2013

# Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences

Jedrzej Kowalczuk
*University of Nebraska-Lincoln*, jkowalczuk2@unl.edu

Eric T. Psota
*University of Nebraska-Lincoln*, epsota@unl.edu

Lance C. Pérez
*University of Nebraska-Lincoln*, lperez@unl.edu

Pages: 94 - 104, DOI: 10.1109/TCSVT.2012.2203200

# Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences

Jędrzej Kowalczuk, *Student Member, IEEE,* Eric T. Psota, *Member, IEEE,* and Lance C. Pérez, *Senior Member, IEEE*

***Abstract*—High-quality real-time stereo matching has the potential to enable various computer vision applications including semi-automated robotic surgery, teleimmersion, and 3-D video surveillance. A novel real-time stereo matching method is presented that uses a two-pass approximation of adaptive support-weight aggregation, and a low-complexity iterative disparity refinement technique. Through an evaluation of computationally efficient approaches to adaptive support-weight cost aggregation, it is shown that the two-pass method produces an accurate approximation of the support weights while greatly reducing the complexity of aggregation. The refinement technique, constructed using a probabilistic framework, incorporates an additive term into matching cost minimization and facilitates iterative processing to improve the accuracy of the disparity map. This method has been implemented on massively parallel high-performance graphics hardware using the Compute Unified Device Architecture computing engine. Results show that the proposed method is the most accurate among all of the real-time stereo matching methods listed on the Middlebury stereo benchmark.**

***Index Terms*—Adaptive support weights, CUDA, iterative refinement, real-time stereo matching.**

## I. Introduction

THE STEREO matching problem is formulated as the process of computing a disparity map given a pair of stereo images. The disparity map associates every pixel with a disparity value, i.e., the positional offset between corresponding points in the stereo images. The stereo matching problem is well known in the field of computer vision, and has led researchers to develop a wide range of effective techniques, including image-segmentation [1], [2], plane-fitting [3], belief propagation [4], and adaptive cost aggregation [5], [6], [7]. These techniques provide excellent still-frame accuracy; however, high computational complexity often prohibits their application in real-time systems.

The emergence of manycore, massively parallel graphics processing units (GPUs) and GPU-accelerated computing engines has enabled the implementation of high-accuracy real-time stereo matching methods. The compute unified device architecture (CUDA), a general-purpose parallel computing architecture and a complementary application programming interface developed by NVIDIA, is one of the most popular high-performance computing engines used by researchers to implement real-time stereo matching methods [8]–[11]. Many of these methods achieve real-time performance by reducing the complexity of window-based matching cost aggregation. In addition, these methods typically avoid image segmentation, which requires a large number of computationally demanding iterations [12], and plane-fitting, which lacks the computational regularity necessary for parallelization.

In this paper, a real-time stereo matching method is introduced that uses computationally efficient window-based cost aggregation and a low-complexity iterative disparity refinement technique implemented on CUDA. The iterative disparity refinement technique is constructed using a probabilistic framework and a series of approximations of the matching cost distributions. In the proposed method, both cost aggregation and iterative disparity refinement make use of the adaptive support-weights first introduced in [5]. First, a two-pass approximation of full window-based matching is used to significantly reduce the complexity of cost aggregation. Then, another set of adaptive support weights is generated for iterative refinement of the disparity map. Disparity refinement operates by computing the expected value for the disparity during the current iteration using nearby pixel disparities from previous iterations. Matching costs are then penalized when disparity candidates deviate from this expected value.

This method has been evaluated using the Middlebury stereo benchmark [13], [14], which reveals it to be the top performing real-time algorithm in terms of overall matching accuracy. Experiments have shown that the number of iterations required for convergence is small, and each iteration adds a negligible amount of complexity to matching cost aggregation. The proposed method has been implemented on CUDA using the NVIDIA GeForce GTX 580 GPU. While many real-time methods focus on reducing the complexity associated with cost aggregation, at the expense of reduced matching accuracy [9], [11], the proposed approach takes full advantage of the GTX 580's computing capabilities to produce a highly accurate stereo matching method that maintains real-time performance.

The remainder of this paper is organized as follows. Section II presents background related to the general stereo matching

Manuscript received September 20, 2011; revised February 7, 2012; accepted April 9, 2012. Date of publication June 6, 2012; date of current version January 9, 2013. This work was supported in part by TATRC2, under Grant W81XWH-08-2-0043. This paper was recommended by Associate Editor T. Fujii.

The authors are with the Department of Electrical Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588 USA (e-mail: jkowalczuk2@unl.edu; epsota2@unl.edu; lperez@unl.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2012.2203200

problem, the adaptive support-weight stereo matching method, and a review of existing real-time methods and their associated cost aggregation techniques. Then, in Section III a probabilistic framework is used to derive the proposed iterative disparity refinement method for stereo matching. The CUDA execution model is discussed in Section IV, followed by a detailed description of the real-time implementation. Finally, the accuracy of the proposed method is evaluated in Section V using the Middlebury stereo benchmark, and Section VI concludes this paper.

## II. Background

Recently, the availability of massively parallel graphics hardware has enabled the development of real-time high-resolution stereo matching. This has inspired researchers to create a variety of competing algorithms over the past decade, the majority of which utilize local aggregation techniques that lend themselves well to parallel implementation. Many of the most accurate real-time stereo matching algorithms use an approximation of the adaptive support-weights (ASW) method introduced by Yoon and Kweon [5]. In the following, essential background and notation is given for the general stereo matching problem, and the adaptive support-weight stereo matching algorithm is described. Then, a summary is presented for the class of real-time algorithms that use an approximation of the ASW method.

### A. Stereo Matching

Stereo matching is the process of identifying correspondences between pixels in a pair of stereo images. It is often assumed that the input images are rectified, i.e., the epipolar lines are collinear and run in parallel with the $x$-axis, thus reducing the search space for matches to corresponding rows of the input images. A common way of representing a match is via disparity which, in the case of rectified images, is a horizontal offset between matching pixels. Given a pixel $p$ at location $(x, y)$ in the reference image, and its match $\bar{p}$ at location $(\bar{x}, y)$ in the target image, the disparity of pixel $p$ is given by

$$d(p, \bar{p}) = x - \bar{x}\,.$$

The result of performing stereo matching on a pair of images is the disparity map, i.e., an image that associates every pixel of the reference image with a disparity value. When searching for correspondences, a limited range of disparities $d_{\min}, \ldots, d_{\max}$ is usually considered. Thus, the search domain for matches of pixel $p$, denoted by $S_p$, is the set of pixels in the corresponding row of the target image with column index $\bar{x} \in \{x + d_{\min}, \ldots, x + d_{\max}\}$.

### B. Adaptive Support-Weight Stereo Matching

Adaptive support weights were introduced in 2005 as a new way of aggregating the cost of window-based stereo matching [5]. This method was shown to improve matching accuracy when compared to window-based aggregation methods that attempt to compute the optimal position, or shape, of the support window [15], [16]. The accuracy of the adaptive support-weights method made it one of the first local methods capable of competing with global algorithms that use graph cuts [17] or belief propagation [18], and it has since been used in many of the most accurate algorithms [19], [20], as listed on the online Middlebury stereo benchmark.

The adaptive support-weight stereo matching algorithm mimics the process of visual grouping in the human vision system through the application of the gestalt principles of perception, among which the principle of proximity and the principle of similarity are particularly important to the problem of stereo correspondence. Under the principle of proximity, scene surfaces are assumed to be locally continuous. Consequently, the likelihood that pixel $p$ belongs to the same surface as pixel $q$ decreases as the geometric distance between $p$ and $q$ increases. On the other hand, the principle of similarity states that typical scene surfaces have locally consistent color. Thus, it is likely that pixel $p$ is on the same surface as pixel $q$ when their color and shade are similar.

In order to make use of the gestalt principles of perception for the purposes of stereo matching, the adaptive support-weight stereo algorithm considers a support window $\Omega_p$ of pixel $p$, that is, a square region centered at pixel $p$, and assigns a support-weight to each pixel $q \in \Omega_p$. Let $\Delta_c(p, q)$ denote the color difference and let $\Delta_g(p, q)$ denote the geometric distance between pixels $p$ and $q$ evaluated using the Euclidean metric. The support-weight, or simply weight, assigned to pixel $q$ in the support window of $p$ is given by

$$w(p, q) = \exp\left(-\frac{\Delta_c(p, q)}{\gamma_c} - \frac{\Delta_g(p, q)}{\gamma_g}\right) \tag{1}$$

where the values of $\gamma_c$ and $\gamma_g$ are chosen empirically.

