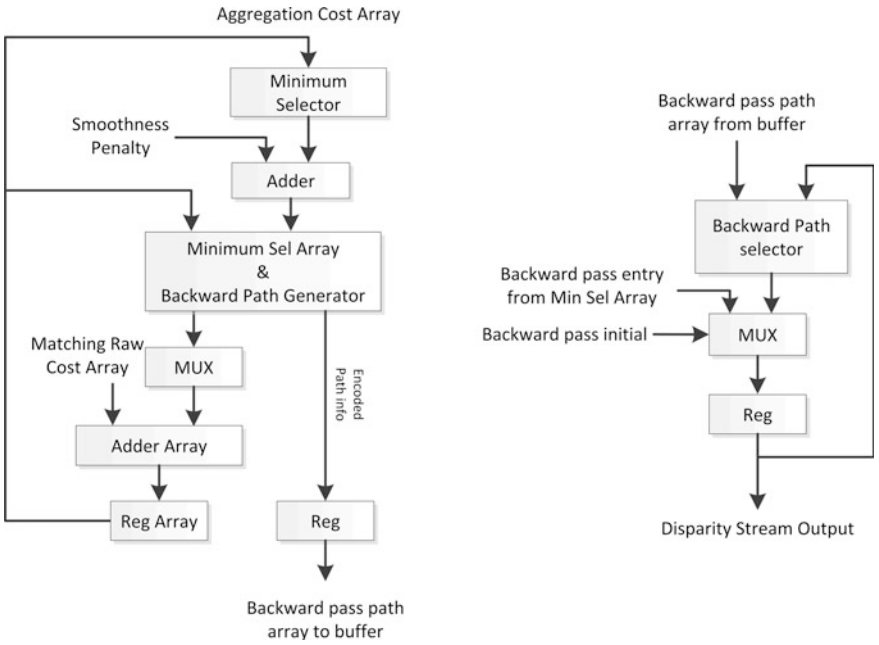

Fig. 3.20 Example design of forward- (*left*) and backward-pass (*right*) functions

from streak effects and speckle noise. They also show some mismatching problems. In the next subsection, the postprocessor is described to circumvent these problems with a disparity map refinement approach.

3.3.3 Postprocessor Hardware Architecture

3.3.3.1 Cross-Check

Due to the occlusion and mismatching disparity problems that affect the quality of the disparity map, we apply a cross-check function in the stereo matching algorithm to detect the mismatching regions. Figure 3.22 is an example that shows the left and right occlusion maps.

An example RTL design of the consistency check function for the left disparity map consistency check is shown in Fig. 3.23. The length of the line buffers are set to the same as the maximum disparity range, which can also be implemented by registers. The disparity value $D(x, y)$ from the left image is compared to the disparity value $D'(x - D(x, y), y)$ from the right disparity buffer in the consistency check block. The consistency check block compares the disparity difference and a threshold value. If the disparity difference is larger than the threshold, it is replaced

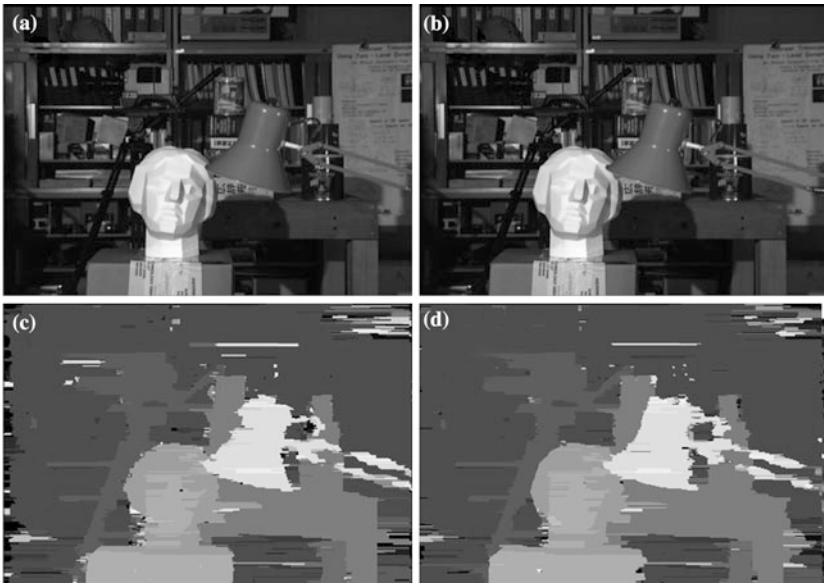

Fig. 3.21 Tsukuba *left* (**a**) and *right* (**b**) stereo image sources, and their resulting *left* (**c**) and *right* (**d**) disparity maps from the DP

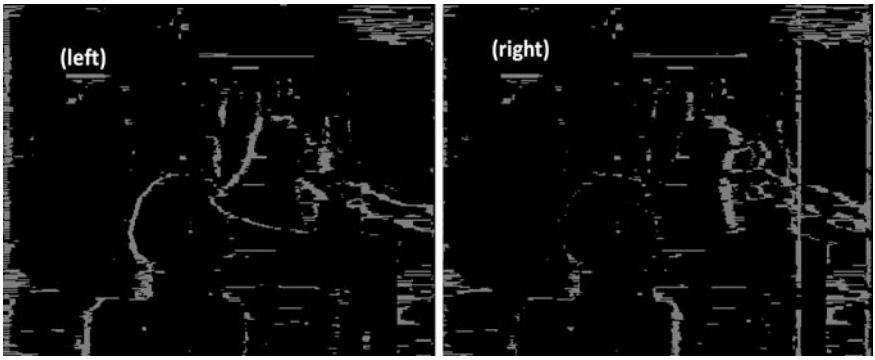

Fig. 3.22 Tsukuba *left* and *right* occlusion maps

by the nearest good disparity value. Another approach is to tag the mismatching pixels to form occlusion maps. Then the information of the occlusion pixels will be used in the disparity voting function. The occlusion pixels are then replaced by the most popular good disparity values within their support region. In the next subsection, the RTL design of the disparity voting will be further explained.

