Real-Time Dense Stereo on a Personal Computer

L. Di Stefano, M. Marchionni, S. Mattoccia
DEIS-ARCES, University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
ldistefano@deis.unibo.it, mmarchionni@litio.it, smattoccia@deis.unibo.it

Abstract

This paper presents a stereo algorithm that enables real time dense disparity measurements on standard personal computers. Unlike many other dense stereo algorithms, which are based on two matching phases, the proposed algorithm relies on a single matching phase and allows for rejecting unreliable matches by exploiting violations of the uniqueness constraint and analysing the behaviour of the correlation scores. The overall algorithm has been carefully optimised using very efficient calculation schemes and deploying massively the SIMD parallel processing capabilities available nowadays in state-of-the-art general purpose microprocessors. The paper describes the algorithm and the optimisation strategies, and provides experimental results obtained on stereo pairs with ground-truth as well as execution times measurements.

1 Introduction

Dense depth measurements are required in many applications such as robot navigation and control, exploration and modelling of unstructured environments, virtual reality. According to a recent taxonomy [17], stereo algorithms that generate dense depth measurements can be divided into two classes, namely *global* and *local* algorithms. The former algorithms rely on iterative schemes and assign disparity values on the basis of the minimisation of a global cost function. These algorithms yield accurate and dense disparity measurements [17] but are unsuited to real-time applications. The latter (*local* or *area-based*) algorithms (e.g. [7, 8, 12, 14]) calculate the disparity at each pixel on the basis of the photometric properties of the neighbouring pixels. The *left-right check* [9] is a widely adopted method [8, 14, 7] aimed at detecting unreliable matches in local algorithms. It consists of two matching phases: first, for each point of the left image the best match into the right image is found; then, the role of the two images is reversed and for each point of the right image the best match into the left image is found. Finally, only those matches that turn out to be coherent are accepted as valid.

Compared to *global* algorithms, *local* algorithms yield significantly less accurate disparity maps but, nowadays, thanks to both research and technology advances, can run fast enough to be deployed in many real-time applications. Moreover, local algorithms exhibits a quite regular computational structure which is suited to deploy the SIMD (*Single Instruction Multiple Data*) parallel instructions available in state-of-the-art general purpose microprocessors (such a those by Intel, Hewlett-Packard, Sun, AMD and Motorola).

This paper presents a fast *local* algorithm which enables real-time dense stereo applications on a standard Personal Computer exploiting efficient calculation schemes and deploying the SIMD parallel capabilities available in general purpose processors. The algorithm is based on a matching core that detects unreliable matches during the direct matching phase and therefore does not require a reverse matching phase.

2. Related work

Many works have dealt with dense stereo but only recently real-time systems based on personal computer Here we briefly review some important contributions towards real-time stereo matching. We first review the systems based on dedicated hardware and then those running on a personal computer. Kanade et al. [12] de-

0-7803-7970-5/03/$20.00 ©2003 IEEE

veloped a system capable of reliable depth maps using multiple cameras (up to 6) working at video-rate with 200×200 pixels images using a custom DSP machine. The matching measure is the Sum of Squared Differences (SSD) and a global value (SSSD) is computed adding the contributes given by the different matches between the various views. The stereo system developed at INRIA by Faugeras et al. [8] uses an imaging system with three cameras and the normalized cross correlation (NCC) as matching measure. Match validation is carried out by the left-right consistency check and analyzing the peaks of the correlation function. The algorithm by Faugeras et al. [8] was implemented on dedicated DSP machines. More recently, Interval Research [19] developed a fast FPGA-based board which fits into a standard PCI board. The system executes 42 disparity measurements per seconds using as input a stereo pair from two cameras at 320×240 pixels of resolution. The matching core is based on the census transform [19]. Another FPGA based stereo system which relies on the census transform was developed by Corke et al. [5]. It delivers 30 fps with 256×256 stereo pairs.

The system developed at SRI [14] provides real-time stereo matching. It does disparity estimation using two images which are preprocessed via the LOG filter and then matched on the basis of the Sum of Absolute Differences (SAD) measure. Point Grey Research [2] developed a stereo system made out of a three CCD cameras and a stereo algorithm based on the SAD measure. Both systems rely on two matching phases (i.e. left-right check) and run on a personal computer.

3 The matching algorithm

In this section we describe the proposed stereo matching algorithm assuming binocular stereo pairs and images in standard form (i.e. with corresponding epipolar lines lying on corresponding image scanlines). Should the latter assumption not be verified, a suitable transformation, known as rectification (see for example [10]), must be applied to obtain a pair of images in standard form from the original ones.

In *local* algorithms, given a point in the reference image the homologous point is selected by searching along the corresponding scanline in the other image, and within a certain disparity range, for the point that minimize (maximize) an error (similarity) function, ε, representing the degree of dissimilarity (similarity) between two small regions of the stereo pair. Unlike algorithms based on the left-right check, which rely on a direct (i.e. left-to-right) and a reverse (i.e. right-to-left) matching phase, our algorithm uses only a direct matching phase. Our approach relies on the *uniqueness constraint*, which states that a 3D point can be projected at most in one point of each image of the stereo pair, as well as on the ability of modifying disparity measurements dynamically as long as the matching process proceeds.

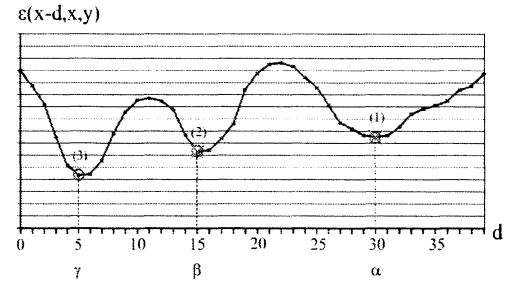

Figure 1. Scores associated with point $R(x, y)$.