To identify a match for a particular pixel of interest, adaptive support-weight matching costs are calculated between the pixel of interest and every pixel located in its correspondence search domain. Given pixel $p$ in the reference frame, pixel $\bar{p}$ in the target frame, and their support windows $\Omega_p$ and $\Omega_{\bar{p}}$, respectively, the matching cost is computed as

$$C(p, \bar{p}) = \frac{\sum_{q \in \Omega_p, \bar{q} \in \Omega_{\bar{p}}} w(p, q) w(\bar{p}, \bar{q}) \delta(q, \bar{q})}{\sum_{q \in \Omega_p, \bar{q} \in \Omega_{\bar{p}}} w(p, q) w(\bar{p}, \bar{q})} \tag{2}$$

where $\delta(q, \bar{q})$ is an arbitrary distance measure between pixels $q$ and $\bar{q}$, typically the sum of absolute color differences.

Once the matching costs have been computed, a match for pixel $p$ can be obtained using the winner-takes-all (WTA) decision criteria that selects the candidate pixel characterized by the minimum matching cost. Precisely, the match for pixel $p$, denoted as $m(p)$, is given by

$$m(p) = \operatorname*{argmin}_{\bar{p} \in S_p} C(p, \bar{p}). \tag{3}$$

### C. Related Work

While the use of adaptive support weights enhances the accuracy of stereo matching, its complexity makes it unsuitable for cost aggregation in real-time applications. Thus, in

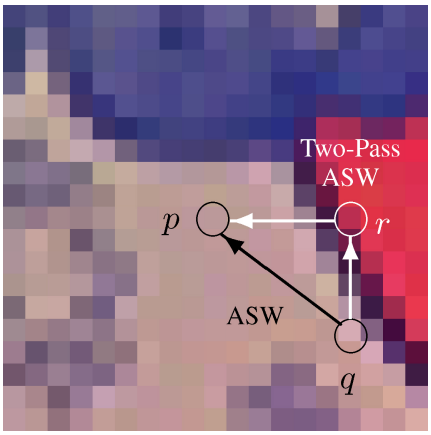


Fig. 1. Support-weights relating the pixel of interest $p$ to its neighbor pixel $q \in \Omega_p$ using regular ASW stereo matching and a two-pass approximation passing through an intermediate pixel $r \in \Omega_p$.

order to achieve real-time performance, it is necessary to reduce the complexity of raw adaptive support-weight cost aggregation. To address this issue, four modifications used for computationally efficient cost aggregation have been proposed: two-pass adaptive support weights [21], approximated joint bilateral filtering [22], exponential step-size adaptive weights [9], and cross-based support weights [11].

Instead of using square windows for matching, the two-pass approach approximates the adaptive support weights by performing cost aggregation along the vertical and then the horizontal direction [21]. During vertical aggregation, pixel-wise costs are added within a 1-D column of pixels centered at the pixel of interest, whereas in the horizontal aggregation, a weighted sum of matching costs along the rows is computed using the adaptive support weights. As a result, the complexity of aggregating the matching cost between two windows of size $\omega \times \omega$ is reduced from $\mathcal{O}(\omega^2)$ to $\mathcal{O}(\omega)$. While the two-pass approximation significantly improves computational efficiency, it fails to accurately approximate the support weights under certain conditions. Fig. 1 illustrates an example where the two-pass algorithm fails to capture the relationship between the pixel of interest $p$ and pixel $q$ in its support window. While ASW compares these two pixels directly to generate a support weight, the two-pass approximation compares the neighboring pixel $q$ to an intermediate pixel $r$ during vertical aggregation and then compares $r$ to $p$ in the horizontal aggregation, thus the normalized weight is approximated by $w(p, q) \approx w(q, r) \times w(r, p)$. The two pixels $p$ and $q$ in Fig. 1 are very similar, both in terms of color and shade, however, the pairs of pixels $(q, r)$ and $(r, p)$ are not similar. As a result, the two-pass approximation wrongly assumes strong dissimilarity between $p$ and $q$.

A modified adaptive support-weight aggregation scheme through approximated joint bilateral filtering was introduced in [23], and its real-time implementation was later presented in [22]. This scheme, also known as fast bilateral stereo (FBS), operates by dividing the support region into blocks of $\omega_b \times \omega_b$ pixels. Instead of assigning each pixel in the block a unique support weight, a single support weight is assigned to the entire block. The spatial component of the support weight is

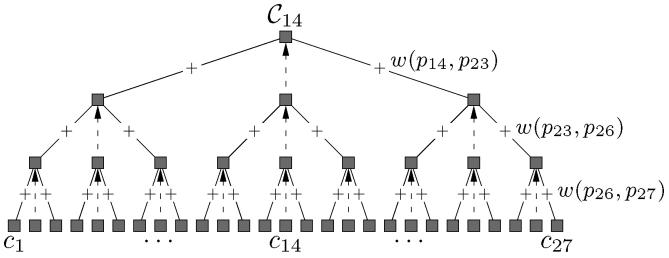


Fig. 2. Pixel-wise costs $(c_1, \ldots, c_{27})$ aggregated recursively in groups of three using ESAW. Each edge operation "+" represents a normalized weighted sum using adaptive support weights.

generated using the distance between the pixel in the center of block and the pixel of interest, and the color component of the support weight is generated using the difference between average block color and the color of the pixel of interest. By computing only one support weight per block, the computational complexity associated with generating support weights is reduced to $\mathcal{O}(\omega^2/\omega_b^2)$. In addition to a reduction in complexity, this aggregation scheme has been shown to provide a higher level of robustness to image noise than the original adaptive support-weight approach.

An alternative method for approximating the adaptive support weights was presented in [9]. While also using a two-pass approximation to square-window adaptive support-weight stereo matching, the aggregation is further accelerated by recursively combining the matching costs over subsets of pixels. This approach, referred to as exponential step-size adaptive weights (ESAW), reduces the complexity of aggregating the matching cost of $\omega$ pixels in both the vertical and horizontal directions from $\mathcal{O}(\omega)$ to $\mathcal{O}(\log(\omega))$. To illustrate the operations of the ESAW method, the example shown in Fig. 2 considers a set of pixel-wise costs $\{c_1, \ldots, c_{27}\}$, that are used to compute the matching cost $\mathcal{C}_{14}$ between pixel $p_{14}$ and some arbitrary pixel in the target image. In this example, the pixel-wise cost $c_{27}$ is aggregated into $\mathcal{C}_{14}$ by performing a weighted sum using the normalized adaptive support weights relating pixel $p_{27}$ to $p_{26}$, $p_{26}$ to $p_{23}$, and finally $p_{23}$ to $p_{14}$. The resulting adaptive support weight that is used to evaluate the similarity between pixels $p_{14}$ and $p_{27}$ is approximated by $w(p_{14}, p_{27}) \approx w(p_{14}, p_{23}) \times w(p_{23}, p_{26}) \times w(p_{26}, p_{27})$. Thus, the similarity of pixels $p_{14}$ and $p_{27}$ estimated using ESAW depends on a chain of three intermediate comparisons. Because these approximations are used within horizontal and vertical aggregation, ESAW is even more susceptible to erroneous support-weight approximation than the two-pass method.