Figure 3.24 shows the disparity map results that are captured from the output of the consistency check function, where the occlusion solving strategy is replacing the mismatching disparity pixel by the nearest good disparity value.

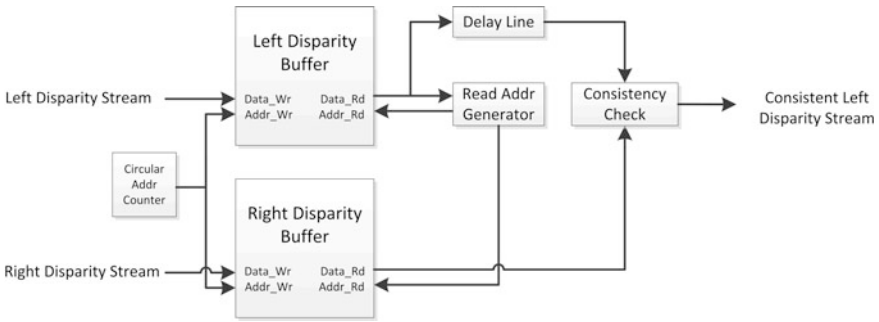

Fig. 3.23 Consistency check RTL example design

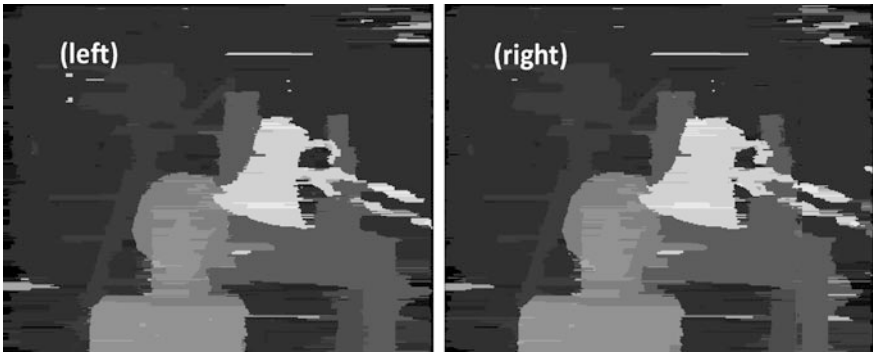

Fig. 3.24 Tsukuba *left* and *right* disparity maps after consistency check

3.3.3.2 Disparity Voting

Disparity voting is used to refine the disparity map. To reduce the computational complexity, the voting process can be modified to a two-pass approach: horizontal voting and vertical voting by using the cross-based support region approach of Sect. 3.3.1 [11]. This makes the histogram calculation easier for hardware implementation. The computational complexity can be reduced from $O(N\text{^}2)$ to $O(2N)$. Figure 3.25 illustrates how the 2D disparity voting procedure can be approximated to two successive1D voting procedures.

The proposed disparity voting RTL design first applies the horizontal voting function, then vertical voting function. Figure 3.26 is an example design which selects the most popular disparity value in the horizontal arm length support region. This architecture is composed of shift registers, a disparity comparator array, a support region mask, a population counter, and a comparator tree. There are $2L + 1$ shift registers, where L represents the maximum support arm length. The disparity comparator array compares the input disparities from the registers

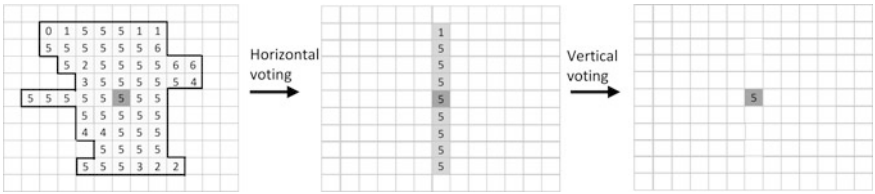

Fig. 3.25 Two pass disparity map voting

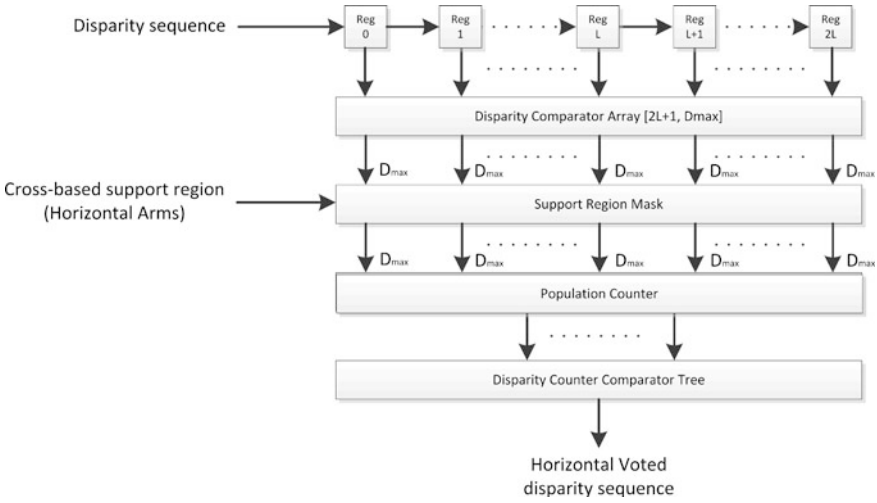

Fig. 3.26 RTL example design of horizontal voting function

and assigns the corresponding disparity bit flag where '1' represents equal and '0' represents not equal. Then the support region mask filters the disparity flags that do not belong to the support region by setting them to '0'. Afterwards, the population counter function accumulates the disparity bit flags into a histogram for the disparity counter comparator tree and selects the most popular disparity value.

Figure 3.27 is an example design of the vertical voting function which selects the most popular disparity value in the vertical support region. In general, the architecture is almost the same as the horizontal voting function, except for the memory architecture. The memory architecture accumulates the incoming disparity scanline stream into line buffers and utilizes data-reuse techniques to extract vertical pixels for disparity voting.