Let's assume that the left image is the reference, that disparity, d, belongs to the interval $[0 \ldots d_{max}]$ and that the left image is scanned from top to bottom and from left to right during the matching process. The algorithm, starting from one point of left image, say $L(x - d_{max}, y)$, searches for the best candidate by evaluating function ε, within the interval $[R(x - d_{max}, y) \ldots R(x, y)]$. Then, for the successive point of reference image $L(x + 1 - d_{max}, y)$ the procedure is repeated searching for the best match within $[R(x + 1 - d_{max}, y) \ldots R(x + 1, y)]$. The process is then iterated for the successive points along the scanline.

Suppose now that the best match found for $L(x + \beta - d_{max}, y)$ is $R(x, y)$, with similarity score $\varepsilon(x + \beta - d_{max}, x, y)$. We adopt the notation $L(x + \beta - d_{max}, y) \rightharpoonup R(x, y)$ to indicate that this match from left to right has been established.

As it is common in area-based algorithms we use photometric properties, encoded by the error (similarity) function, as the main cue driving the match-

ing process, even though this cue may be ambiguous, due to many causes such as for example photometric distortions, occlusions and signal noise. However, wrong matches expose inconsistencies within the set of matches already established that can be deployed to detect and discard them.

Thus, let's suppose that another point of the left image, say $L(x+\alpha-d_{max},y)$, with $\alpha \leq \beta$, has previously matched with $R(x,y)$ with score $\varepsilon(x+\alpha-d_{max},x,y)$. This situation, that violates the uniqueness constraint, is used in our approach to detect wrong matches. In fact, based on the uniqueness constraint we assume that at least one of the two matches, i.e. $L(x+\beta-d_{max},y) \rightharpoonup R(x,y)$ or $L(x+\alpha-d_{max},y) \rightharpoonup R(x,y)$, is wrong and retain the match having the better score. Thus, if the point currently analyzed $L(x+\beta-d_{max},y)$ has a better score than $L(x+\alpha-d_{max},y)$ (i.e. $\varepsilon(x+\beta-d_{max},x,y) \leq \varepsilon(x+\alpha-d_{max},x,y)$) our algorithm will reject the previous match and accept the new one. This implies that, although the proposed approach relies on a direct matching phase only, it allows for recovering from possible previous matching errors.

The capability of our algorithm to recover from previous errors as long as better matches are found during the search is shown in Figure 1, which plots as a function of $d \in [0 \ldots d_{max}]$ all the scores between the point $R(x,y)$ of right image and the points in the reference image $[L(x-d_{max},y) \ldots L(x,y)]$ that are allowed to establish a correspondence with $R(x,y)$.

The reliability of the disparity measurements provided by the basic matching core can be improved by incorporating additional constraints. Since our algorithm is aimed at real-time applications, we introduce new constraints by analysing the behaviour of the error (similarity) function. This allows us to assess the reliability of the matches at a very small computational cost by exploiting the data already computed by the matching core as well as the SIMD-processing capabilities provided by general purpose processors.

In our algorithm the global minimum of the SAD (Sum of Absolute Differences) error function is located very quickly using a parallel technique, described in section 5, that with a few SIMD instructions (i.e. MMX instructions) yields the score (SAD_{min}) and the position within the disparity range (d_{min}) of the global minimum as well as the scores (SAD_1, SAD_2, SAD_3) and positions (d_1, d_2, d_3) of three candidate minima, referred to as *pseudo-minima*. These exhibit small error scores but are not guaranteed to correspond to local minima. To discard ambiguous matches we estimate [6] the behaviour of the error function by means of two tests, called *distinctiveness test* and *sharpness test*, that are carried out using only the global minimum and the three *pseudo-minima*.

When the three *pseudo-minima* fall far from the position of the global minimum the match is potentially ambiguous (for example, this happens in presence of repetitive patterns), unless the error score of the global minimum is much smaller than those of the *pseudo-minima*. On the other hand, when the *pseudo-minima* are close to the position of the global minimum, the match can be considered reliable even though the score of the global minimum turns out not much smaller than those of the *pseudo-minima*.

The following relationship, referred to as *sharpness test*, evaluates the degree of aggregation of the *pseudo-minima* in proximity of the global minimum:

$$\Delta d = \sum_{i=1}^{3} |d_i - d_{min}| \qquad (1)$$

A low Δd value (the lowest value is 4) indicates that the *pseudo-minima* are localized in proximity of the global minimum and thus the match is accepted as reliable. Conversely an high Δd value means that the *pseudo-minima* are spread within the disparity range and hence the match is potentially ambiguous. In order to evaluate the reliability of the matches that do not satisfy the *sharpness test* we perform an additional test, referred to as *distinctiveness test*, aimed at evaluating whether the score of the global minimum is much smaller than those of the *pseudo-minima*. The following relation

$$\Delta SAD = \sum_{i=1}^{3} (SAD_i - SAD_{min}) \qquad (2)$$

embodies information about the distinctiveness of the global minimum with respect to the three *pseudo-minima*. Actually, we consider the ratio $\frac{\Delta SAD}{SAD_{min}}$ to evaluate the distinctiveness of the global minimum, with high ratios indicating distinctive global minima.

The overall stereo algorithm consists of the four major steps shown in Figure 2).

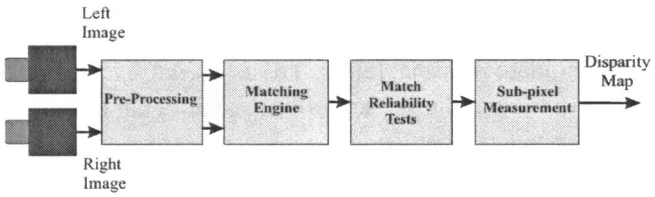

Figure 2. The overall stereo algorithm.