Another approach to computationally efficient stereo matching, known as the cross-based method of support region construction [11], uses binary weights and two-pass aggregation. The method operates by extending cross-shaped windows vertically and horizontally from the pixel of interest. The windows are extended in all four directions until two consecutive pixels are encountered that differ from the pixel of interest by a predefined threshold, or the maximum window length is reached. Although this approach significantly reduces the computational complexity when compared to generating real-valued adaptive support weights, its cost aggregation is limited

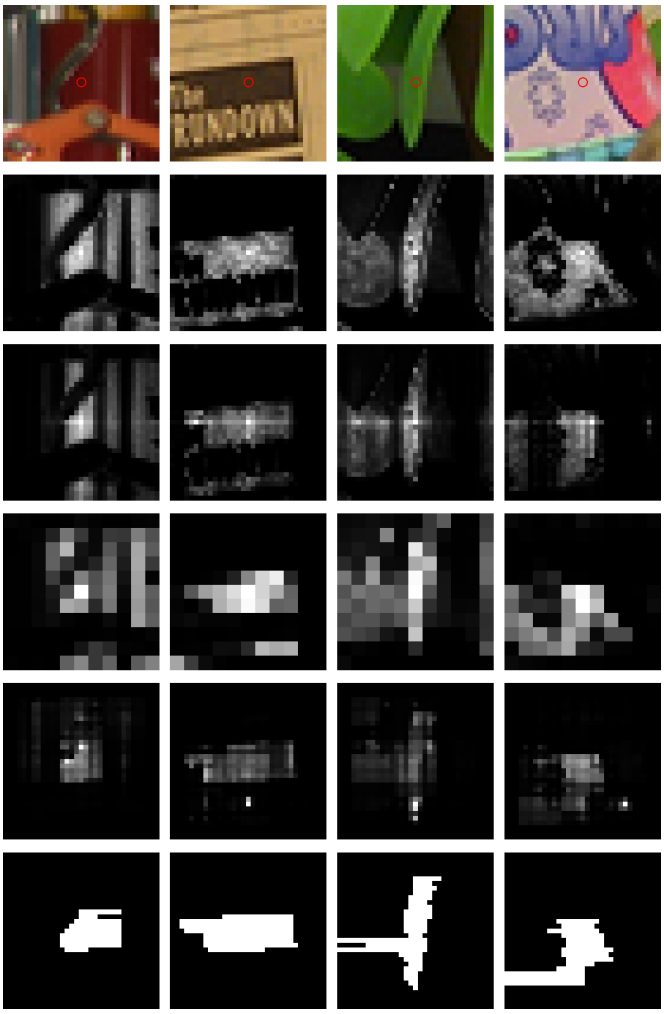

Fig. 3. Support weights generated for four different image patches (first row) using adaptive support-weights (second row), a two-pass adaptive support-weights approximation (third row), approximated joint bilateral filtering (fourth row), exponential step size adaptive weights (fifth row), and cross-based support weights (sixth row).

by an inherent inability to extend the window over local color boundaries. The binary nature of the support weights also strongly favors fronto-parallel surfaces, thus reducing the accuracy of matching when applied to curved surfaces.

Fig. 3 compares the five previously discussed methods for computing support weights. The second row of Fig. 3 shows the original adaptive support weights [5] generated for the four image patches given in the first row. The intensity images illustrating the support weights have been obtained through a linear mapping of weights to the interval [0.0, 1.0]. The results of the two-pass approximation [21], given in the third row of Fig. 3, display similar patterns when compared to the original adaptive support weights. However, as demonstrated in Fig. 1, the limitations of this method can be seen in areas where the path from neighboring pixels to the pixel of interest must pass through areas of strong dissimilarity. The fourth row demonstrates the support weights obtained using FBS with a block size of $3 \times 3$. Unlike the other approximations considered, FBS does not use a two-pass approach, thus it is does not suffer from the disadvantages of algorithms that rely on intermediate weights. However, due to the use of block averaging, the support weights are less accurate along object boundaries. The support weights produced by ESAW [9], shown in the fifth row, weakly resemble those of the original adaptive support weights, and include disproportionally high weights for pixels that are isolated from their neighbors in terms of color. These high weights result from the concatenation and normalization of weights when estimating the support weights of such isolated pixels. Finally, the binary cross-based support weights [11] are given in the last row. Although it captures nearby surface similarities around the pixel of interest, this method is incapable of crossing color boundaries, struggles to define edges, and produces streaking artifacts due to the thresholding operation.

ESAW and cross-based support weights are intended to reduce the computational complexity of matching cost aggregation, however, they introduce a tradeoff between processing time and the accuracy of support-weight approximation. As clearly illustrated by Fig. 3, the two-pass cost aggregation method outperforms both ESAW and cross-based support weights in terms of the overall accuracy of support-weight approximation. In contrast, the FBS method presents the opposite tradeoff, producing accurate support weights at the expense of increased computational complexity when compared to two-pass cost aggregation. Therefore, to achieve both highly reduced computational complexity and accurate support-weight approximation, the two-pass approximation is used by the method presented in this paper. By combining the two-pass approximation with a low-complexity iterative disparity refinement technique implemented on high-performance graphics hardware, the proposed method is able to achieve a high level of accuracy while maintaining real-time operation.

## III. Adaptive Support-Weight Stereo Matching With Iterative Disparity Refinement

In this section, a new method for iterative disparity refinement is derived that improves the accuracy of the adaptive support-weight stereo matching. Let $p \leftrightarrow \bar{p}$ denote a probabilistic event, in which pixel $\bar{p}$ in the target image is the correct match for pixel $p$ in the reference image. Recall the decision criteria for finding a match for pixel $p$, given by (3). In the ideal case, the resulting match $\bar{p} = m(p)$ is the candidate with the highest probability of being the correct match, given the two images, expressed by

$$m(p) = \underset{\bar{p} \in S_p}{\operatorname{argmax}} P(p \leftrightarrow \bar{p} \mid I, \bar{I}). \tag{4}$$

In general, the size of the images prohibits the evaluation of (4), therefore it is necessary to reduce the computational complexity associated with selection of matches. To obtain a more manageable expression, the class of window-based stereo matching methods considers only the support windows $\Omega_p$ and $\Omega_{\bar{p}}$ surrounding pixels $p$ and $\bar{p}$, respectively. A reduced-complexity approximation of (4), achieved using the window-based approach, is given by

$$m(p) \approx \underset{\bar{p} \in S_p}{\operatorname{argmax}} P(p \leftrightarrow \bar{p} \mid \Omega_p, \Omega_{\bar{p}}). \tag{5}$$

In order to further simplify the expression, and eventually lead to a formulation that facilitates iterative processing, Bayes' theorem is applied to the posterior probability on the right-hand side of (5). Under the assumption that $\Omega_p$ and $\Omega_{\bar{p}}$ are independent and equiprobable, i.e. all image patches are expected to be observed with the same frequency, the posterior probability can be expressed as

$$P(p \leftrightarrow \bar{p} \mid \Omega_p, \Omega_{\bar{p}}) = P(\Omega_p, \Omega_{\bar{p}} \mid p \leftrightarrow \bar{p}) \times P(p \leftrightarrow \bar{p}) \quad (6)$$

where $P(\Omega_p, \Omega_{\bar{p}} \mid p \leftrightarrow \bar{p})$ is the likelihood and $P(p \leftrightarrow \bar{p})$ is the prior. Because $\Omega_p$ and $\Omega_{\bar{p}}$ reside in high-dimensional probability spaces, it is computationally intractable to evaluate their probabilities. Therefore, it is often assumed that pixels located within these support windows are pairwise-independent, and the likelihood is approximated by

$$P(\Omega_p, \Omega_{\bar{p}} \mid p \leftrightarrow \bar{p}) \approx \prod_{q \in \Omega_p,\, \bar{q} \in \Omega_{\bar{p}}} P(q, \bar{q} \mid p \leftrightarrow \bar{p}). \quad (7)$$

Combining the approximations given by (6) and (7) with the matching criteria of (5) yields

$$m(p) \approx \underset{\bar{p} \in S_p}{\operatorname{argmax}} \prod_{q \in \Omega_p,\, \bar{q} \in \Omega_{\bar{p}}} P(q, \bar{q} \mid p \leftrightarrow \bar{p}) \times P(p \leftrightarrow \bar{p}). \quad (8)$$

Stereo matching is typically performed by using an additive distance metric, arbitrarily denoted by $\delta(q, \bar{q})$, to measure the dissimilarity between pixels $q \in \Omega_p$ and $\bar{q} \in \Omega_{\bar{p}}$. This is equivalent to approximating the probability distribution of the pixel likelihoods by

$$P(q, \bar{q} \mid p \leftrightarrow \bar{p}) \approx \exp\big(-\delta(q, \bar{q})\big). \quad (9)$$