Figure 3.28 demonstrates the disparity map results that are captured from the disparity voting function. Comparing to Fig. 3.24, the disparity mismatch, the occlusion and streaking effects are heavily alleviated.

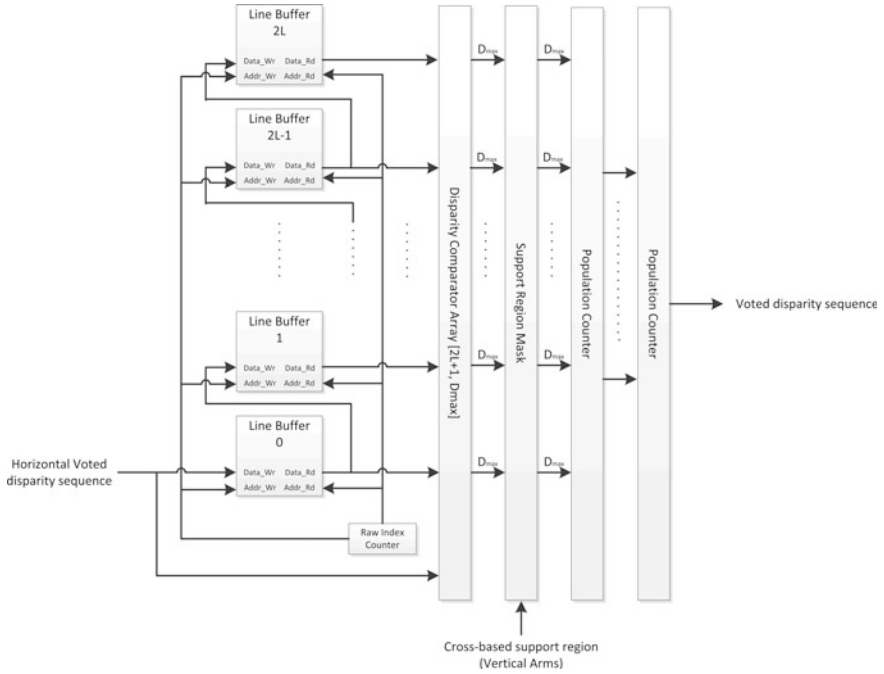

Fig. 3.27 Example design of the vertical voting function

3.3.3.3 Median Filter

In order to reduce the speckle and impulse noises in the disparity map, a median filter is chosen in the final refinement stage. Figure 3.29 is an example design showing the memory architecture of the median filter to keep all scanlines on-chip. Figure 3.30 shows the hardware architecture of the median sorting array that was proposed by Vega-Rodriguez et al. [29]. To avoid critical path constraints, a pipeline technique can be applied to the median sorting array.

After the processing of the median filter, the final disparity map results are shown in Fig. 3.28b.

3.4 Viewpoint Synthesis Kernel

The viewpoint synthesis kernel has the function to generate virtual views in applications like free viewpoint TV and autostereoscopic 3D displays systems [21]. In these systems, the view synthesis engine is a back-end process that synthesizes the virtual view(s) after the depth extraction (e.g. stereo matching) from the multi-view video decoder [30]. In the associated view synthesis algorithms, the

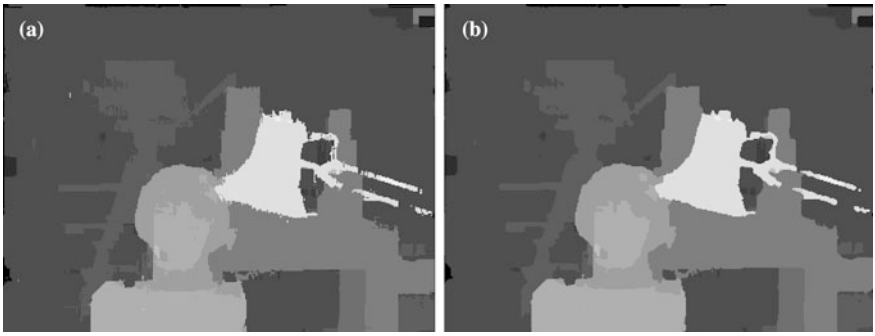

Fig. 3.28 Tsukuba *left* and *right* disparity maps after voting (**a**), and the following median filtering (**b**)

depth-image-based rendering (DIBR) algorithm is now a common approach [31]. It warps an image or video to another view according to the given depth map. Various ways have been proposed to improve the synthesis quality in previous DIBR research [32–34]. Erhan et al. first preprocess the depth map with a motion-adaptive median filter. Lin et al. propose the hybrid inpainting method to fill out holes on disocclusion regions. Tong et al. [34] perform view synthesis in the sub-pixel or quarter-pixel space to improve the subjective quality. With the above improvements, the MPEG consortium developed the reference software for free viewpoint TV, called the view synthesis reference software (VSRS), which synthesizes high-quality virtual views from a limited number of input views.

As mentioned in Sect. 3.1, it is important to achieve real-time performance for HD and full-HD broadcasting, which represents a challenge in hardware design. In the state of the art, there are many researchers working on a high-frame rate implementation following two main approaches. Horng et al. develop the view synthesis engine, which supports HD1080p real time for a single view [35]. Tsung et al. designed a set-top box system on chip for FTV and 3D-TV system, and support the real-time 4K-by-2K process for 9 views [36]. These proposed algorithms consider the input videos with camera rotation; therefore, they have to deal with the complex matrix and fractional computations to align the respective camera views, resulting in many random memory accesses in the calculation process.

In order to overcome these problems and to obtain a cost-efficiency solution, we propose to achieve the synthesis by using the input video on a raster scanline basis as in the aforementioned stereo-matching method and to exclude the effect of camera rotation (the camera rotation is preprocessed in advance by a camera calibration/registration technique). Both input and output turns into raster-scan order. The view synthesis engine design based on this order significantly reduces the hardware cost on buffers, and no reordering buffer is needed.