The input images are normalized by subtraction of the mean values of the intensities computed in a small window centered at each pixel [8]. This allows for compensating for different settings of the cameras and-or different photometric conditions. Moreover, since matching turns out to be highly unreliable when dealing with poorly textured areas, the variance of the intensities is calculated at each pixel considering a window of the same size as that used to obtain the means. This information is used to detect regions with lack of texture [8]. The normalized images are matched according to the matching approach described previously and using the SAD error function. The reliability of the matches provided by basic matching core is improved by means of the *distinctiveness* and *sharpness* tests. In addition, this step uses the variance map computed in the pre-processing step to reject the matches found in poorly textured areas. The final step performs $\frac{1}{16}$ sub-pixel refinement of disparities.

4 Computational optimisation

In this section we first review a technique aimed at reducing the number of redundant operations already described in [7]. Successively, we show how to achieve a further improvements of the technique by introducing an additional level of incremental calculation.

The computation of SAD scores is the most expensive task performed by the stereo algorithm. Suppose $SAD(x, y, d)$ is the SAD score between a window of size $(2n+1) \times (2n+1)$ centered at coordinates (x, y) in the left image and the corresponding window centered at $(x + d, y)$ in the right image:

$$SAD(x, y, d) = \sum_{i,j=-n}^{n} |L(x + j, y + i) - R(x + d + j, y + i)| \quad (3)$$

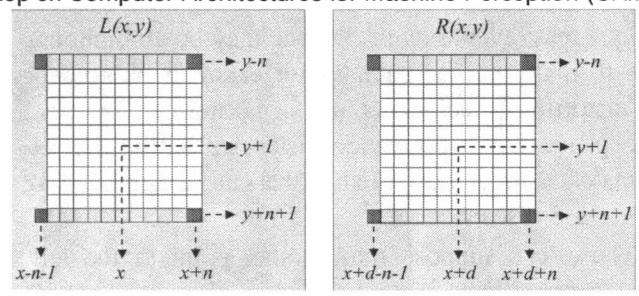

Figure 3. Recursive calculation of SAD scores.

Observing Figure 3, it is easy to notice that $SAD(x, y+1, d)$ can be attained from $SAD(x, y, d)$:

$$SAD(x, y + 1, d) = SAD(x, y, d) + U(x, y + 1, d) \quad (4)$$

with $U(x, y + 1, d)$ representing the difference between the SADs associated with the lowermost and uppermost rows of the matching window (shown in light-gray in Figure 3):

$$U(x, y + 1, d) = \sum_{j=-n}^{n} |L(x + j, y + n + 1) - R(x + d + j, y + n + 1)| - \sum_{j=-n}^{n} |L(x + j, y - n) - R(x + d + j, y - n)| \quad (5)$$

Furthermore, $U(x, y + 1, d)$ can be computed from $U(x-1, y+1, d)$ by simply considering the contributes associated with the four points shown in dark-gray in Figure 3:

$$\begin{aligned} U(x, y + 1, d) = {} & U(x - 1, y + 1, d) + \\ & (|L(x + n, y + n + 1) - R(x + d + n, y + n + 1)| - \\ & |L(x + n, y - n) - R(x + d + n, y - n)|) - \\ & (|L(x - n - 1, y + n + 1) - \\ & R(x + d - n - 1, y + n + 1)| - \\ & |L(x - n - 1, y - n) - R(x + d - n - 1, y - n)|) \end{aligned} \quad (6)$$

This allows for keeping complexity small and independent from the size of the matching window, since only four elementary operation are needed to obtain the SAD score at each new point.

The pre-processing step requires computation of the mean and variance of the two images. Considering the left image, and posing $N^2 = (2n+1) \cdot (2n+1)$, the mean is given by

$$\mu(x,y) = \frac{1}{N^2} \sum_{i,j=-n}^{n} L(x+j, y+i) = \frac{1}{N^2} S_1(x,y) \tag{7}$$

while the variance can be expressed [4] as

$$\begin{aligned} \sigma^2(x,y) &= \frac{1}{N^2} \sum_{i,j=-n}^{n} L(x+j, y+i)^2 - \mu^2(x,y) \\ &= \frac{1}{N^2} S_2(x,y) - \mu^2(x,y) \end{aligned} \tag{8}$$

Since equations (7) and (8) rely on the same basic operation it can be easily verified that the computation of mean and variance can be carried out using the following schemes:

$$S_1(x, y+1) = S_1(x,y) + U_{S_1}(x, y+1) \tag{9}$$

$$U_{S_1}(x, y+1) = \sum_{j=-n}^{n} (L(x+j, y+n+1) - L(x+j, y-n)) \tag{10}$$

$$\begin{aligned} &U_{S_1}(x, y+1) = U_{S_1}(x-1, y+1) + \\ &(L(x+n, y+n+1) - L(x+n, y-n)) - \\ &(L(x-n-1, y+n+1) - L(x-n-1, y-n)) \end{aligned} \tag{11}$$

$$S_2(x, y+1) = S_2(x,y) + U_{S_2}(x, y+1) \tag{12}$$

$$U_{S_2}(x, y+1) = \sum_{j=-n}^{n} (L(x+j, y+n+1)^2 - L(x+j, y-n)^2) \tag{13}$$

$$\begin{aligned} &U_{S_2}(x, y+1) = U_{S_2}(x-1, y+1) + \\ &(L(x+n, y+n+1)^2 - L(x+n, y-n)^2) - \\ &(L(x-n-1, y+n+1)^2 - L(x-n-1, y-n)^2) \end{aligned} \tag{14}$$

In both matching and pre-processing steps it is possible to introduce a third level of incremental computation aimed at achieving additional speed-up. Formulas 6, 11 and 14 show that the pixels at the corners of the correlation window contribute to two terms, say A and B, where A includes the two pixels on the left corners and B those on the right.