Substituting this approximation into (8) results in

$$m(p) \approx \underset{\bar{p} \in S_p}{\operatorname{argmax}} \prod_{q \in \Omega_p,\, \bar{q} \in \Omega_{\bar{p}}} \exp\big(-\delta(q, \bar{q})\big) \times P(p \leftrightarrow \bar{p}). \quad (10)$$

Taking the logarithm of the expression eliminates the exponential operation, and negating the result leads to

$$m(p) \approx \underset{\bar{p} \in S_p}{\operatorname{argmin}} \sum_{q \in \Omega_p,\, \bar{q} \in \Omega_{\bar{p}}} \delta(q, \bar{q}) - \log P(p \leftrightarrow \bar{p}). \quad (11)$$

Replacing the arbitrary distance metric $\delta(q, \bar{q})$ with the adaptive support-weight matching cost of Equation (2) results in the matching criteria given by

$$m(p) \approx \underset{\bar{p} \in S_p}{\operatorname{argmin}} C(p, \bar{p}) - \log P(p \leftrightarrow \bar{p}). \quad (12)$$

For noniterative stereo matching methods, there exist no disparity estimates prior to matching. Consequently, the additive term $-\log P(p \leftrightarrow \bar{p})$ in the decision criteria of (12) is a constant for all $p$ and $\bar{p}$, and can be ignored since it does not affect the minimization. On the other hand, the nature of iterative methods allows them to incorporate the additive term by considering the disparity estimates produced after the first iteration of stereo matching. In the following, a method is given for iterative disparity refinement that uses an approximation of the additive term $-\log P(p \leftrightarrow \bar{p})$.

### A. Iterative Disparity Refinement

Let $D_p^i$ be the disparity estimate for pixel $p$ obtained in the $i$th iteration of matching. Also, let $F_p^i \in [0, 1]$ be the fractional decrease from the second lowest matching cost to the minimum matching cost, used to express the confidence level associated with the disparity estimate of pixel $p$ [20]. Essentially, the confidence level rates the uniqueness of the minimum-cost match obtained with the WTA approach by comparing its cost against the nearest competitor. Once the first iteration of stereo matching is complete, disparity estimates along with confidence levels can be used to guide matching in subsequent iterations. This is done by adding a cost penalty to candidate disparities that deviate from their expected values.

Using the disparity estimate $D_p^{i-1}$ from the previous iteration, the probability that pixel $\bar{p}$ is the correct match for pixel $p$ in the current iteration, is approximated by

$$P(p \leftrightarrow \bar{p}) \approx \exp\big(-\alpha \times \big|D_p^{i-1} - d(p, \bar{p})\big|\big) \quad (13)$$

where the scalar $\alpha$ is determined empirically. Thus, the approximation of the additive term $-\log P(p \leftrightarrow \bar{p})$ is given by

$$-\log P(p \leftrightarrow \bar{p}) \approx \alpha \times \big|D_p^{i-1} - d(p, \bar{p})\big|. \quad (14)$$

Note that the approximation in (14) only considers a single disparity estimate from the previous iteration computed exclusively for the pixel of interest, which does not facilitate message passing between pixels. To allow for the exchange of disparity information, a revised estimate of the additive term is obtained using the adaptive support weights and the confidence levels of the disparity estimates in the support window of $p$, resulting in a reformulation of (14) given by

$$-\log P(p \leftrightarrow \bar{p}) \approx \alpha \times \left| \frac{\sum_{q \in \Omega_p} w(p, q) F_q^{i-1} D_q^{i-1}}{\sum_{q \in \Omega_p} w(p, q) F_q^{i-1}} - d(p, \bar{p}) \right|. \quad (15)$$

To discourage pixel disparities from deviating from their predicted values, a cost penalty is added to the adaptive support-weight matching cost. The penalty is a weighted sum of the additive terms, as defined in (15), over the support region of $p$. The weighted sum is calculated using both the similarity of neighboring pixels and their respective confidence levels, and is given by

$$\Lambda^i(p, \bar{p}) = \alpha \times \sum_{q \in \Omega_p} w(p, q) F_q^{i-1} \left| D_q^{i-1} - d(p, \bar{p}) \right|. \quad (16)$$

Note that this cost penalty is similar to the smoothness constraint used by the nonlinear diffusion technique described in [24]; however, the proposed formulation incorporates the confidence term, uses color similarity to enforce disparity continuity, and the penalty is evaluated using a broad support region instead of using only the adjacent pixels. Finally, the penalty in (16) is incorporated into iterative disparity

refinement for stereo matching. Denoting the adaptive support-weight matching cost of (2) as $C^0(p, \bar{p})$, the matching cost in subsequent iterations is defined as

$$C^i(p, \bar{p}) = C^0(p, \bar{p}) + \Lambda^i(p, \bar{p}) \tag{17}$$

for every pair of pixels $p$ and $\bar{p}$ that are matching candidates.

After the matching costs are computed, the minimum-cost matches are found for both reference and target images using the WTA decision criteria given by (3), substituting the updated cost $C^i(p, \bar{p})$ for $C(p, \bar{p})$. The resulting matches are then used to construct disparity maps associated with both images. If $\bar{p} = m(p)$ and $p' = m(\bar{p})$, disparity $d(p, \bar{p})$ is assigned to the reference disparity map and disparity $d(p', \bar{p})$ is assigned to the target disparity map. Because this back-and-forth mapping is not always one-to-one, the following procedure is applied to the resulting disparity maps to determine if the two disparity maps are consistent. Pixel $p$ is deemed inconsistent if $|d(p, \bar{p}) - d(p', \bar{p})| > 1$, and if so, its confidence $F_p^i$ is set to zero. Similarly, this back-and-forth mapping is also used to determine inconsistencies in the disparities of the target image.

## IV. Implementation on Parallel Hardware

Modern, multithreaded, manycore GPUs deliver high computational power and high memory bandwidth. Due to their computational capabilities, GPUs have been enthusiastically adopted in numerous applications of high-performance computing, including computer vision and image processing. To address the growing demand for easy-to-use GPU-accelerated computing engines, the NVIDIA Corporation introduced the CUDA, which is a general-purpose parallel computing architecture and a complementary application programming interface that enables rapid development of massively parallel applications.

### A. CUDA Execution Model

The CUDA computing engine virtualizes graphics hardware available to the programmer through the use of uniquely numbered threads that are organized into 1-D, 2-D, or 3-D blocks of arbitrary size. A thread can be thought of as a scalar arithmetic processor, whereas a block of threads is an abstract representation of a multiprocessor composed of multiple scalar processors and capable of performing operations in parallel. The threads are executed on the graphics device equipped with a GPU, hereafter referred to as the device, serving as a coprocessor that enhances the computational capabilities of the workstation, referred to as the host.

The memory of the device is disjoint from the memory of the host, making it necessary to allocate and transfer blocks of data to the device prior to executing threads. In addition to off-chip random access memory, termed global memory, the device offers a limited amount of low-latency on-chip memory accessible to all threads within a block, referred to as shared memory. On-chip memory is also available in the form of registers, which are only accessible to individual threads.

The device code is encapsulated in special functions called kernels that are invoked by the host, and executed in parallel by multiple threads. At runtime, each block of threads gets mapped to a single multiprocessor on the device, and the threads within the block are executed in groups of 32, called warps. The execution follows the single instruction multiple thread model, which guarantees parallel execution as long as the threads in a warp do not experience a divergence of code due to branching instructions. To ensure peak performance, it is imperative to maximize the occupancy of the multiprocessors and to minimize the latency associated with global memory access by selecting the appropriate granularity of computations and the proper assignment of thread block dimensions.