Figure 3.31 shows the algorithm flowchart of our hardware architecture. This view synthesis algorithm is a simplified version of the VSRS algorithm, and it is assumed that the input videos have been rectified and normalized, and no camera rotation is further required. This algorithm consists of three sequential stages:

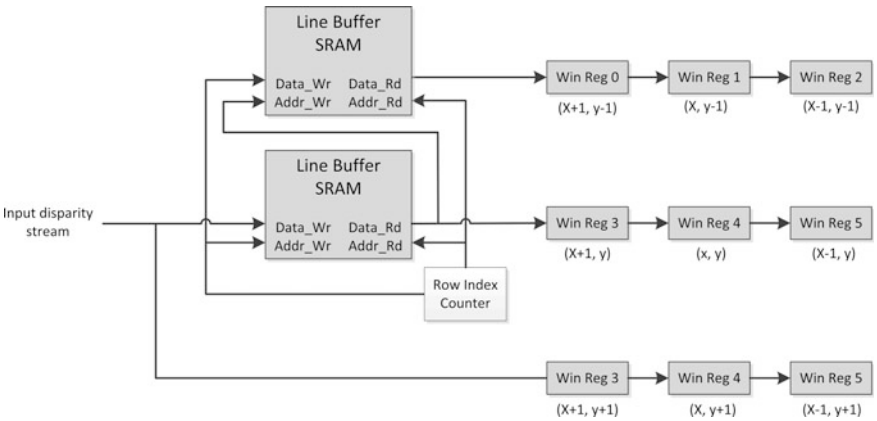

Fig. 3.29 Memory architecture example design of the median filter

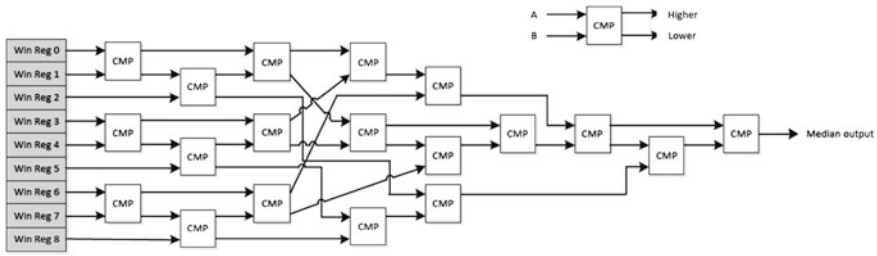

Fig. 3.30 Median filter sorting architecture [29]

- Forward warping stage.
- Reverse warping stage.
- Blending and hole filling stage.

In the forward warping, the depth maps of the left view *DL* and right view *DR* are loaded and warped forward to the depth maps of the virtual to be synthesized view *VDL* and *VDR* separately. The warping location of each pixel is determined by its disparity. When more than two pixels are warped to the same location, the pixel with the greatest disparity value (which is assumed to be in the foreground scene) will be chosen, revealing foreground object pixels with greater priority. After the warping, there are some blanking holes to which none of the image pixels have been warped. The blanking holes are the result of two causes: the occluded region of the reference image, and the downgraded precision from the integer truncation of warping locations, as shown in Fig. 3.31. To reduce the impact of the error to the next stage, a general median filter is used to fill these small blanking holes with selected neighboring data, completing the virtual view depth maps (*VDL* and *VDR*). (Fig. 3.32)

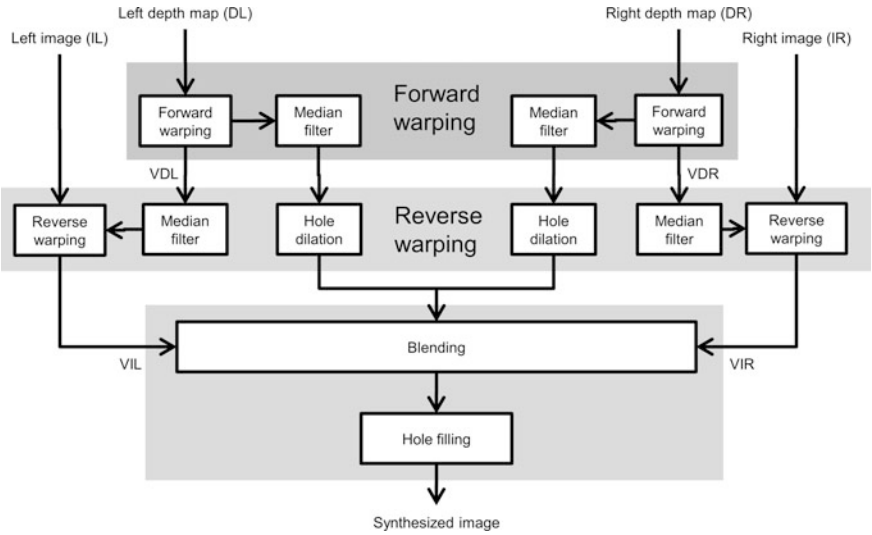

Fig. 3.31 Algorithm flowchart of virtual view synthesis. It mainly consists of the forward warping, reverse warping and a blending process

Given the warped *VDL* and *VDR* from the forward warping stage, the reverse warping fetches the texture data from the original input left and right images (*IL* and *IR*) to produce the two virtual view frames (*VIL* and *VIR*). This stage warps in a similar way as in the forward warping stage. However, instead of using the depth map as a reference image, the reverse warping stage uses the left/right image as reference and warps these images based on the virtual view depth maps. After this stage, some of the occlusion region again causes some blank regions, which are dealt with in the next stage. Besides the warping, the hole-dilation, on the other hand, expands the hole region by one pixel, so that the dilated result can be used in the next stage for improving synthesis quality.