Because term B plays the role of term A when the correlation window is shifted horizontally by $2n+1$ units, with a very small memory cost we can store the most recent $2n+1$ B terms so that they can be re-used $2n+1$ units later in place of the A terms. Naming T the array of the B terms, each element can be referenced with the index $\tilde{x} = x \ mod \ (2n+1)$ and thus all elements are visited each time the correlation window is shifted horizontally by $2n+1$ units. When shifting the window by one unit, a new B term is calculated while the needed A term is fetched from $T(\tilde{x})$. After both terms have been used, $T(\tilde{x})$ is updated to the newly calculated B term.

To introduce this third level of incremental computation into the pre-processing step, formula 11 for the mean calculation is rewritten as follows:

$$\begin{aligned} &U_{S_1}(x, y+1) = U_{S_1}(x-1, y+1) + \\ &(L(x+n, y+n+1) - L(x+n, y-n)) - T_1(\tilde{x}) \end{aligned} \tag{15}$$

where

$$\begin{aligned} &T_1(\tilde{x}) = \\ &L(x-n-1, y+n+1) - L(x-n-1, y-n) \\ &\text{with} \quad \tilde{x} = x \ mod \ (2n+1) \end{aligned} \tag{16}$$

Similarly formula 14 for the variance calculation becomes:

$$U_{S_2}(x, y+1) = U_{S_2}(x-1, y+1) + (L(x+n, y+n+1)^2 - L(x+n, y-n)^2) - T_2(\tilde{x}) \quad (17)$$

where

$$T_2(\tilde{x}) = (L(x-n-1, y+n+1)^2 - L(x-n-1, y-n)^2) \quad \text{with} \quad \tilde{x} = x \; mod \; (2n+1) \quad (18)$$

In the matching step the third level of incremental computation is applied for each disparity value $d \in [0, d_{max}]$; thus, the array T grows by one dimension and formula 6 is rewritten as follows:

$$U(x, y+1, d) = U(x-1, y+1, d) + (|L(x+n, y+n+1) - R(x+d+n, y+n+1)| - |L(x+n, y-n) - R(x+d+n, y-n)|) - T(\tilde{x}, d) \quad (19)$$

where

$$T(\tilde{x}, d) = |L(x-n-1, y+n+1) - R(x+d-n-1, y+n+1)| - |L(x-n-1, y-n) - R(x+d-n-1, y-n)| \quad \text{with} \quad \tilde{x} = x \; mod \; (2n+1), d \in [o, d_{max}] \quad (20)$$

5 Mapping on a processor with parallel SIMD instructions

Our stereo algorithm is designed to run on Intel Pentium processors providing at least the MMX [16] multimedia instruction set. Because the new generation of Pentium processors (Pentium III and Pentium 4) provide a better support for SIMD (Single Instruction Multiple Data) programming with the SSE (Streaming SIMD Extensions) [18] instruction set, we also developed an optimised SSE version which will be discussed next.

5.1 Pre-processing step

The *preprocessing step* takes in input the left and right frame buffers, calculates the means and obtain the normalized frames used in the matching step. In addition, the left frame variances are also evaluated. The pre-processing steps relies on the incremental calculation scheme described in section 4 to dramatically decrease the number of operations executed at each pixel. Figure 4 shows which steps are taken to preprocess an input frame. The pixel (n, n) is processed first to compute its mean by running two nested loops completely visiting all the pixels in the window centered at (n, n). With this step $2n + 1$ elements are stored in a vector called `DeltaMean` containing each the sum of the pixels within the light gray vertical stripes. $2n + 1$ elements are also stored in a vector called `DeltaVariance` for the sum of squared values of the same pixels.

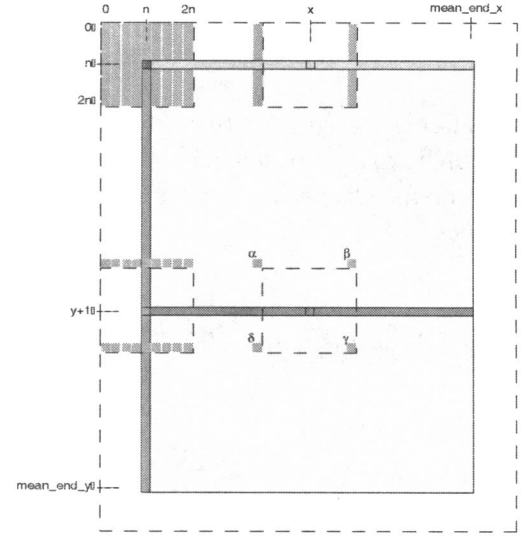

Figure 4. Pixels involved in the preprocessing step.

Next step consists in moving to the remaining pixels of the first row applying the horizontal recursion. For a given pixel $(x, n), x > n$, the mean computation needs the sum of values of all pixels in the window centered at (x, n). Those quantities can be obtained from the same sum computed for the previous pixel $(x - 1, n)$, with a correction given by the two vertical strips shown in Figure 4 on the left and

right sides of the window centered at (x, n). The sum of values referred to the strip on the left is found in `DeltaMean` vector in position $j = (x-n) mod (2n+1)$. For each pixel (x, n) is then required one loop only, to compute the sum of values of the pixels in the right strip. An identical process making use of the `DeltaVariance` vector is carried out to compute the variance (sum of squared values) for the pixels $(x, n), x > n$ in the left image.

A generic row in a frame is processed by making strong use of SIMD instructions. This is possible by first initializing some data structures while processing the first pixel $(n, y+1)$ of the current row. The mean for the pixel $(n, y+1)$ is given by the sum of values referred to the pixels in the window centered in (n, y) and the corrections represented by the horizontal strips on the top and bottom of the window centered at $(n, y+1)$. Variance calculation simply differs because based on the squared values of the same pixels.