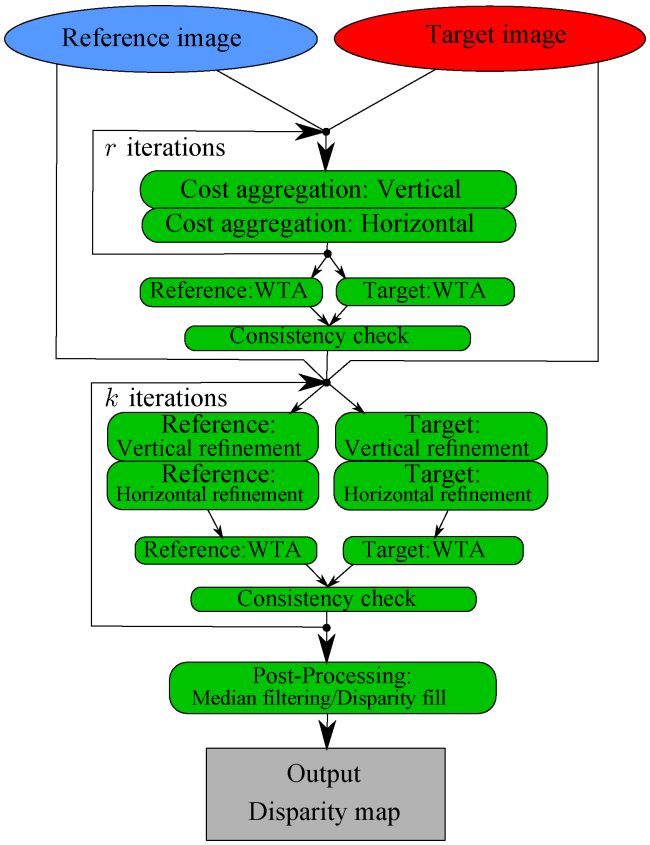


Fig. 4. Illustration of the data flow within a CUDA implementation of the proposed method. The cost aggregation kernels are called $r$ times, where $r$ is the number of disparity hypotheses, and disparity refinement kernels are called $k$ times, where $k$ is the number of disparity refinement iterations.

### B. Stereo Matching on CUDA

The implementation of the proposed method utilizes the NVIDIA GeForce GTX 580 GPU computing processor, equipped with 512 CUDA cores. The GTX 580 is a compute capability 2.0 device, allowing it to accelerate local and global memory references that exhibit spatial or temporal locality through the use of memory caching. As a consequence, image data retrieval from global memory on this device has the same latency as fetching image data from texture memory [25].

The organization of kernel functions within the proposed stereo matching framework is shown in Fig. 4. The algorithm operates by first computing the matching cost volume $V$, such that $V(x, y, d)$ contains the aggregated cost of matching pixel

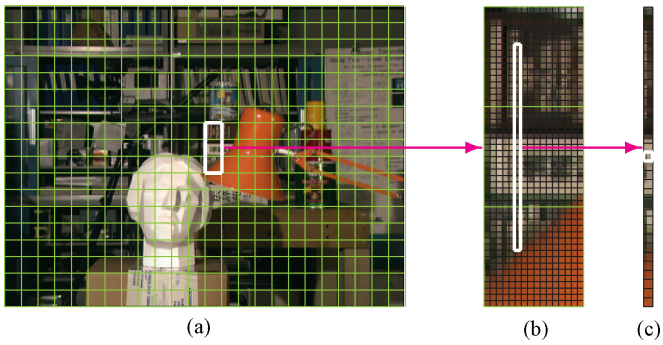


Fig. 5. Organization of thread blocks and the shared memory window used by the vertical aggregation kernel. (a) Image is broken down into a grid of blocks. (b) Each thread block uses a $48 \times 16$ window of shared memory with $16 \times 16$ buffers above and below the location of the block. (c) Threads within the block access pixels contained in vertical windows above and below the pixel of interest.

$p$ at location $(x, y)$ in the reference image to pixel $\bar{p}$ at location $(x + d, y)$ in the target image. When evaluating a particular disparity hypothesis, kernel functions are used to aggregate the cost for all pixels in both the vertical and horizontal direction to produce a single layer of the cost volume. The kernels are designed such that each thread within a block is responsible for computing the matching cost for a single pair of pixels. This granularity of computations allows the threads in each warp to take advantage of memory coalescing. Precisely, when a warp executes an instruction that references adjacent memory elements, the memory access can be coalesced into as little as one 128-byte transaction.

Because there exists significant overlap in pixels accessed by adjacent threads, each block allocates an extended window of shared memory for storing these pixels in order to reduce the latency associated with repeated global memory reads. The image data is loaded into the extended window in a way that avoids bank conflicts, since no two threads in a block write to the same bank in shared memory. Although the device allows the maximum of 1024 threads within a block, corresponding to a block size of $32 \times 32$, due to limitations in the amount of shared memory available to each multiprocessor, the chosen block size is $16 \times 16$. Choosing this block size allows more resident blocks and warps per multiprocessor, resulting in an occupancy of 83.3% for cost aggregation and disparity refinement kernels, and 100% for WTA, consistency checking, and postprocessing kernels. This high multiprocessor occupancy makes it possible for the hardware to effectively hide the latency of memory access. Fig. 5 illustrates the $48 \times 16$ extended window of shared memory used by a single block, along with the $33 \times 1$ window of pixels used by a single thread during vertical aggregation. Horizontal aggregation is performed analogously using a $16 \times 48$ extended window and a $1 \times 33$ window of pixels for each thread.

After the initial cost volume has been computed, a WTA and consistency checking kernel function is used to obtain initial disparity maps and each pixel's associated confidence level. The cost volume, disparity maps, and the sets of confidence levels are then passed to the kernels responsible for iterative refinement. Iterative refinement is performed for both the reference and target images using a two-pass, vertical-horizontal approximation of (15) similar to the method used for matching cost aggregation. Details of the CUDA kernels used for reference disparity refinement are given by Algorithms 1, 2, and 3. While performing the two-pass approximation the denominator $\mathcal{D}_p$ and disparity estimate $\mathcal{E}_p$ are stored in global memory for every pixel $p$ in order to allow for easy evaluation of the penalty function $\Lambda$, which is used by WTA to penalize disparity hypothesis that deviate from $\mathcal{E}_p$. Prior to advancing to the next iteration, disparity consistency checking is performed. If a disparity $D_p^i$ is found to be inconsistent, its confidence is set to $F_p^i = 0$.

Finally, the resulting disparity map is postprocessed using a combination of median filtering and a disparity fill operation. The $3 \times 3$ median filter is used to remove spurious noise artifacts from the disparity map. The disparity fill operation assigns valid disparity values to pixels that have been classified

**Algorithm 1** Vertical disparity refinement: iteration $i$

**for** every pixel $p = (x, y)$ **do**
  $\hat{\mathcal{N}}_p \leftarrow 0$
  $\hat{\mathcal{D}}_p \leftarrow 0$
  **for** $q+ = (x, y - \lfloor \omega/2 \rfloor)$ to $(x, y + \lfloor \omega/2 \rfloor)$ **do**
    $\hat{\mathcal{N}}_p+ = w(p, q) F_q^{i-1} D_q^{i-1}$
    $\hat{\mathcal{D}}_p+ = w(p, q) F_q^{i-1}$
  **end for**
  $\hat{\mathcal{E}}_p \leftarrow \frac{\hat{\mathcal{N}}_p}{\hat{\mathcal{D}}_p}$
**end for**

**Algorithm 2** Horizontal disparity refinement: iteration $i$

**for** every pixel $p = (x, y)$ **do**
  $\mathcal{N}_p \leftarrow 0$
  $\mathcal{D}_p \leftarrow 0$
  **for** $q+ = (x - \lfloor \omega/2 \rfloor, y)$ to $(x + \lfloor \omega/2 \rfloor, y)$ **do**
    $\mathcal{N}_p+ = w(p, q) F_q^{i-1} \hat{\mathcal{D}}_q \hat{\mathcal{E}}_q$
    $\mathcal{D}_p+ = w(p, q) F_q^{i-1} \hat{\mathcal{D}}_q$
  **end for**
  $\mathcal{E}_p \leftarrow \frac{\mathcal{N}_p}{\mathcal{D}_p}$
**end for**

**Algorithm 3** WTA: iteration $i$

**for** every pixel $p = (x, y)$ **do**
  $\text{Min}_1 \leftarrow \infty$
  $\text{Min}_2 \leftarrow \infty$
  **for** $d = 0$ to $r - 1$ **do**
    $\Lambda \leftarrow \alpha \times \mathcal{D}_p \times |\mathcal{E}_p - d|$
    **if** $\text{Min}_1 > V(x, y, d) + \Lambda$ **then**
      $D_p^i \leftarrow d$
      $\text{Min}_2 \leftarrow \text{Min}_1$
      $\text{Min}_1 \leftarrow V(x, y, d) + \Lambda$
    **else if** $\text{Min}_2 > V(x, y, d) + \alpha \mathcal{D}_p |\mathcal{E}_p - d|$ **then**
      $\text{Min}_2 \leftarrow V(x, y, d) + \Lambda$
    **end if**
  **end for**
  $F_p^i \leftarrow \frac{\text{Min}_2 - \text{Min}_1}{\text{Min}_2}$
**end for**