The blending process, which is considered to be the final stage of the proposed synthesizes process, merges the left/right virtual images (*VIL* and *VIR*) into one to create the synthesized result. Because the synthesized result has most of its pixels that can be seen in the left/right virtual images, the blending process is set as an average process of the texture values from *VL* and *VR*. This improves the color consistency between the two input views (*L* and *R*) with higher quality. For the occlusion regions, because most of the occlusion regions could be seen by either one of the two views, *VL* and *VR* are then used to complement each other, and to generate a synthesized view. In addition, the preceding hole-dilation step has more background information copied from the complementary view.

After the blending process, there are still some holes that cannot be found from both input views. This problem can be solved by inpainting methods. Here, a simplified method proposed by Horng [35] is employed to do the inpainting. Unlike other methods that always suffer from their computational complexity and time-consuming iterative process, the proposed hole filling process adopts a 5-by-9 weighted array filter as a bilinear interpolation to fill the hole region with less computational complexity.

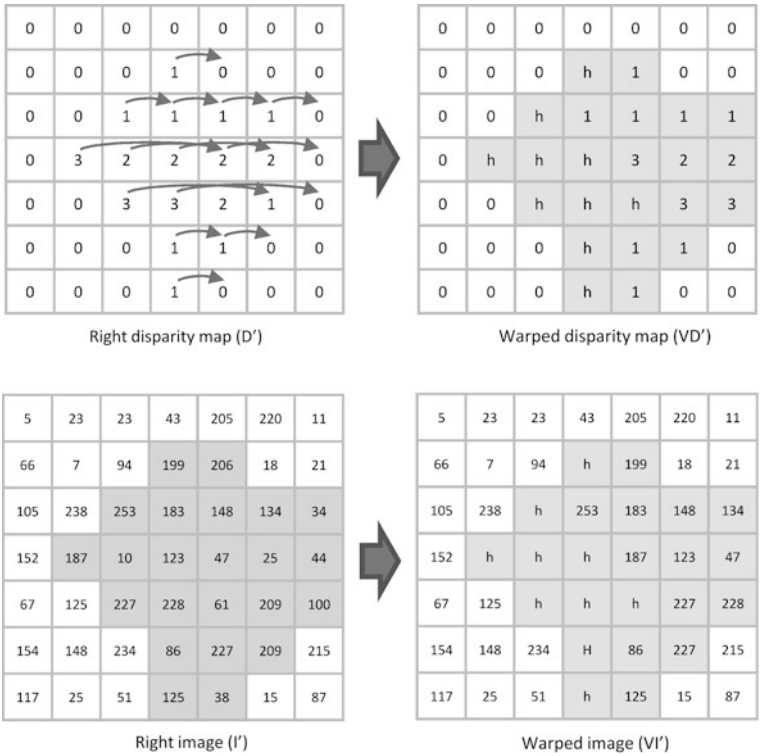

Fig. 3.32 Forward warping (*top*) and reverse warping (*bottom*). In the forward warping, depth maps are warped based on the disparity value, and holes are generated after this value has been warped to a new location. In the reverse warping, a similar process is performed on the image in a pixel-by-pixel process based on the warped disparity map. Note that the warping direction is opposite to the left image

The proposed hardware architecture is shown in Fig. 3.33. Our view synthesis engine adopts a scanline-based pipelined architecture. Because the input frames of the texture images and depth maps are both rectified in the preceding stereo-matching method, the process is considered to be a one-by-one scanline processing. During the viewpoint synthesis, all rows in a specific input frame are independently processed in creating the final output results. Under these considerations, we synthesize the virtual view row by row.

Because of the scanline-based architecture, the input and output of our view synthesis engine may be able to receive and deliver a sequential line-by-line data stream. For example, for 1024 × 768 video frames, the input stream has a length of 768 lines of 1024 data packets. Under this scanline-based architecture, the forward warped depth information can be stored in internal memory, so that the bandwidth of the warped depth data access from the external memory is greatly reduced. For the row-based architecture, the size of the internal memory buffers

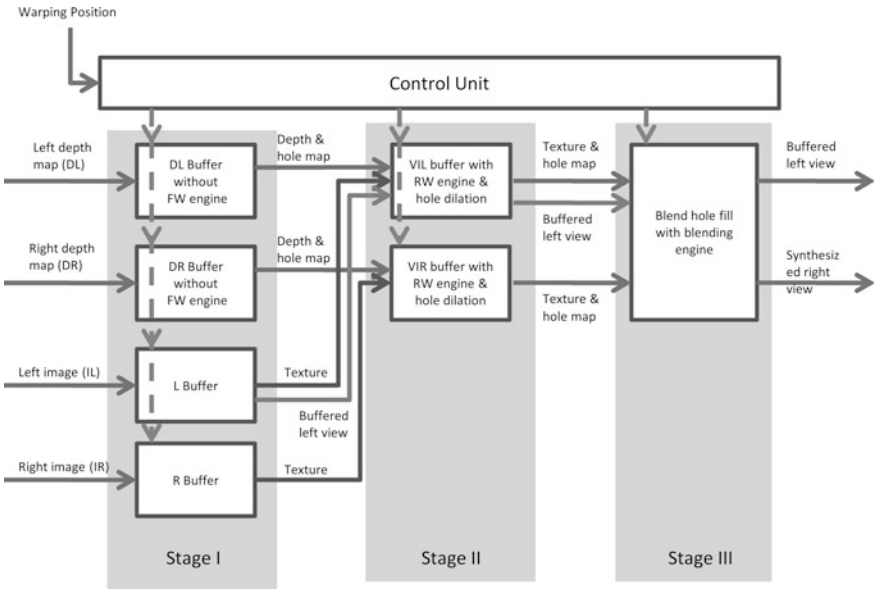

Fig. 3.33 Block diagram of the viewpoint synthesis hardware implementation

depends on the frame width for storing a frame row. In our design, we adopted the SRAM with 1920 $\times$ 8 bits memory space to support up to HD1080p video.