While visiting both horizontal strips, `DeltaMean` and `DeltaVariance` arrays are initialized so that each element contains respectively the difference of values and difference of squared values between a pixel in the bottom strip and the corresponding pixel in the top strip. This is done to support the third incremental computation described in 15 and 17.

5.2 Pre-processing step: SIMD implementation

SIMD programming brings optimal results when applied to the final step of the preprocessing step because it allows for parallel mean and variance computations.

The first (*vertical*) incremental step consists in retrieving the sums of values and sums of squared values referred to the pixels $(x+i, y), i = 0 \ldots 7$ located on the previous row and correcting those quantities with the contributions $U_{s1}(x+i, y+1)$ and $U_{s2}(x+i, y+1)$, $i = 0 \ldots 7$ in formulas 9 and 12.

The second (*horizontal*) incremental step computes terms $U_{s1}(x+i, y+1)$ and $U_{s2}(x+i, y+1), i = 0 \ldots 7$ as a progressive correction of the same terms referred to the previous pixel. Formulas 11 and 14 show that the corrections are given by the pixels located at the corners of the window centered on the current pixel. With reference to one pixel of the eight processed in parallel, figure 4 shows that U_{s1} is given by $(\gamma - \beta) - (\delta - \alpha)$ while U_{s2} by the quantity $(\gamma^2 - \beta^2) - (\delta^2 - \alpha^2)$. The third and final incremental step is intended to compute and store terms $(\gamma - \beta)$ and $(\gamma^2 - \beta^2)$ so that they can be re-used as terms $(\delta - \alpha)$ and $(\delta^2 - \alpha^2)$ when processing the pixels $(x + (2n+1) + i, y+1)$, $i = 0 \ldots 7$.

The described process is slightly complicated by the need to store terms $(\gamma - \beta)$ in words and terms $(\gamma^2 - \beta^2)$ in double words, respectively reducing the parallelism to 4 and 2. The following MMX code shows how we compute terms $(\gamma - \beta)$. 8 γ values are read from the bottom-right corner of the correlation window and casted to words with the MMX instructions `punpcklbw` and `punpckhbw`, and the same is done for the 8 β values from the top-right corner. A pair of `psubsw` (packet subtract words with saturation) MMX instructions are finally used to compute in parallel the 4+4 $(\gamma - \beta)$ differences.

```
pxor      mm7, mm7
; mm7=0
; DnR: accessing 8 gamma values
movq      mm6, [esi+ecx]
; mm6 = pixel 0-7 from corner DnR (BYTES)
movq      mm5, mm6
; mm5 = mm6 copy
punpcklbw mm6, mm7
; mm6 = pixel 0-3 from corner DnR (WORDS)
punpckhbw mm5, mm7
; mm5 = pixel 4-7 from corner DnR (WORDS)
; UpR: accessing 8 beta values
movq      mm4, [esi]
; mm4 = pixel 0-7 from corner UpR (BYTES)
movq      mm3, mm4
; mm3 = mm4
punpcklbw mm4, mm7
; mm4 = pixel 0-3 from corner UpR (WORDS)
punpckhbw mm3, mm7
; mm3 = pixel 4-7 from corner UpR (WORDS)
; (gamma - beta) terms
psubsw    mm6, mm4
; mm6 = (0-3 DnR) - (0-3 UpR)
psubsw    mm5, mm3
; mm5 = (4-7 DnR) - (4-7 UpR)
```

5.3 Matching step

The *matching step* takes in input the normalised frames computed in the preprocessing step and generates the disparity map for the pair of original frames.

The complexity of the matching step is d_r (with $d_r = d_{max} + 1$) times higher than the complexity of the preprocessing step. Thus, the code optimisation carried out within the matching step have a strong impact on the overall execution time. The matching step is implemented according to the incremental computation scheme described in section 4, which yields a dramatic reduction of the operations executed at each pixel. Figure 5 shows which steps are taken to process the normalised left and right frames in order to establish matches between pixels in the left frame and pixels in the right frame.

The matching process consists in visiting each pixel (x, y) in the normalised left frame to evaluate the d_r terms $SAD(x, y, d)$, $d = 0 \ldots d_{max}$ indicating the error scores between (x, y) and the d_r pixels $(x + d, y)$, $d = 0 \ldots d_{max}$ located within the normalised right frame. The best match found for the current pixel (x, y) is identified by the smaller score:

$\texttt{SAD\_MIN} = SAD(x, \texttt{disparity}, y)$

The first processed pixel is located in the left normalized frame at coordinates $(2n, 2n)$. For each disparity value, two nested loops visit all pixels within the correlation window centered in $(2n, 2n)$ computing one term $SAD(2n, 2n, d)$ as indicated by formula 3. Each term is compared with the smaller previously computed so that, after considering all d_r disparity values, the best match for the pixel $(2n, 2n)$ is found. Similarly to the pre-processing step, the partial sums of absolute values associated with $2n + 1$ vertical strips shown in figure 5 must be stored to sustain the incremental computation required for the remaining pixels of the first row. The main difference between the two steps is that the matching step requires storing $(2n + 1) \cdot d_r$ terms instead of just $2n + 1$.