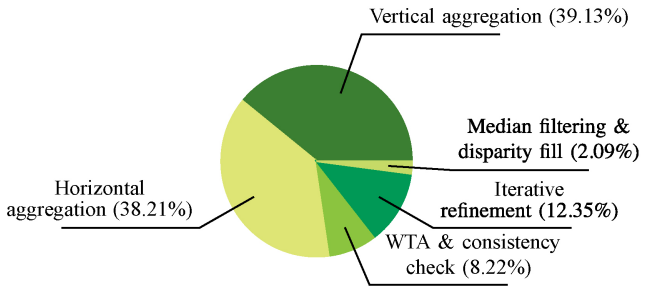


Fig. 6. Percentages of the total execution time taken by CUDA kernel functions to perform stereo matching using the proposed method.

as inconsistent in the last iteration of disparity refinement. This disparity value is computed using a weighted average of the disparities of neighboring consistent pixels located in a horizontal window centered at the pixel of interest.

### C. Relative Complexity and Runtime Distribution

Both the computational complexity of the proposed method and the runtime distribution of the kernels are discussed in the following. Let $r = d_{\max} - d_{\min}$ be the number of disparity hypotheses, $\omega$ be the window size, $k$ be the total number of refinement iterations, and $m$ and $n$ be the dimensions of the stereo images. With $s$ threads operating in parallel, the complexity of computing the matching cost volume is $\mathcal{O}(mn\omega r/s)$. Alternatively, the complexity of iterative refinement is $\mathcal{O}(mn\omega k/s)$. Thus, the increase in complexity associated with adding iterative refinement to matching cost volume computation is $k/r$. It has been determined that $k = 6$ iterations of refinement are typically required for convergence of the error rate, while the number of disparity hypotheses is typically much larger.

After testing the implementation of the proposed stereo matching method on a $640 \times 480$ image pair with $r = 60$ disparities and $k = 6$ iterations, the distribution of execution times for the various components was recorded, and is shown in Fig. 6. As expected, the vertical and horizontal aggregation operations necessary for computing the cost volume consume the majority of the processing time, at 77.34%. The addition of iterative refinement, however, only constitutes 12.35% of the total processing time. Table II shows the execution times for both a CPU and GPU implementation of the proposed method. The amount of time required to match two frames with resolution $640\times480$ with 60 disparity levels is reduced from 23.7 to 0.12 s, resulting in a speedup factor of $198\times$.

## V. Results

By comparing to existing real-time stereo matching methods, the proposed method is evaluated in terms of both speed and accuracy. The results presented in this section were obtained using the parameters $\gamma_c = 30.91$ and $\gamma_g = 28.21$ for matching cost aggregation, along with a different set of parameters $\gamma_c = 10.94$ and $\gamma_g = 118.78$ for iterative disparity refinement, and the disparity penalty was set to $\alpha = 0.085$. Note that the spatial parameter $\gamma_g$ is significantly larger in the iterative disparity refinement stage than in the cost aggregation stage, thus decreasing the effect of spatial distance on the adaptive weight relating two pixels in the same support region. As a result, the propagation of disparity information is largely unaffected by the spatial distance between pixels, and this relaxed spatial constraint promotes the spread of disparity information and allows the majority of the disparities to converge to their final values after only six iterations.

The effects of iterative processing on the Tsukuba image set are presented in Fig. 7. Note that red regions in the images correspond to pixels that are labeled as inconsistent after WTA matching and consistency checking. The error rates of the nonoccluded disparity estimates before refinement, after

TABLE I
Execution Times of Both CPU and GPU Implementations of the Proposed Stereo Matching Functions for $640\times480$ Images With 60 Disparity Levels

| Operation | Execution Time (s) | | Speedup |
|---|---|---|---|
| | CPU[1] | GPU[2] | |
| Vertical aggregation | 7.41 | 0.0469 | 158× |
| Horizontal aggregation | 8.50 | 0.0458 | 185× |
| WTA and consistency check | 3.54 | 0.0099 | 359× |
| Vertical refinement | 1.81 | 0.0079 | 230× |
| Horizontal refinement | 2.17 | 0.0069 | 313× |
| Median filtering and disparity fill | 0.27 | 0.0025 | 108× |
| Full matching | 23.7 | 0.12 | 198× |

[1]3.0 GHz AMD PhenomII X6 1075T (using a single core).
[2]NVIDIA GeForce GTX 580.

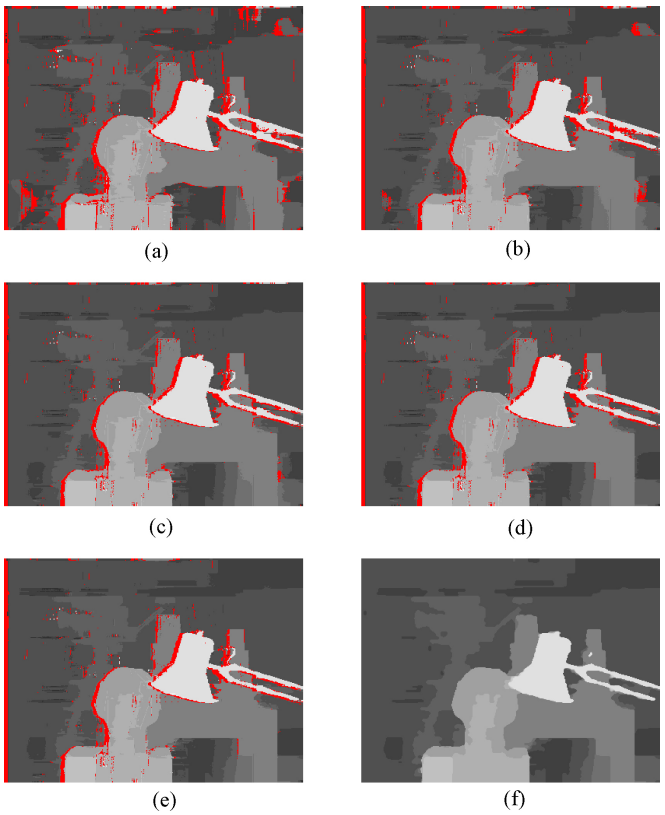


Fig. 7. Tsukuba disparity image during several stages of the iterative refinement process. (a) Prerefinement. (b) First iteration. (c) Second iteration. (d) Fourth iteration. (e) Sixth iteration. (f) After postprocessing.

TABLE II

COMPARISON OF ACCURACY AND SPEED FOR LEADING REAL-TIME STEREO MATCHING METHODS

| Method | GPU | CUDA Cores[1] | MDE/s | MDE/s/Core[1] | Speed[2] (f/s) | Avg. % Bad Pixels[3] |
|---|---|---|---|---|---|---|
| Our method | GeForce GTX 580 | 512 | 152.5 | 0.30 | 62 | 6.20 |
| FastBilateral [22] | Tesla C2070 | 448 | 50.6 | 0.11 | 21 | 7.31 |
| RealtimeBFV [11] | GeForce 8800 GTX | 128 | 114.3 | 0.89 | 46 | 7.65 |
| RealtimeBP [26] | GeForce 7900 GTX | – | 20.9 | – | 8 | 7.69 |
| ESAW [9] | GeForce 8800 GTX | 128 | 194.8 | 1.52 | 79 | 8.21 |
| RealTimeGPU [21] | Radeon XL1800 | – | 52.8 | – | 21 | 9.82 |
| DCBGrid [27] | Quadro FX 5800 | 240 | 25.1 | 0.10 | 10 | 10.90 |

[1]Applies only to devices that are compatible with CUDA.
[2]Assumes 320 × 240 images with 32 disparity levels.
[3]As measured by the Middlebury stereo performance benchmark.