In the view synthesis engine, the data flow is described according to Fig. 3.31. In "stage I", the *DL* and *DR* are warped toward the requested viewpoint. This process is done in scanline order from the depth map and two scanline buffers (*L*/*R* buffer) for left and right images, respectively. Holes in the warped depth images are filled in the next two stages: stage II handles small 1-pixel width cracks with hole dilation, and stage III handles more extensive holes before outputting the *L*/*R* synthesized views.

Based on this structure, the whole viewpoint synthesis process can be segmented into three sequential and independent stages. Since there is no recursive data fed back from the last stage, the whole process can be pipelined. For example, while "stage II" handles scanline number "i", scanline number "$i - 1$" is in "stage I" and "$i + 1$" is in "stage III". Consequently, three rows are independently active in the engine, and the process latency is greatly reduced by efficiently operating every stage in the engine within a predetermined execution time.

3.5 System Quality Performance

In order to evaluate the performance of the proposed system, the Middlebury stereo benchmark was adopted to quantitatively estimate the depth map quality [3, 37]. In its database, four sets of stereo images, i.e., Tsukuba, Venus, Teddy, and Cones, are provided as a reference for observing various shapes and textures that

Fig. 3.34 Disparity results of the proposed implement for the Tsukuba, Venus, Teddy, and Cones stereo images (from *left* to *right*)

would possibly impact the performance of the tested algorithm. In the following, we describe the quality estimation of the stereo matching and viewpoint synthesis, respectively.

3.5.1 Stereo Matching

To evaluate the system performance on depth quality, the four data sets of the stereo images were processed to create the depth map. The results are shown in Fig. 3.34. The quality of resulting depth maps were quantitatively measured based on the ground truth of the datasets by the Middlebury website and listed in Table 3.2. Four result sets are listed in three values, including: 1. "nonocc" value (estimated with only the result from only non-occluded area of the image), 2. "all" value (shows the result for the whole image), and 3. "disc" value (presents the error from only the depth discontinuities on the image). All of these values are in a unit of percentage, and smaller values correspond to a higher quality, i.e,. depth map closer to the ground truth. As listed in the table, most values from the proposed system are below 5 in non-occlusion areas, which are considered as very high-quality indicators.

3.5.2 View Synthesis

Performance of the viewpoint synthesis was evaluated by using five image sets from Middlebury, including *Venus*, *Teddy*, *Sawtooth*, *Poster*, and *Cones* [37]. Each set has five images numbered from 2 to 6 (*IM*2–*IM*6), and additional two ground truth images from image 2 and 6 (*GT*2, and *GT*6). Images 2 and 6 were used to

Table 3.2 Quantitative results of the stereo matching for the original Middlebury stereo database

Algorithm	Tsukuba			Venus			Teddy			Cones			Average percent bad pixels
	Nonocc	All	Disc	Nonocc	All	Disc	Nonocc	All	Disc	Nonocc	All	Disc	
Proposed	2.54	3.09	11.1	0.19	0.42	2.36	6.74	12.4	17.1	4.42	10.2	11.5	6.84
RT-ColorAW [38]	1.4	3.08	5.81	0.72	1.71	3.8	6.69	14	15.3	4.03	11.9	10.2	6.55
SeqTreeDP [39]	2.21	2.76	10.3	0.46	0.6	2.44	9.58	15.2	18.4	3.23	7.86	8.83	6.82
MultiCue [40]	1.2	1.81	6.31	0.43	0.69	3.36	7.09	14	17.2	5.42	12.6	12.5	6.89
AdaptWeight [41]	1.38	1.85	6.9	0.71	1.19	6.13	7.88	13.3	18.6	3.97	9.79	8.26	6.67
InteriorPtLP [42]	1.27	1.62	6.82	1.15	1.67	12.7	8.07	11.9	18.7	3.92	9.68	9.62	7.26

nonocc: non-occluded regions

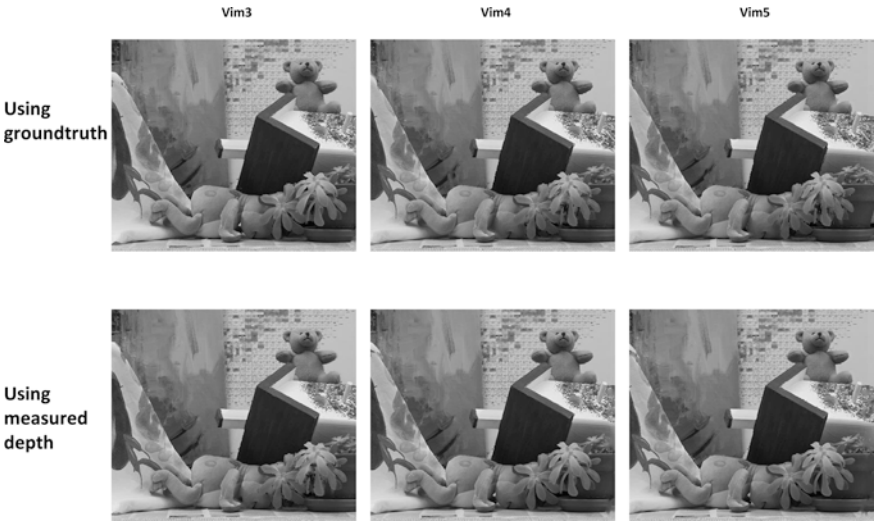

Fig. 3.35 The synthesized virtual images from various viewing angles by using the ground truth (*top row*) and the measured depth map (*bottom row*)

Table 3.3 PSNR of the synthesized virtual images by using the depth map from the ground truth and the proposed stereo-matching method. Note that each number is an average value from *IM*3 to *IM*5

PSNR (dB)	Venus	Teddy	Sawtooth	Poster	Cones
GroundT	45.07	44.76	44.77	44.24	43.12
StereoM	45.09	44.53	44.57	44.07	42.92

create the left/right depth maps (i.e., *DL* and *DR*). The calculated depth map and the ground truth were then used to synthesize the virtual views at exactly the viewpoints *IM*3, *IM*4, and *IM*5, as shown in Fig. 3.35. Images on the top row show the virtual view synthesized by using ground truth (i.e., *VIM*3, *VIM*4, and *VIM*5) and those on the bottom row show the result by using the resulting depth map from stereo matching. As a comparison, PSNRs were calculated from the virtual views and its corresponding images, as listed in Table 3.3. It is clear that the PSNR of the virtual images synthesized from the ground truth and the calculated depth are in a value around 43–45 and 42–45 dB, respectively.