For a generic pixel $(x, 2n)$ in the first row of the left normalised frame, the d_r terms $SAD(x, 2n, d)$ are computed by adding to the terms $SAD(x - 1, 2n, d)$ previously stored, the corrections given by the difference between the sum of absolute values associated with the vertical strip on the right of the correlation window and those associated with the vertical left strip. The most interesting portion of code in our stereo algorithm is the matching process for a generic row, since it is heavily based on SIMD techniques. Following the same scheme adopted in the pre-processing step, the generic row is treated after initializing some data structures when processing the first pixel of the generic row. The d_r terms $SAD(2n, y + 1, d)$, $d = 0 \ldots d_{max}$ are computed as a correction of the terms $SAD(2n, y, d)$, $d \in [0..d_{max}]$ stored when processing the previous row, basing on the following relation:

$$SAD(2n, y + 1, d) = SAD(2n, y, d) + \sum_{j=2n-n}^{2n+n} T(j, d)$$
$$d \in [0..d_{max}] \tag{21}$$

where

$$T(j, d) = |L(j, y + n + 1) - R(j + d, y + n + 1)| - |L(j, y - n) - R(j + d, y - n)| \tag{22}$$

Figure 6 shows the case of $d = 6$. The correction is given by summing up the contribution of $2n + 1$ terms $T(j, 6)$, each of them being the difference of two absolute values. The first absolute value refers to the pixel j on the bottom horizontal strip in the left image and the pixel $j + 6$ on the corresponding strip in the right image. The second absolute value refers to pixels in same positions but on the top horizontal strips.

5.4 Matching step: SIMD implementation

A generic row at coordinate $y + 1$ is processed in the matching step with a main loop iterating one by one, each pixel of the row. Consider formula 19. To compute the first absolute value appearing in formula 19, we load an MMX register with 8 copies of the same left pixel $L(x + n, y + n + 1)$ and, in parallel, read the 8 pixels $R(x + d + n, y + n + 1)$, $d = 0 \ldots 7$ on the bottom of the correlation window. The following code, based on the *saturated arithmetic*, shows how the eight absolute values are computed.

```
; mm1 contains 8 copies of the
; DnR pixel in Left frame
movq    mm2, [esi+ebx]
; Read 8 DnR pixels from Right frame
movq    mm3, mm1
```

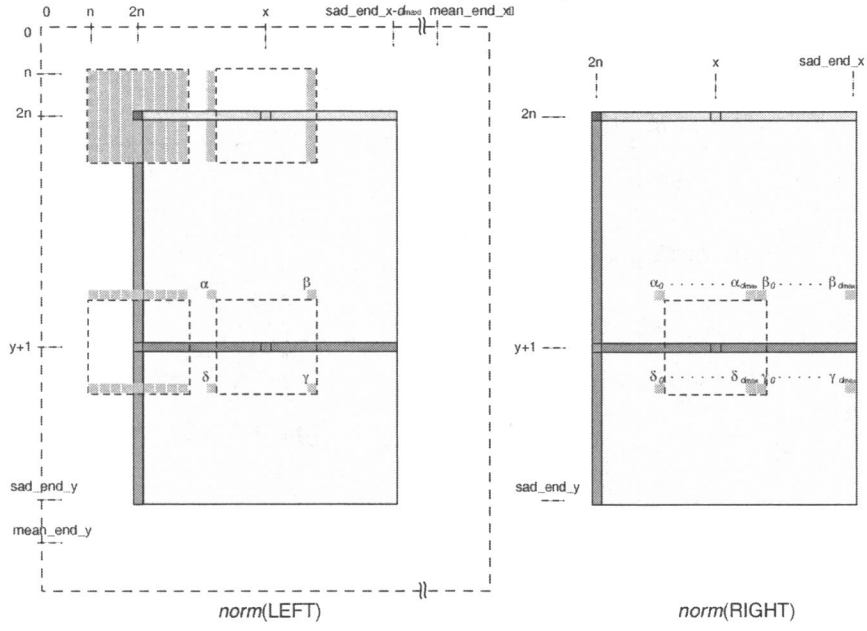

Figure 5. Pixels involved in the matching step.

```
; in mm3 8 copies of DnR pixel from Left
movq     mm4, mm1
;mm4 = 8 copies of DnR pixel from Left
; DnR absolute difference
psubusb mm3, mm2
; Left(DnR)-Right(DnR), negative terms to 0
psubusb mm2, mm4
; Right(DnR)-Left(DnR), negative terms to 0
por      mm3, mm2
; merge non zero terms (abs values) in mm3
```

The first `psubusb` (subtract unsigned packed bytes with saturation) instruction stores in `mm3` the positive differences between the 8 copies of the pixel in the left image and the 8 pixels read from the right image. The negative differences are instead converted to zero due to *saturated arithmetic*. The second `psubusb` instruction exchanges the factors of the packed difference by fact, saturating the terms previously positive and storing as positive, those previously saturated. The final `por` (packed or) instruction merges in `mm3` the positive results from the two computed differences providing the eight absolute values. Formula 19 is computed for the current pixel a total of d_r times evaluating d_r differences of absolute values and accessing d_r terms $T(\tilde{x}, d)$. Each time the mentioned formula is computed, the difference of absolute values is also used to replace the corresponding term $T(\tilde{x}, d)$ by fact, completing the inner level of incremental computation.

5.5 SIMD search for the best match

The d_r terms $SAD(x, y+1, d)$ computed for the current pixel $(x, y+1)$ represent d_r error scores between the current pixel and the $(x+d, y+1), d = 0 \ldots d_{max}$ pixels in the right image. The following MMX/SSE code shows how VSA searches the minimum SAD term within the d_r available.

```
movq mm0, [edx+ecx*8]
; mm0: current 4 SAD min (WORDS)
movq mm1, HiDisparities
; mm1: current 4 indexes
; dmax,dmax-1,dmax-2,dmax-3
movq mm6, Mask04040404
; mm6: mask to decrease indexes by 4
movq [edi], mm1
; [edi] holds indexes 4 SAD min

align 16
loopMinimi:
  movq     mm2, [edx+ecx*8-8]
  ; mm2: next 4 SAD values (WORDS)
  psubw    mm1, mm6
  ; mm1: next 4 indexes
  pminsw   mm0, mm2
  ; mm0: updates current 4 SAD min
```