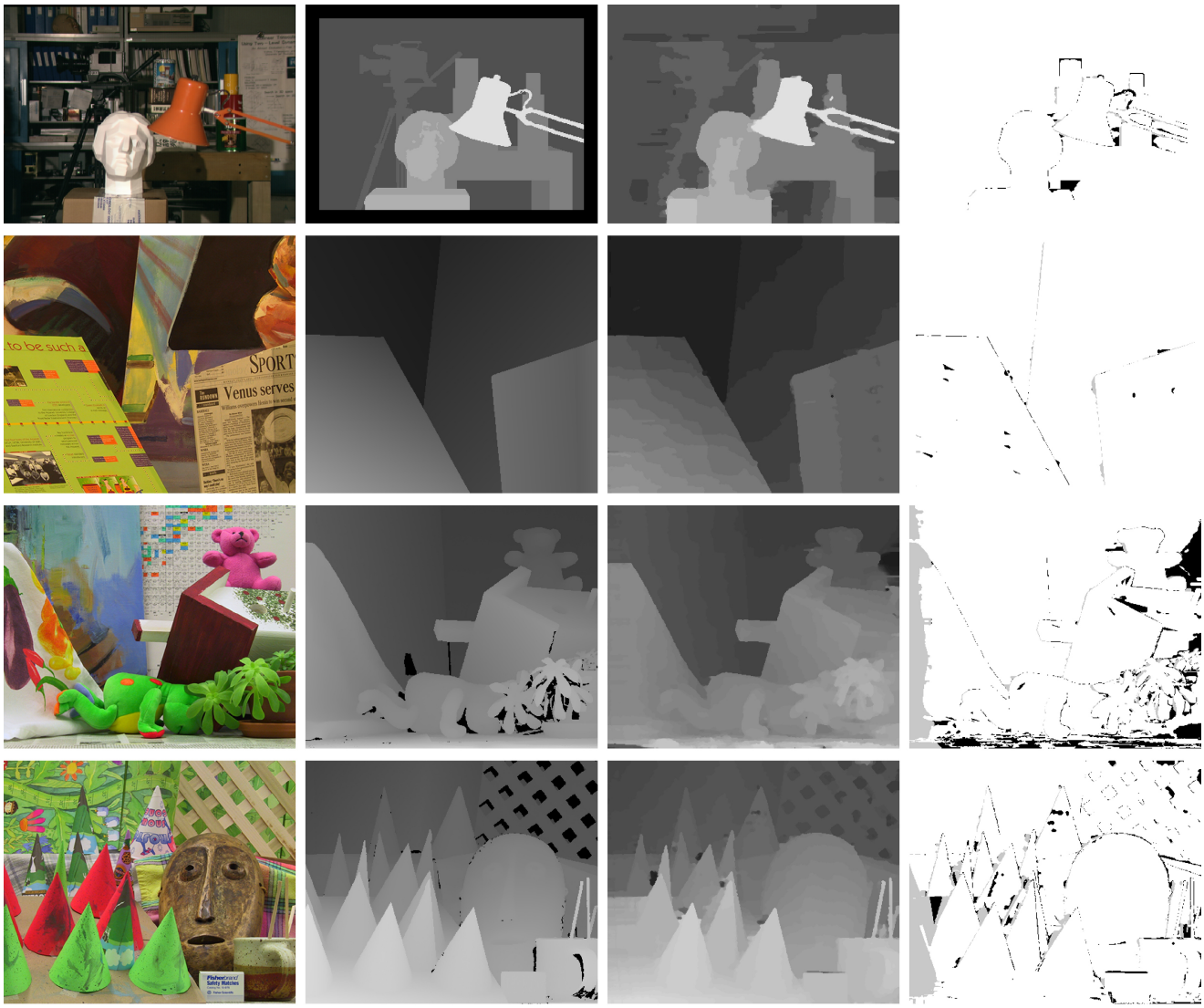


Fig. 8. Results of the proposed stereo matching method on the Tsukuba, Venus, Teddy, and Cones images from the Middlebury stereo image set.

TABLE III
RESULTS ON THE MIDDLEBURY TEST SET, AS MEASURED BY THE PERCENT OF PIXELS IN THE DENSE DISPARITY MAP WITH ABSOLUTE ERROR GREATER THAN 1 AND 2

| Method | Threshold | Avg. % Bad Pixels | Tsukuba | | | Venus | | | Teddy | | | Cones | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | NonOcc | All | Disc | NonOcc | All | Disc | NonOcc | All | Disc | NonOcc | All | Disc |
| Our method | Error > 1 | **6.20** | **1.45** | **1.99** | 7.59 | 0.40 | **0.81** | 3.38 | 7.65 | 13.3 | **16.2** | 3.48 | 9.34 | 8.81 |
| | Error > 2 | **4.24** | **1.12** | **1.58** | 5.92 | 0.21 | 0.43 | 2.83 | 4.59 | **8.09** | **9.83** | 2.40 | **7.42** | **6.45** |
| FastBilateral [22] | Error > 1 | 7.31 | 2.38 | 2.80 | 10.4 | **0.34** | 0.92 | 4.55 | 9.83 | 15.3 | 20.3 | **3.10** | **9.31** | **8.59** |
| | Error > 2 | 5.18 | 2.08 | 2.36 | 8.80 | **0.18** | **0.39** | **2.30** | 6.94 | 9.30 | 13.1 | **2.33** | 7.59 | 6.74 |
| RealtimeBFV [11] | Error > 1 | 7.65 | 1.71 | 2.22 | **6.74** | 0.55 | 0.87 | **2.88** | 9.90 | 15.0 | 19.5 | 6.66 | 12.3 | 13.4 |
| | Error > 2 | 5.51 | 1.48 | 1.96 | **5.58** | 0.36 | 0.62 | 2.47 | 7.00 | 9.73 | 13.0 | 4.66 | 9.59 | 9.68 |
| RealtimeBP [26] | Error > 1 | 7.69 | 1.49 | 3.40 | 7.87 | 0.77 | 1.90 | 9.00 | 8.27 | **13.2** | 17.2 | 4.61 | 11.6 | 12.4 |
| | Error > 2 | 5.43 | 1.25 | 3.04 | 6.66 | 0.63 | 1.53 | 7.68 | 5.68 | 8.27 | 10.2 | 2.90 | 9.11 | 8.27 |
| ESAW [9] | Error > 1 | 8.21 | 1.92 | 2.45 | 9.96 | 1.03 | 1.65 | 6.89 | 8.48 | 14.2 | 18.7 | 6.56 | 12.7 | 14.4 |
| | Error > 2 | 5.81 | 1.67 | 2.13 | 8.30 | 0.67 | 1.15 | 5.73 | 5.48 | 8.45 | 12.5 | 4.09 | 9.98 | 9.56 |
| RealTimeGPU [21] | Error > 1 | 9.82 | 2.05 | 4.22 | 10.6 | 1.92 | 2.98 | 20.3 | **7.23** | 14.4 | 17.6 | 6.41 | 13.7 | 16.5 |
| | Error > 2 | 6.46 | 1.34 | 3.27 | 7.17 | 1.02 | 1.90 | 12.4 | **3.90** | 8.65 | 10.4 | 4.37 | 10.8 | 12.3 |
| DCBGrid [27] | Error > 1 | 10.90 | 5.90 | 7.26 | 21.0 | 1.35 | 1.91 | 11.2 | 10.5 | 17.2 | 22.2 | 5.34 | 11.9 | 14.9 |
| | Error > 2 | 7.94 | 4.21 | 5.16 | 17.2 | 0.98 | 1.23 | 7.89 | 7.69 | 10.8 | 15.2 | 3.95 | 9.48 | 11.5 |

"NonOcc" denotes the set of nonoccluded pixels in the disparity map, and "disc" denotes the areas in the disparity map that contain discontinuities. Bold entries indicate a #1 ranking among all real-time algorithms.

iteration 1, 2, 4, 6, and after postprocessing are 6.26%, 4.03%, 3.06%, 2.93%, 2.92%, and 1.45%, respectively. Improvements in the disparity map can be seen as the number of iterations increases, using the ground truth disparity as a reference (shown in Fig. 8). These improvements are especially noticeable in the background areas on the bottom left and bottom right, and the area to the right of the sculpture head.