3.6 System Speed Performance

In this section, we evaluate the system performances with MDE/s (million disparity estimations per second) for the stereo matching and FPS (frames per second) for the viewpoint synthesis. The performance of GPU-based stereo matching has

Table 3.4 The predefined engine spec of the proposed system in two cases

	HD	FullHD
Image width	1024	960
Image height	768	1080
Frame rate (FPS)	60	60
Maximum disparity level	128	128
Speed performance (MDE/s)	6039	7962

Table 3.5 Estimated system resources and hardware costs based on TSMC 250 nm process by using Cadence RTL compiler

	Process	Gate count (k gates)	Memory size (k bytes)
Stereo matching	CTSR	65.6	62
	Rawcost	88.2	0
	DP	79.3	43
	Post	98	54
Viewpoint synthesis	Forward_warp	41.8	4
	Reversed_warp	38.2	1
	Hole_filling	58.6	2
Total		469.7	159
Image size		1920 × 1080	
Maximum disparity		64 levels	

been reported to be 2796.2 MDE/s (HD with 27.8 FPS and disparity range of 128) [26]. In addition, Jin et al. report a FPGA stereo matching system with a speed performance of 4522 MDE/s (VGA with 230 FPS and disparity range of 64) [27]. Since most of the algorithms described in previous section are focused on both the quality of the depth map and the implementation on hardware the question "How fast is the system?" is very relevant. Here, we synthesized the engine in the Altera Quartus II design framework and implemented it on to the stratix III FPGA with the spec listed in Table 3.4.

In Table 3.4, two cases have been implemented for HD and FullHD (i.e., 1080p) video format. In the HD case, the input image resolution is set to be 1024 by 768 and with a frame rate of 60 FPS. The maximum disparity of these two cases is defined as 128 levels to cover a large dynamic depth range in the video. Note that the configuration in the FullHD case is a preliminary test for the FullHD video steam and was configured for "half side-by-side" format. In this scenario, a speed performance of 7,962 MDE/s is achieved.

Furthermore, we also studied on how this algorithm performed on an ASIC (application specific integrated circuit). It includes an estimation of the complexity, internal memory resource usage, and the possible execution speed. An ASIC platform synthesis and simulation have been performed to get estimated figures of merit. We use the TSMC 250 nm process library with cadence RTL compilation at medium area optimization for logic synthesis as reference forwarding to ASIC implementation.

Table 3.5 lists the estimated gate count and memory size. The gate count is below 500 kgates, which is very hardware-friendly for ASIC implementation in a low-cost technology node. For FullHD 1080p, the gate count of the system remains unchanged. The disparity level is predefined to be 64, which satisfies the depth budget of the FullHD (∼3 %) for high-quality 3D content [43].

3.7 Conclusion

In this chapter, we have demonstrated the application of a user-defined depth scaling in 3D-TV, where a new left/right stereo image pair is calculated and rendered on the 3D display, providing a reduced depth impression and better viewing experience with less eye strain. Calculating more than two output views by replicating some digital signal processing kernels supports the "Stereo-Into Multiply-Viewpoint-Out" functionality for 3D auto-stereoscopic displays. Depth is extracted from the pair of left/right input images by stereo matching, and multiple new viewpoints are synthesized and rendered to provide the proper depth impression in all viewing directions. However, there is always a tradeoff between system performance and cost. For instance, an accurate cost calculation needs larger aggregate size (Sect. 3.2.1), which leads to a higher gate count and memory budget in the system and degrades the system performance. Consequently, building a high-quality, real-time system remains a challenge. We proposed a way to achieve this target. For every step in the proposed stereo matching system, we employ efficient algorithms to provide high-quality results at acceptable implementation cost, demonstrated on FPGA. We have demonstrated the efficiency and quality of the proposed system both for the stereo-matching and viewpoint synthesis, achieving more than 6,000 million disparity estimations per second at HD resolution and 60 Hz frame rate with a BPER of no more than 7 % and a PSNR of around 44 dB on MPEG's and Middlebury's test sequences.

References

1. Hartley R, Zisserman A (2004) Multiple view geometry in computer vision. Cambridge University Press, Cambridge
2. Papadimitriou DV, Dennis TJ (1996) Epipolar line estimation and rectification for stereo image pairs. IEEE Trans Image Process 5(4):672–676
3. Scharstein D, Szeliski R, Zabih R (2001) A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In: Proceedings IEEE workshop on stereo and multi-baseline vision (SMBV 2001)
4. Brown MZ, Burschka D, Hager GD (2003) Advances in computational stereo. IEEE Trans Pattern Anal Mach Intell 25(8):993–1008
5. Porter RB, Bergmann NW (1997) A generic implementation framework for FPGA based stereo matching. In: IEEE region 10 annual conference. Speech and image technologies for computing and telecommunications (TENCON '97), Proceedings of IEEE