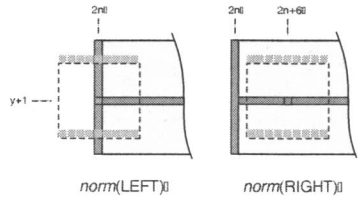

Figure 6. Matching for the first pixel of a generic row.

```
  pcmpeqw  mm2, mm0
  ; mask: FFFF=min changed, 0=no change
  maskmovq mm1, mm2
  ; update [edi] indexes where mm2=FFFF
loop loopMinimi
; jumps decreasing ecx by 1

movq mm1, [edi]
; mm1: disparities 4 SAD (pseudo)min
```

The search begins initializing `mm0` register with the 4 last terms $SAD(x, y+1, d), d = d_{max} - 3 \ldots d_{max}$ previously computed and `mm1` register with the disparity values $d_{max} - 3$, $d_{max} - 2$, $d_{max} - 1$ and d_{max} of the SAD terms in `mm0`. The memory locations pointed by the `edi` register are used to store the disparities of the smaller SAD terms currently found. The loop begins loading `mm2` register with the 4 SAD terms immediately preceding the previously accessed SADs and `mm1` register with the respective disparities.

The searching process is based on the SSE instruction `pminsw` (minimum of packed signed words) which updates the current 4 minimum SAD terms in `mm0` considering the values in `mm2`. The MMX instruction `pcmpeqw` (compare packed words for equality) is then used to build a bit mask identifying which term in `mm0` is changed. The mask is used with the SSE instruction `maskmovq` (store selected bytes of quadword) to selectively move in `[edi]` the disparities of the SADs terms recently moved in `mm0` by `pminsw` instruction.

At the end of the loop `mm0` contains 4 SAD terms one of which is the smallest of the d_r available while the other 3 are said *pseudo minima*. The name *pseudo minima* comes from the consideration that the parallel search method doesn't ensure that the three terms are the smaller SADs greater than the minimum, they only have good chances because of the continuity of the error function.

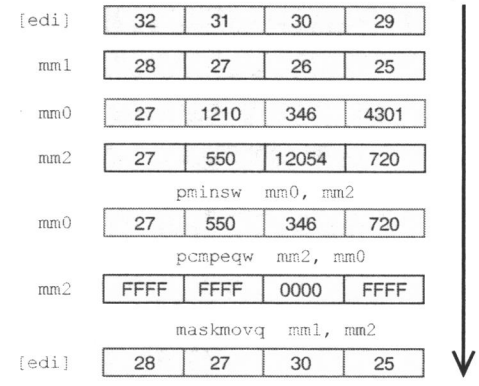

Figure 7. SSE minimum SAD search.

The portion of code following the search method is used to isolate the term `SAD_MIN` as the smallest SAD from the 4 terms in `mm0` and retrieving the corresponding `disparity`. At the same time we carry out *sharpness* and *distinctiveness* tests (2 and 1).

6 Experimental results

In this section we show the experimental results obtained with the proposed algorithm using the "Tsukuba" stereo pair, from University of Tsukuba (the left image of the stereo pair is shown in Figure 8). We also provide the results obtained with SVS 2.0 [3, 14], a well-known area-based algorithm based on two matching phase. Additional experimental results can be found at the website [1].

The "Tsukuba" stereo pair contains objects at different depths generating, several occlusions, as well as poorly-textured regions in the background, such as the wall at the top-right corner. Moreover, it also contains some specular regions (i.e. the face of the statue and some regions of the lamp) that could render quite difficult the stereo matching process. Comparing the output of our algorithm (top of Figure 9) with the ground truth image (bottom of Figure 8) we can observe first that the rough 3D structure has been clearly recovered: the camera an its trestle on the background have been recovered almost correctly as well as the objects closer to the stereo acquisition system, such as the statue and the lamp's head. Moreover, it is worth observing that the major occlusions have been discarded (since in

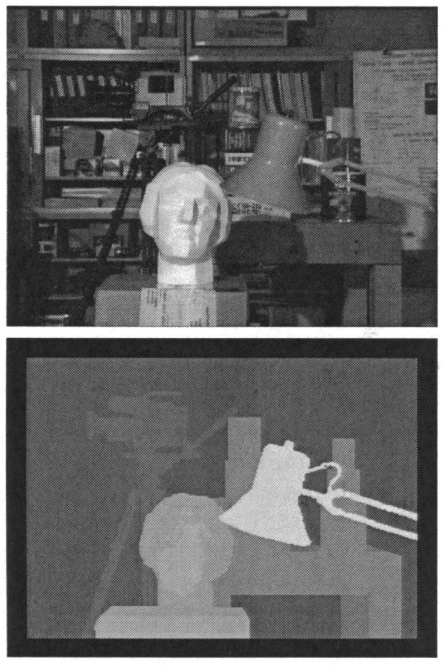

Figure 8. (Top) Left image of the "Tsukuba" stereo pair and (Bottom) ground-truth.

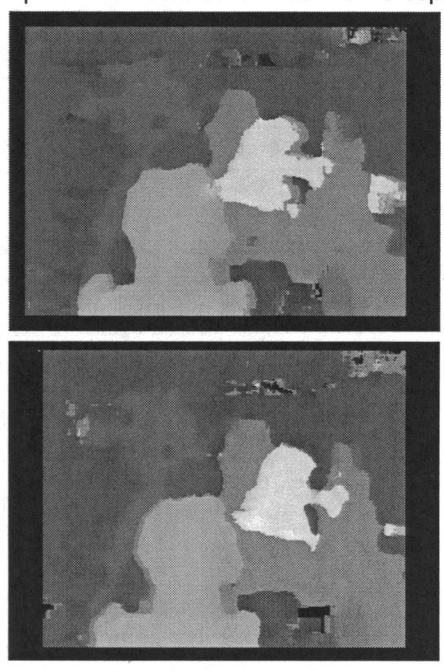

Figure 9. (Top) Disparity map computed with the proposed approach and (Bottom) with SVS 2.0.