The accuracy of real-time stereo matching methods is measured using the Middlebury stereo benchmark [13], [14], which uses four stereo image pairs along with the percentage of incorrectly assigned disparities as a way to quantify stereo matching accuracy. The number of frames per second (f/s) and the number of millions of disparity estimates per second (MDE/s) are the two most common metrics used to evaluate the speed of the implementation. The accuracy and speed presented in Table II provide a comparison between the proposed method and other real-time stereo matching methods, where accuracy is summarized as the average percentage of bad pixels across all test images. Though slower than both ESAW and RealtimeBFV, which uses cross-based aggregation, the proposed method improves upon the average percentage of bad pixels by 2.01% and 1.45%, respectively, while maintaining a high frame rate.

The results of the proposed method are given in Fig. 8, and a detailed comparison to other top-performing real-time methods is presented in Table III. The error images in Fig. 8 indicate that the method is capable of reproducing highly accurate edge discontinuities and smooth surfaces. It performs particularly well on the Tsukuba, Venus, and Cones image sets, but the method struggles to produce accurate disparities for Teddy in the white, untextured area to the right of the stuffed bear and the highly slanted floor surface at the bottom of the images. These two areas of the Teddy image set are challenging for methods that do not use image segmentation and/or plane-fitting.

As indicated by the results presented in Table III, the proposed method performs very well when compared to other top-performing real-time stereo matching methods. It ranks 1st in 46.15% of categories used to evaluate matching performance for the four images, and its nearest competitor in terms of the average percentage of bad pixels, FastBilateral, produces an average of 20% more errors. The RealtimeBFV method produces more accurate results in the discontinuous regions of the Tsukuba and Venus images. This may be due to its strict binary assignment of weights near boundaries, and the fact that all weights on the other side of boundaries are forced to zero.

## VI. CONCLUSION

This paper presented a novel method for high-quality real-time stereo matching. The proposed method used a two-pass approximation of adaptive support weights for matching cost aggregation, and an iterative refinement technique that enforced consistency of disparities. Although more computationally complex than other adaptive support weight approximation methods, the two-pass approach produced the most accurate results and, when implemented on modern high-performance graphics hardware, was capable of achieving real-time operation. By introducing an additive cost term into the match selection criteria, the refinement technique iteratively improved the accuracy of the disparity map and typically converged after only six iterations. The added complexity associated with iterative refinement was shown, both analytically and experimentally, to be relatively small when compared to the complexity of matching cost aggregation. When evaluated using the Middlebury stereo benchmark, the proposed method outperformed all existing real-time stereo matching methods in terms of accuracy.

## REFERENCES

[1] A. Klaus, M. Sormann, and K. Karner, "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure," in *Proc. 18th ICPR*, vol. 3. 2006, pp. 15–18.

[2] M. Bleyer, C. Rother, and P. Kohli, "Surface stereo with soft segmentation," in *Proc. IEEE Conf. CVPR*, Jun. 2010, pp. 1570–1577.
[3] Q. Yang, C. Engels, and A. Akbarzadeh, "Near real-time stereo for weakly textured scenes," in *Proc. Br. Mach. Vision Conf.*, 2008, pp. 72.1–72.10.
[4] L. Xu and J. Jia, "Stereo matching: An outlier confidence approach," in *Proc. Comput. Vision ECCV*, LNCS 5305. 2008, pp. 775–787.
[5] K.-J. Yoon and I.-S. Kweon, "Locally adaptive support-weight approach for visual correspondence search," in *Proc. IEEE Comput. Soc. Conf. CVPR)*, vol. 2. Jun. 2005, pp. 924–931.
[6] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann, "Local stereo matching using geodesic support weights," in *Proc. 16th IEEE Int. Conf. Image Process.*, Nov. 2009, pp. 2093–2096.
[7] E. T. Psota, J. Kowalczuk, J. Carlson, and L. C. Pérez, "A local iterative refinement method for adaptive support-weight stereo matching," in *Proc. Int. Conf. IPCV*, Jul. 2011, pp. 271–277.
[8] S. Grauer-Gray and C. Kambhamettu, "Hierarchical belief propagation to reduce search space using CUDA for stereo and motion estimation," in *Proc. WACV*, Dec. 2009, pp. 1–8.
[9] W. Yu, T. Chen, F. Franchetti, and J. C. Hoe, "High performance stereo vision designed for massively data parallel platforms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 11, pp. 1509–1519, Nov. 2010.
[10] Y. Zhao and G. Taubin, "Real-time stereo on GPGPU using progressive multi-resolution adaptive windows," *Image Vision Comput.*, vol. 29, no. 6, pp. 420–432, 2011.
[11] K. Zhang, J. Lu, Q. Yang, G. Lafruit, R. Lauwereins, and L. Van Gool, "Real-time and accurate stereo: A scalable approach with bitwise fast voting on CUDA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 7, pp. 867–878, Jul. 2011.
[12] M. Carreira-Perpinan, "Acceleration strategies for Gaussian mean-shift image segmentation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognit.*, vol. 1. Jun. 2006, pp. 1160–1167.
[13] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, pp. 7–42, Apr.–Jun. 2002.
[14] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proc. IEEE Comput. Soc. Conf. Computer Vision Pattern Recognit.*, vol. 1. Jun. 2003, pp. 195–202.
[15] A. Fusiello, V. Roberto, and E. Trucco, "Efficient stereo with multiple windowing," in *Proc. IEEE Comput. Soc. Conf. Computer Vision Pattern Recognit.*, Jun. 1997, pp. 858–863.
[16] S. B. Kang, R. Szeliski, and J. Chai, "Handling occlusions in dense multi-view stereo," in *Proc. IEEE Comput. Soc. Conf. CVPR*, vol. 1. 2001, pp. I-103–I-110.
[17] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions using graph cuts," in *Proc. 8th IEEE Int. Conf. Comput. Vision*, vol. 2. Jul. 2001, pp. 508–515.
[18] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 7, pp. 787–800, Jul. 2003.
[19] Z.-F. Wang and Z.-G. Zheng, "A region based stereo matching algorithm using cooperative optimization," in *Proc. IEEE CVPR*, Jun. 2008, pp. 1–8.
[20] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister, "Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 3, pp. 492–504, Mar. 2009.
[21] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in *Proc. 3rd Int. Symp. 3DPVT*, 2006, pp. 798–805.
[22] S. Mattoccia, M. Viti, and F. Ries, "Near real-time fast bilateral stereo on the GPU," in *Proc. IEEE Comput. Soc. Conf. CVPRW*, Jun. 2011, pp. 136–143.
[23] S. Mattoccia, S. Giardino, and A. Gambini, "Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering," in *Proc. ACCV*, vol. 5995. 2009, pp. 371–380.
[24] D. Scharstein and R. Szeliski, "Stereo matching with nonlinear diffusion," *Int. J. Comput. Vision*, vol. 28, pp. 155–174, Jun. 1998.
[25] *NVIDIA CUDA C Programming Guide*, 4.0 ed., Mar. 2011.

**Jędrzej Kowalczuk** (S'11) received the B.S. and M.S. degrees in computer science from the Technical University of Szczecin, Szczecin, Poland, in 2007. He is currently pursuing the Ph.D. degree in electrical engineering with the University of Nebraska-Lincoln (UNL), Lincoln.

He is currently a Research Assistant with the Department of Electrical Engineering, UNL. His current research interests include multiview computer vision, image processing, and high-performance computing.

**Eric T. Psota** (M'05) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Nebraska-Lincoln (UNL), Lincoln, in 2004, 2006, and 2010, respectively.

He is currently a Research Assistant Professor with the Department of Electrical Engineering, UNL. His current research interests include multiview computer vision, biomedical image processing, and the study of iterative decoders for error control coding.

Dr. Psota is a member of Eta Kappa Nu.

**Lance C. Pérez** (SM'87) received the B.S. degree in electrical engineering from the University of Virginia, Charlottesville, and the M.S. and Ph.D. degrees in electrical engineering from the University of Notre Dame, Notre Dame, IN.

He has been a Faculty Member with the Department of Electrical Engineering, University of Nebraska-Lincoln, Lincoln, since August 1996, where he is currently a Professor of electrical engineering. His current research interests include error control coding, information, image processing, and engineering education.

Dr. Pérez is a member of ASEE, Tau Beta Pi, and Eta Kappa Nu.