6. Lane RA, Thacker NA (1998) Tutorial: overview of stereo matching research. Available from: http://www.tina-vision.net/docs/memos/1994-001.pdf
7. Zhang K et al (2009) Robust stereo matching with fast normalized cross-correlation over shape-adaptive regions. In: 16th IEEE international conference on image processing (ICIP)
8. Chang NYC, Tseng Y-C, Chang TS (2008) Analysis of color space and similarity measure impact on stereo block matching. In: IEEE Asia Pacific conference on circuits and systems (APCCAS)
9. Hirschmuller H, Scharstein D (2009) Evaluation of stereo matching costs on images with radiometric differences. IEEE Trans Pattern Anal Mach Intell 31(9):1582–1599
10. Humenberger M, Engelke T, Kubinger W (2010) A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality. In: IEEE computer society conference on computer vision and pattern recognition workshops (CVPRW)
11. Zhang K et al (2011) Real-time and accurate stereo: a scalable approach with bitwise fast voting on CUDA. IEEE Trans Circuits Syst Video Technol 21(7):867–878
12. Zhang K, Lu J, Gauthier L (2009) Cross-based local stereo matching using orthogonal integral images. IEEE Trans Circuits Syst Video Technol 19(7):1073–1079
13. Yoon K-J, Kweon I-S (2005) Locally adaptive support-weight approach for visual correspondence search. In: IEEE computer society conference on computer vision and pattern recognition (CVPR)
14. Hosni A et al (2009) Local stereo matching using geodesic support weights. In: 16th IEEE international conference on image processing (ICIP)
15. Wang L et al (2006) High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In: Third international symposium on 3D data processing, visualization, and transmission
16. Chang NYC et al (2010) Algorithm and architecture of disparity estimation with mini-census adaptive support weight. IEEE Trans Circuits Syst Video Technol 20(6):792–805
17. Zhang L et al (2011) Real-time high-definition stereo matching on FPGA. In: Proceedings of the 19th ACM/SIGDA international symposium on field programmable gate arrays (FPGA)
18. Zhang L (2010) Design and implementation of real-time high-definition stereo matching SoC on FPGA. In: Department of microelectronics and computer engineering, Delft university of technology, The Netherlands
19. Yi GY (2011) High-quality, real-time HD video stereo matching on FPGA. In: Department of microelectronics and computer engineering, Delft university of technology: The Netherlands
20. Hirschmuller H (2005) Accurate and efficient stereo processing by semi-global matching and mutual information. In: IEEE computer society conference on computer vision and pattern recognition (CVPR)
21. Tanimoto M (2004) Free viewpoint television—FTV. In: Proceedings of picture coding symposium
22. Boykov Y, Veksler O, Zabih R (2001) Fast approximate energy minimization via graph cuts. IEEE Trans Pattern Anal Mach Intell 23(11):1222–1239
23. Sun J, Zheng N–N, Shum H-Y (2003) Stereo matching using belief propagation. IEEE Trans Pattern Anal Mach Intell 25(7):787–800
24. Berdnikov Y, Vatolin D (2011) Real-time depth map occlusion filling and scene background restoration for projected pattern-based depth cameras. Available from: http://gc2011.graphicon.ru/files/gc2011/proceedings/conference/gc2011berdnikov.pdf
25. Wang L et al (2008) Stereoscopic inpainting: joint color and depth completion from stereo images. In: IEEE conference on computer vision and pattern recognition (CVPR)
26. Banz C et al (2010) Real-time stereo vision system using semi-global matching disparity estimation: architecture and FPGA-implementation. In: International conference on embedded computer systems (SAMOS)
27. Seunghun J et al (2010) FPGA design and implementation of a real-time stereo vision system. Circuits and Systems for Video Technology. IEEE Trans Circuits Syst Video Technol 20(1):15–26
28. Gehrig SK, Meyer FET (2009) A real-time low-power stereo vision engine using semi-global matching. Lect Notes Comput Sci 5815:134–143

29. Vega-Rodríguez MA, Sánchez-Pérez JM, Gómez-Pulido JA (2002) An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems. In: Proceedings of the 10th mediterranean conference on control and automation. Lisbon
30. Aljoscha S (2008) Introduction to multiview video coding. Antalya, Turkey: ISO/IEC JTC 1/SC 29/WG 11
31. Fehn C (2004) Depth-image-based rendering (DIBR), compression and transmission for a new approach on 3D-TV. In: Proceedings of SPIE conference on stereoscopic displays and virtual reality system
32. Ekmekcioglu E, Velisavljevic V, Worrall ST (2009) Edge and motion-adaptive median filtering for multi-view depth map enhancement. In: Picture coding symposium (PCS)
33. Tsung P-K et al (2011) A 216fps 4096 × 2160p 3DTV set-top box SoC for free-viewpoint 3DTV applications. In: IEEE international solid-state circuits conference (ISSCC). Digest of technical papers
34. Tong X et al (2010) A sub-pixel virtual view synthesis method for multiple view synthesis. In: Picture coding symposium (PCS)
35. Horng Y, Tseng Y, Chang T (2011) VLSI architecture for real time HD1080p view synthesis engine. IEEE Trans Circuits Syst Video Technol 21(9):1329–1340
36. Tsung P-K et al. (2011) A 216fps 4096 × 2160p 3DTV set-top box SoC for free-viewpoint 3DTV applications. In: IEEE international solid-state circuits conference (ISSCC), San Francisco
37. Scharstein D, Szellski R (2011) Middlebury stereo vision page. Available from: http://vision.middlebury.edu/stereo/
38. Chang X et al (2011) Real-time accurate stereo matching using modified two-pass aggregation and winner-take-all guided dynamic programming. In: International conference on 3D imaging, modeling, processing, visualization and transmission (3DIMPVT)
39. Deng Y, Lin X (2006) A fast line segment based dense stereo algorithm using tree dynamic programming. In: Proceedings of ECCV
40. Liu T, Luo L (2009) Robust context-based and separable low complexity local stereo matching using multiple cues. Submitted to TIP
41. Yoon K-J, Kweon IS (2006) Adaptive support-weight approach for correspondence search. IEEE Trans Pattern Anal Mach Intell 28(4):650–656
42. Bhusnurmath A, Taylor CJ (2008) Solving stereo matching problems using interior point methods. In: Fourth international symposium on 3D data processing, visualization and transmission (3DPVT)
43. Knorr S et al (2011) Basic rules for "good 3d" and the avoidance of visual discomfort in stereoscopic vision. IBC, Amsterdam