Figure 9 the points left unmatched are represented in red, this can be seen clearly if the paper is printed with colours. However, some details such as for example the lamp's wire, the lamp's switch and the two roads that sustain the lamp, have vanished. Moreover the disparity map shows the *border-localisation* problem. This causes clearly an unaccurate fitting of the object's silhouette in the original image into the disparity map. These problems are inherent to local algorithms since they depend on the method adopted to establish correspondences, which relies on the use of a local support area. Local algorithms behave correctly when the correlation window covers a region at constant depth but are likely to produce artifacts when the correlation window covers regions at different depths. Some authors [15, 11] propose local algorithms aimed at reducing the *border-localisation* problem. However, it is worth pointing out that the results provided by these algorithms are still less accurate that those generated with global, slow algorithms such as [13].

The results obtained by SVS on the "Tsukuba" image pair (bottom of Figure 9) are very similar: the rough $3D$ structure is recovered and the major occlusions are correctly detected. This result show a similar behaviour between left-right constraint based algorithms and the proposed matching approach. Additional experimental results can be found at the web site [1].

Finally, we report in Table 1 some measurements aimed at assessing the speed of the two algorithms with different image sizes and disparity ranges. These measurements have been obtained on an Intel Pentium III processor running at 800 MHz. From Table 1 we can see that for a small disparity range (i.e. 16) SVS is always faster, much faster for small images (i.e. (320×240)) and slightly faster for bigger images. Yet, as the disparity range is increased, our algorithm gets faster than SVS, significantly faster for big images and large disparity range.

7 Conclusion

We have presented an area-based stereo matching algorithm which relies only on a single matching phase and that detects unreliable matches by detecting violations of the uniqueness constraint. Further reliability

Alg. (size)/d	16	32	48	64
P.A. (320 × 240)	39.59	31.25	27.44	25.94
SVS (320 × 240)	57.99	33.68	20.49	15.31
P.A. (640 × 480)	8.94	6.92	5.77	5.17
SVS (640 × 480)	11.99	5.93	4.07	3.18
P.A. (800 × 600)	5.56	4.28	3.60	3.18
SVS (800 × 600)	6.96	3.65	2.53	1.94
P.A. (1024 × 768)	3.32	2.56	2.09	1.86
SVS (1024 × 768)	3.79	2.07	1.45	1.06

Table 1. Speed measurements (in terms of frame per second fps) for the proposed algorithm (P.A.) and SVS 2.0.

improvement is achieved, at a low computational cost thanks to the use of parallel SIMD programming, by constraining the behavior of the error scores. Elimination of redundant operations is based on efficient calculation schemes and deploys a further level of recursion aimed at avoiding redundant computations that take place within the correlation window. We have provided a detailed description of the parallel mapping of the algorithm onto a general purpose processors with SIMD capabilities. We have shown the experimental results obtained on a stereo pair with ground truth and compared these results with the available ground-truth and with those obtained by a well-known algorithm based on the left-right constraint. Finally, we have provided some speed measurements showing that the algorithm runs remarkably fast on a standard Personal Computer.

References

[1] *Exp. results.* `www.vision.deis.unibo.it/~smattoccia`.

[2] *Point Grey Research.* World Wide Web, `http://www.ptgrey.com`, 2001.

[3] *Videre Design.* World Wide Web, `http://www.videredesign.com`, 2001.

[4] S. Changming. A fast stereo matching method. In *Digital Image Computing: Techniques and Applications*, pages 95–100, Auckland, Nov. 1997.

[5] P. Corke, P. Dunn, and J. Banks. Frame-rate stereopsis using non-parametric transforms and programmable logic. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, volume 3, pages 1928 –1933, Detroit, Michigan, May 1999.

[6] L. Di Stefano, M. Marchionni, S. Mattoccia, and G. Neri. Dense stereo based on the uniqueness constraint. In *16th. Int. Conference on Pattern Recognition*, Quebec City, Canada, Aug 2002.

[7] L. Di Stefano and S. Mattoccia. Real-time stereo within the VIDET project. *Real-Time Imaging*, 8(5):439–453, Oct 2002.

[8] O. Faugeras et al. Real-time correlation-based stereo: algorithm, implementation and applications. INRIA Technical Report n. 2013, 1993.

[9] P. Fua. Combining stereo and monocular information to compute dense depth maps that preserve depth discontinuities. In *12th. International Joint Conference on Artificial Intelligence*, pages 1292–1298, Sydney, Aug. 1991.

[10] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22, 2000.

[11] H. Hirschmuller, P. Innocent, and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *Int. Journal of Computer Vision*, 47(1-3):229–246, 2002.

[12] T. Kanade, H. Kato, S. Kimura, A. Yoshida, and K. Oda. Development of a video-rate stereo machine. In *Proc. of International Robotics and Systems Conference (IROS '95)*, volume 3, pages 95 – 100, August 1995.

[13] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusions using graph cuts. In *Proceedings of International Conference on Computer Vision*, 2001.

[14] K. Konolige. Small vision systems: Hardware and implementation. In *8th Int. Symposium on Robotics Research*, pages 111–116, Hayama, Japan, 1997.

[15] M. Okutomi, Y. Katayama, and S. Oka. A simple stereo algorithm to recover precise object boundaries and smooth surfaces. *Int. Journal of Computer Vision*, 47(1-3):261–273, 2002.

[16] A. Peleg and U. Weiser. Mmx Technology Extension to the Intel Architecture. *IEEE Micro*, 16(4):42–50, Aug. 1996.

[17] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Microsoft Research Technical Report MSR-TR-2001-81, 2001.

[18] T. Shreekant and T. Huff. Implementing streaming simd extensions on the pentium III processor. *IEEE Micro*, 20(4):47 –57, July 2000.

[19] J. Woodfill and B. Von Herzen. Real-time stereo vision on the parts reconfigurable computer. In *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, Apr. 1997.